GlobalPlatform

Card Specification

Version 2.1

June 2001

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 three leading technologies in this area are Java Card™, Windows® Powered Smart Cards, 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.

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.

Open Platform defines a flexible and powerful specification for Card Issuers to create 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 Open Platform Specifications that shall be implemented on Open Platform 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 Open Platform 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
CAMS v3.0 June 2000	Multi Application Smart Card Management Systems GlobalPlatform Functional Requirements
FIPS PUB 46-3	Federal Information Processing Standards Publication 46-3, Reaffirmed 1999: Data Encryption Standard (DES): U.S. Department Of Commerce/National Institute Of Standards And Technology
FIPS PUB 180-1	Federal Information Processing Standards Publication 180-1, 1995: Secure Hash Standard: U.S. Department Of Commerce, Technology Administration, National Institute Of Standards And Technology
ISO/IEC 7816-4:1995	Identification cards – Integrated circuit(s) cards with contacts - Part 4: Inter-industry commands for interchange
ISO/IEC 7816-5:1994	Identification cards – Integrated circuit(s) cards with contacts - Part 5: Numbering systems and registration procedure for application identifiers
ISO/IEC 7816-6: 1996	Identification cards- Integrated circuit(s) cards with contacts- Part 6: Interindustry data elements
ISO 8731-1:1987	[IS8731-1] Banking - Approved algorithms for message authentication – Part 1: DEA
ISO/IEC 9797:1994	[IS9797] Information technology – Security techniques - Data integrity mechanism using a cryptographic check function employing a block cipher algorithm
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
Java Card™ 2.1.1	Go to the following web site for Java Card™ documentation: www.javasoft.com
PKCS#1 (RFC 2437)	PKCS #1 v2.0: RSA Cryptography Standard, RSA Laboratories, October 1998 (RFC 2437).
Windows®-Powered Smart Cards	Go to the following web site for more information on Windows for Smart Cards®: www.microsoft.com

Table 1-1: Normative References

1.3 Terminology and Definitions

Table 1-1 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.

Term	Definition
Application	Instance of an Executable Module after it has been installed and made selectable
Application Extradition	Process that allows an Application associated with a Security Domain to be associated with another Security Domain

Term	Definition
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 during a Card Session starting with the Application selection and ending with Application de-selection or termination of the Card Session
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
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 OP card
Cardholder	The end user of a card
Card Issuer	Entity that owns the card and is ultimately responsible for the behavior of the card
Card Manager	Generic term for the 3 card management entities of an Open Platform card i.e. the Open Platform Environment, Issuer Security Domain and the Cardholder Verification Method Services provider
Card Recognition Data	Information that tells an external system, in particular a Card and Application Management System (CAMS), how to work with the card (including indicating that this is an OP card)
Card Session	The link between the card and the external world starting with the ATR and ending with a subsequent reset or a deactivation 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 Verification Method (CVM)	A method to ensure that the person presenting the card is the person to whom the card was issued
Controlling Authority	The Controlling Authority has the privilege to keep the control over the Card Content through the use of the Mandated DAP
DAP Block	Part of the Load File used for ensuring Load File Data Block verification
Digital Signature	An asymmetric 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

Term	Definition
Host	A logical term used to represent the back end systems that support the Open Platform 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	On-card entity providing support for the control, security, and communication requirements of the Card Issuer
Life Cycle	The existence of Card Content on an Open Platform card and the various stages of this existence where applicable
Life Cycle State	A specific state within the Life Cycle of the card or of Card Content
Load File	A file transferred to an Open Platform 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) and libraries or 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
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 Platform Environment	The central on-card administrator that owns the Open Platform Registry
Open Platform Registry	A container of information related to Card Content management
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
Retry Limit	The maximum number of times an invalid CVM value can be presented prior to the CVM handler prohibiting further attempts to present a CVM value
Retry Counter	A counter, used in conjunction with the Retry Limit, to determine when attempts to present a CVM value shall be prohibited
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 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 the Application Provider
Symmetric Cryptography	A cryptographic technique that uses the same secret key for both the originator's and the recipient's transformation

Table 1-2: Terminology and Definitions

1.4 Abbreviations and Notations

Abbreviation	Meaning
AID	Application Identifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
ATR	Answer-to-Reset
BER	Basic Encoding Rules
CIN	Card Image Number / Card Identification Number
CLA	Class Byte of the Command Message
CVM	Cardholder Verification Method
DAP	Data Authentication Pattern
DES	Data Encryption Standard
EMV	Europay, MasterCard, and Visa; used to refer to the ICC Specifications for Payment Systems
FCI	File Control Information
IC	Integrated Circuit
ICC	Integrated Circuit Card
IIN	Issuer Identification Number
INS	Instruction Byte of 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
OPEN	Open Platform Environment
P1	Parameter 1
P2	Parameter 2
P3	Parameter 3
PIN	Personal Identification Number
RAM	Random Access Memory
RFU	Reserved for Future Use
RID	Registered Application Provider Identifier
ROM	Read-only Memory
RSA	Rivest / Shamir / Adleman asymmetric algorithm
SW	Status Word
SW1	Status Word One
SW2	Status Word Two
TLV	Tag Length Value

Table 1-3: Abbreviations and Notations

1.5 Revisions History

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

This section provides a brief reminder 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 *OPEN PLATFORM 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 Open Platform Card Specification V2.1

The following major adjustments are the modifications decided by GlobalPlatform Members. All of these modifications are intended to make Open Platform 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 Open Platform 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 (see Section 6.7.6.1 - *Load File Data Block Hash*).

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 (see Section 9.5 - *INSTALL Command*).

1.5.2.3 Security Domain Association and Extradition

In the previous Open Platform Card Specification an Executable Load File was associated to a Security Domain and all Applications instantiated from an Executable Load file, when installed, were also associated to 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 (see Section 7.6.3 - *Delegated Extradition*).

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 Open Platform Registry at the time that the Executable Load File is registered (see Section 9.5 - *INSTALL Command*).

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 (see Section 5.2.2 - *Executable Module Life Cycle*).

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 Open Platform 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 Open Platform Card Specification. Information such as: this is an Open Platform card, implemented in a particular way, and with a particular version number is now available (see Section 6.8.3 - *Card Recognition Data*).

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 Open Platform 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 (see Appendix D - *Secure Channel Protocol '01'*and E - *Secure Channel Protocol '02'*).

1.5.2.7 Cardholder Verification Method Services

Additional services and features regarding the optional CVM have been included (see Section 6.9 - *CVM Management*). 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. (See Section 3.2 - *Card Manager*) While the term Card Manager is still present, it now encompasses the Open Platform Environment, 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 Open Platform API for Windows Powered Smart Card is now included in Appendix A.3 - *Open Platform on Windows Powered Smart Card.*

1.5.2.10 Java Card API

Taking into account the various new features of the Open Platform a new API (See Appendix A.2 - *Open Platform on a Java Card)* 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 Open Platform Card Specification only contains generic Open Platform descriptions. All information specific to a particular implementation has been positioned in a set of Appendices.

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.

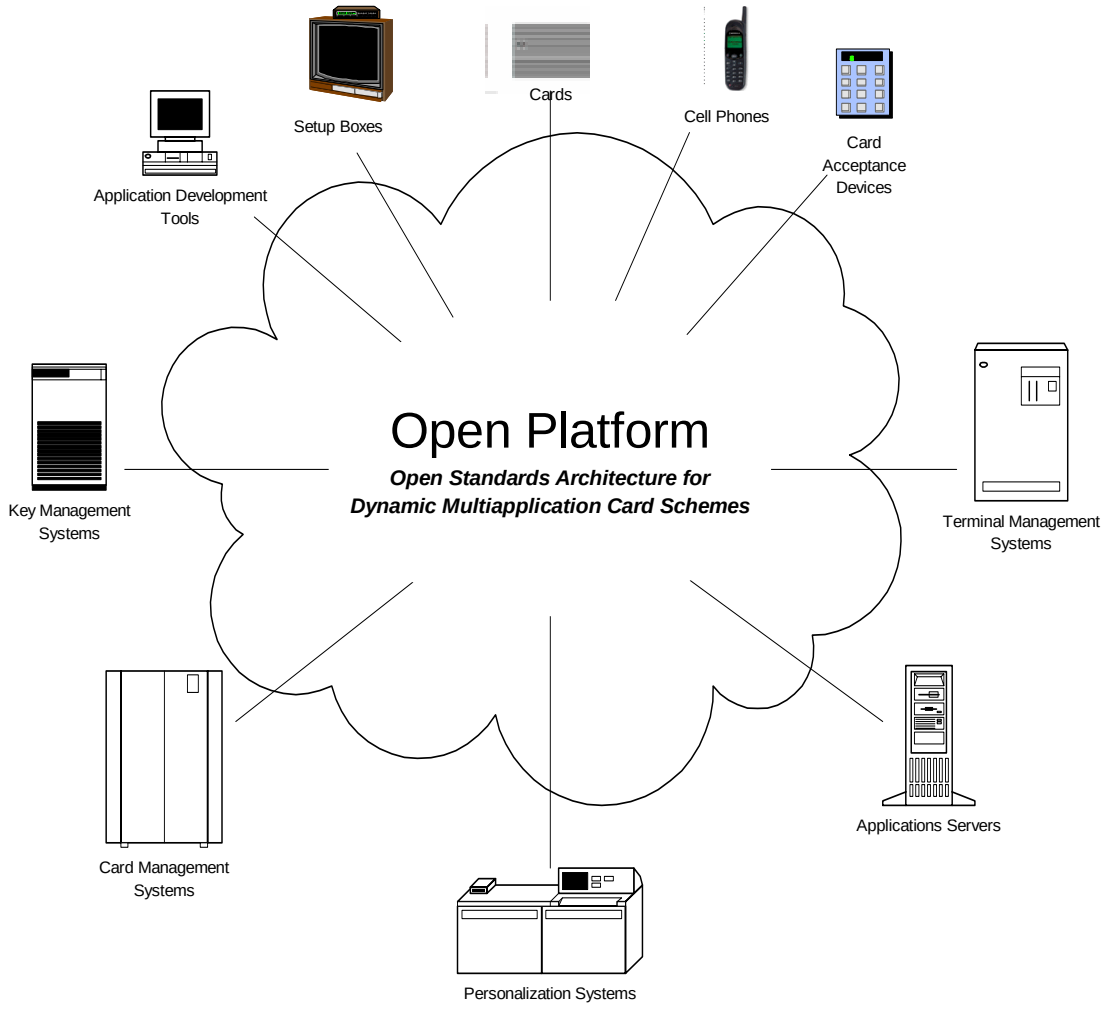


Figure 2-1: Open Platform architecture

The Open Platform architecture is designed to provide Card Issuers with the system management architecture for managing these smart cards. Although Open Platform 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.

The Open Platform 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 the Open Platform, 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 Open Platform 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 an application from the Card Issuer as well an application from one of the business partners of the Card Issuer referred to as an Application Provider.

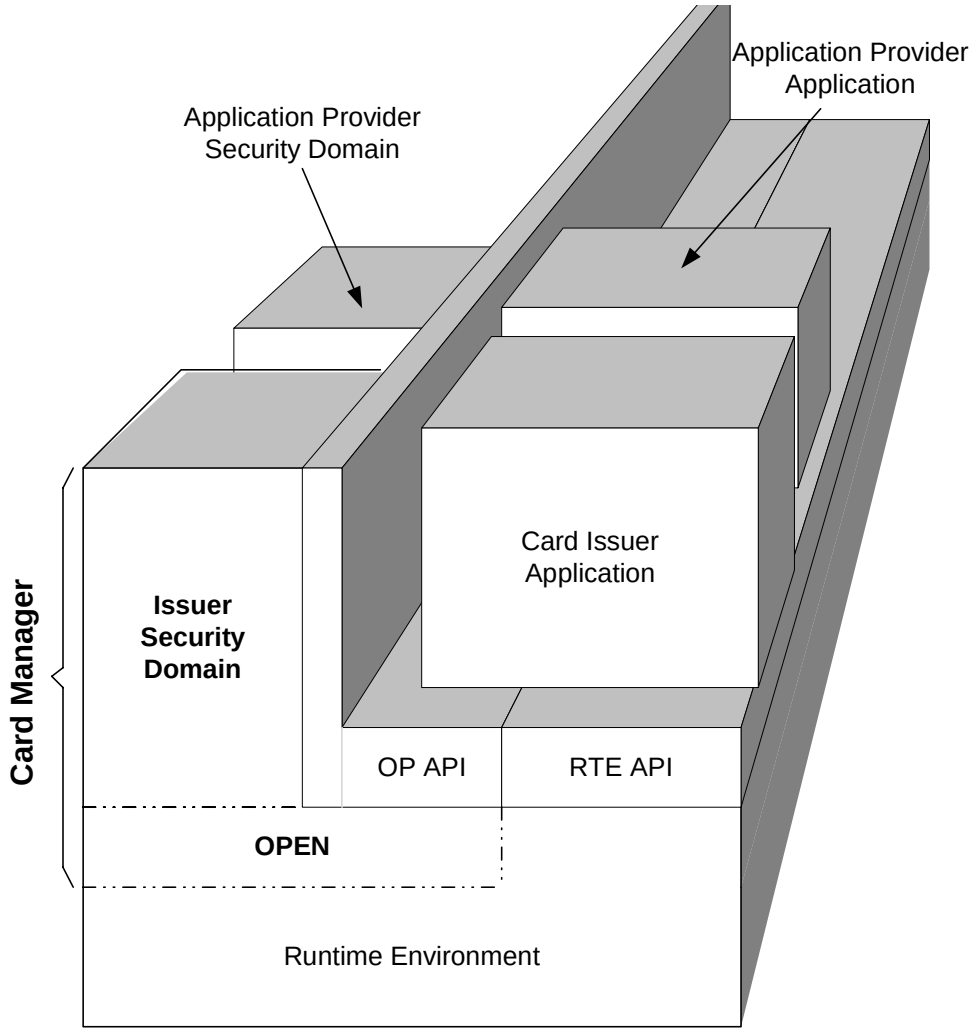


Figure 3-1: Open Platform Card Architecture

All applications shall be implemented in a secure runtime environment that includes a hardware-neutral Application Programming Interface (API) to support application portability. The Open Platform does not mandate a specific runtime environment technology. The Card Manager is the primary Open Platform card component that acts as the central administrator for an Open Platform card. Special key and security management applications called Security Domains are created to ensure complete separation of keys between the Card Issuer and multiple Application Providers.

3.1 Runtime Environment

The Open Platform 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.

3.2 Card Manager

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

Some of these responsibilities are the same as, or very close to, those typically performed by the card runtime environment and are described in Section 3.2.1 - *Open Platform Environment (OPEN).*

Another major responsibility of the Card Manager is to be the on-card representative of the Card Issuer. Section 3.2.2 - *Issuer Security Domain* details this responsibility.

The Card Manager can be viewed as three entities:

- The Open Platform Environment,
- The Issuer Security Domain, and
- The Cardholder Verification Methods

These three entities are either incorporated as one or each can be viewed as a distinct separate entity.

See Section 6.1 - *Card Manager Overview* for a detailed description of the functions and responsibilities of the Card Manager.

3.2.1 Open Platform Environment (OPEN)

The main responsibilities of the Open Platform Environment (OPEN) are to provide an API to Applications, command dispatch, Application selection, and Card Content management. These functions are available, if not provided by the runtime environment or 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.

The OPEN also manages the installation of applications loaded to the card. The OPEN is responsible for enforcing security principles defined for content loading and installation. These principles encompass verification of the application code and that authorization to load and/or install has been provided by the Card Issuer.

Another important function provided by the OPEN is APDU command dispatching and application selection. When a SELECT command is received, 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 OPEN owns and uses an internal Open Platform Registry as an information resource for Card Content management. The Open Platform Registry contains information for managing the card, Executable Load Files, applications, Security Domain associations, and privileges.

3.2.2 Issuer Security Domain

The Issuer Security Domain, as the mandatory on-card representative of the Card Issuer, has the capability of loading, installing, and deleting applications that belong either to the Card Issuer or to other Application Providers. In this and most other aspects it is very similar to any other Security Domain (see Section 3.3 - *Security Domains)* on the card.

3.2.3 Cardholder Verification Management

The Card Manager may provide services related to Cardholder Verification (see section 6.9 - *CVM Management).*

3.3 Security Domains

Just as the Issuer Security Domain is the on-card representative of the Card Issuer, an Application Provider referred to simply as a Security Domain in this Specification is the on-card representative of an Application Provider or Controlling Authority. 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 a Security Domain as its on-card representative.

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

The Security Domain has application characteristics such as application AID, application privileges, and Life Cycle State (the Issuer Security Domain inherits the Life Cycle State of the card). An example of the Security Domain functioning as an Application is when the Security Domain is selected in order to load a new application to the card.

Each Security Domain implements a Secure Channel Protocol defining the security applied during communication between the Card Issuer, Application Provider or Controlling Authority and its on-card Security Domain.

The Security Domain also provides an interface for Applications to access the Security Domain's services. It has a well-defined external APDU interface to ensure that all Security Domain implementations behave consistently and can be managed identically by the same off-card management systems. As the majority of these services and APDU commands are related to Card Content management, the Security Domain is closely interacting with the OPEN.

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.

3.4 Open Platform API

The Open Platform 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 an example of an implementation of the Application Programming Interface (API) on a Java Card™, see appendix A.2

For an example of an implementation of the Application Programming Interface (API) on a Windows Powered Smart Card®, see appendix A.3

3.5 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. Open Platform is intended to support multiple Executable Load Files and multiple Executable Modules and as such multiple Applications may co-exist on an Open Platform card.

Figure 3-2 represents the relationship between an Executable Load File, an Executable Module and an Application.

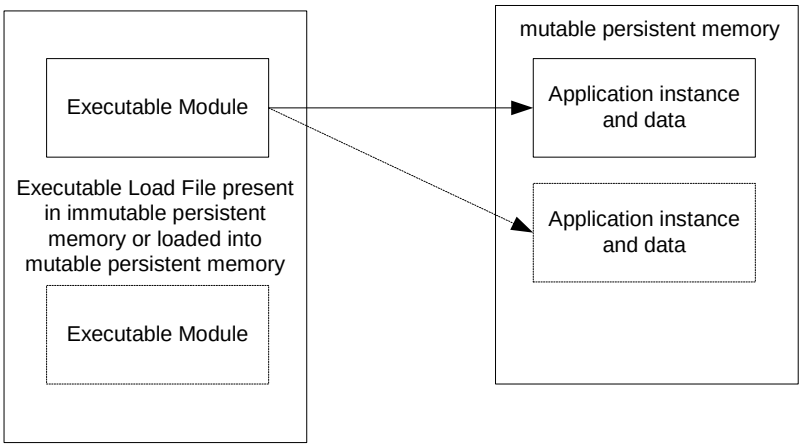


Figure 3-2 Card Content Relationships

4. Security Architecture

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

This section 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, and
- The cryptographic support provided by Open Platform

4.1 Goals

The primary goal of the Open Platform 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
- The Security Domains,
- The Applications

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

- Data integrity,
- Resource availability,
- Confidentiality, and
- 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 Open Platform 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

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, Application 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. (See Section 6.4.3 - *Content Extradition* for a description of the Delegated Management).

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 application code signatures according to the Application Provider's security policy,
- Obtaining pre-authorization for load, install, and extradition from the Card Issuer, and
- Returning Load, Install, Delete, and Extradition Receipts, 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, and
- Providing application code signatures according to its own defined security standards and policies to Card Issuers and Application Providers.

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 as well as the runtime environment itself 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

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

4.2.4.2 OPEN Security Requirements

The OPEN shall:

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

4.2.4.3 Issuer Security Domain Security Requirements

The Issuer Security Domain enforces the security policies of the Card Issuer.

The Issuer Security Domain shall:

- Communicate with off-card entities in accordance with the Card Issuer's security policy in pre-issuance and post-issuance,
- Manage card data,
- Be able to provide cryptographic protection services for its own Applications during their personalization,
- Verify the Card Issuer authorization for Card Content changes initiated by a Security Domain other than the Issuer Security Domain,
- Request the OPEN to load, install, extradite, and delete applications and,

- Generate Load, Install, Extradition, and Delete Receipts when required by the Card Issuer's security policy.

4.2.4.4 CVM Handler Security Requirements

The CVM handler shall:

- Be able to provide CVM services to Applications, such as verifying incoming CVM data,
- Hold the CVM data securely, and
- Perform internal velocity checks on the CVM to prevent card and Application access violations (see Section 6.7.4 - *Operational Velocity Checking*).

4.2.4.5 Security Domains Security Requirements

The Security Domains enforce the security policies of an Application Provider or Controlling AuthoritySecurity Domains other than the Issuer Security Domain shall:

- Communicate with off-card entities in accordance with the Application Provider's security policy,
- Be able to provide cryptographic protection services for their associated Applications,
- Verify the application signature when requested by the OPEN.

Security Domains authorized by the Card Issuer to perform Card Content changes shall:

- Request the OPEN to load, install, extradite, and delete applications and,
- Return to the off-card entity any Load, Install, Extradition, and Delete Receipts when provided by the Issuer Security Domain.

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 the Open Platform Card Specification.

4.3 Cryptographic support

One of the major requirements for an Open Platform 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. The Open Platform card should support symmetric cryptography such as the Data Encryption Standard (DES) algorithm. The Open Platform card may also support asymmetric cryptography such as the Rivest / Shamir / Adleman (RSA) algorithm.

The following cryptographic services are described in this chapter:

- 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 Integrity and Authentication for 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 choice of cryptographic algorithms for integrity and authentication is assumed to be industry and product specific.

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 an Open Platform card. Its intention is to provide a method for the OPEN to verify the integrity of the Load File Data Block. (See Section 6.4.1.1 - *Card Content Loading* for further details.)

The Load File Data Block Hash also has two other functions:

- It is used in the computation of the Load File Data Block Signature (see Section 4.3.1.2 - *Load File Data Block Signature*).
- It is included in the computation of the Load Token (see Section 4.3.1.3 - *Delegated Management Tokens*)

4.3.1.2 Load File Data Block Signature

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

The Delegated Management Tokens are signatures of one or more Delegated Management functions (loading application code, installing Applications and extraditing Applications) generated by the Card Issuer and used to provide the Card Issuer the control over these Card Content changes. Tokens are required when the Issuer Security Domain is not managing the Card Content changes itself. The Issuer Security Domain shall verify Tokens.

4.3.1.4 Receipts

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

4.3.2 Secure Communication

An Open Platform 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.

An Open Platform card offers the following security services associated with messages and defined within a Secure Channel Protocol (see Chapter 8 - *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; and,
- **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 Open Platform defines Life Cycle models to control the functionality and security of the following Open Platform components:

- Card,
- Executable Load Files,
- Executable Modules and
- Applications.

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

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

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 an Open Platform 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.

5.1.1 Card Life Cycle States

The following card Life Cycle States shall apply:

1. OP_READY
2. INITIALIZED
3. SECURED
4. CARD_LOCKED
5. TERMINATED

The Card Life Cycle states OP_READY and INITIALIZED are intended for use during the pre-issuance phases of the card life cycle.

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 Cycle.

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 Default Selected Application,
- Executable Load Files that were included in immutable persistent memory shall be registered in the Open Platform Registry,
- An initial key shall be available within the Issuer Security Domain.

The card shall be capable of Card Content changesFor Applications expected to be available at card issuance, 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:

- Security Domains other than Issuer Security Domain 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.

The card shall be capable of Card Content changes.

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 is used by the OPEN to enforce the Card Issuer's security policies related to post-issuance card behavior such as application loading, installation and activation. The state transition from INITIALIZED to SECURED is irreversible.

The card shall be capable of Card Content and Application content changes.

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 Card Issuer with the capability to disable Security Domain and Applications functionality. The Card Life Cycle state transition from SECURED to CARD_LOCKED is reversible.

Setting the card to this state means that the card shall no longer function except via the Issuer Security Domain.

Card Content changes are not allowed in this case.

Either the OPEN, or an Application on the card with the relevant privilege (see Section 6.6.2.4 - *Application Privileges*), or an off-card entity authenticated by the Issuer Security Domain 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. When in the state TERMINATED, all APDU commands shall be routed to the Issuer Security Domain and the Issuer Security Domain shall only respond to the GET DATA command .

The state TERMINATED shall be used to permanently disable all card functionality including most of the functionality of the Issuer Security Domain itself. 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.

The OPEN itself, an Application on the card with the relevant privilege (see Section 6.6.2.4 - *Application Privileges*) or an off-card entity authenticated by the Issuer Security Domain may initiate the transition from any of the previous states to the state TERMINATED.

5.1.2 Card Life Cycle 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.

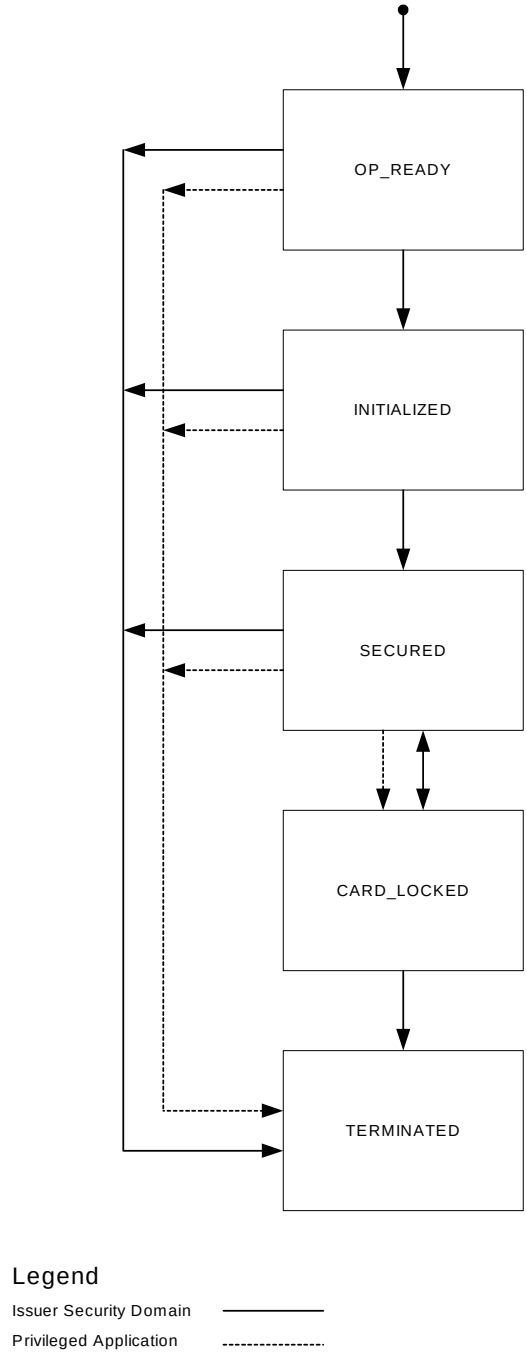


Figure 5-1: Card Life Cycle State Transitions

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 Open Platform 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 Open Platform Registry following the successful completion of the load process. Executable Load Files present in immutable persistent memory shall automatically have entries within the Open Platform Registry and 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 Open Platform Registry of the Executable Load File and each Executable Module within the Executable Load File shall also be removed, and the card shall not keep any records of the Executable Load File's or Executable Module's previous existence.

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 Open Platform Registry and the OPEN sets the Application Life Cycle State to the initial state of INSTALLED during the Application installation process initiated by the Issuer Security Domain or the Security Domain associated with the Executable Load File. The OPEN is also responsible for making the Application available for selection by setting its Life Cycle to SELECTABLE upon request during the Application installation process. While the OPEN actually performs these transitions, they may be initiated from a Security Domain with the Delegated Management privilege.

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.

The OPEN owns and maintains the Life Cycle State information within the Open Platform Registry. The Issuer Security Domain may inform off-card entities of Life Cycle States (see Section 9.4 - *GET STATUS Command*).

5.3.1 Application Life Cycle States

This Specification defines the following Application Life Cycle States:

1. INSTALLED
2. SELECTABLE
3. 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 State. It may use any Application specific states as long as these do not conflict with the states already defined by Open Platform. 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.

Note: Applications using the deprecated API should be aware that when converting to the Open Platform API the OPEN no longer enforces Application specific Life Cycle State transitions.

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 Open Platform Registry and this entry is accessible to authenticated off-card entities. 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 or the off-card entity authenticated by the Issuer Security Domain 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 authenticated by the Issuer Security Domain 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 Issuer Security Domain 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 Open Platform Registry is also removed, 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.

Figure 5-2illustrates 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.

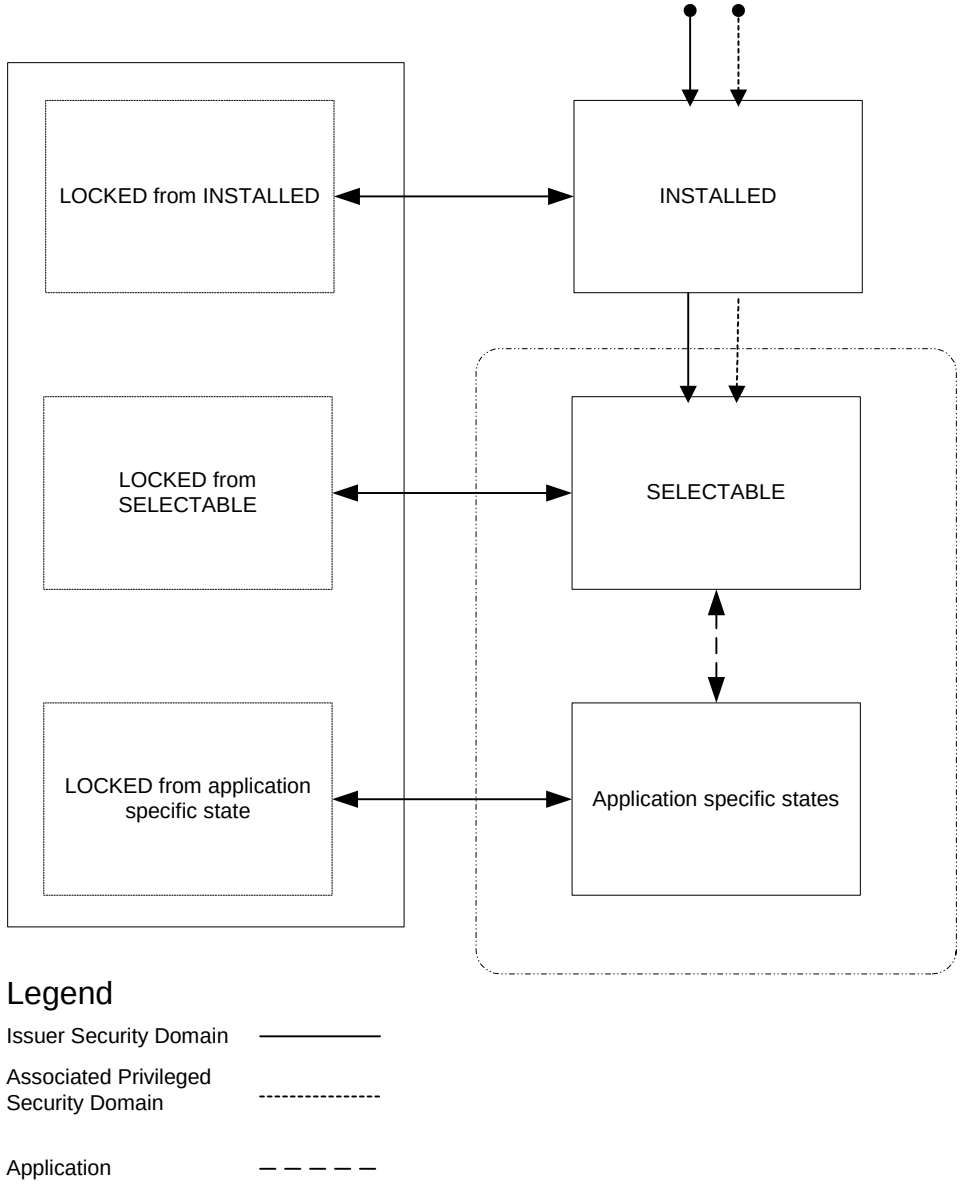


Figure 5-2: Application Life Cycle State Transitions

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 Open Platform Registry and this entry is accessible to authenticated off-card entities. 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 or the off-card entity authenticated by the Issuer Security Domain 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 authenticated by the Issuer Security Domain 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 but does not have any required effect on the access to that Security Domain's services by Applications through the OPEN. However, Security Domains do have the option to modify their own behavior to prohibit or restrict their services while locked.

Once the Life Cycle State is LOCKED, only the Issuer Security Domain 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 Open Platform Registry is also removed, and the OPEN is not required to maintain a record of the deleted Security Domain's previous existence.

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.

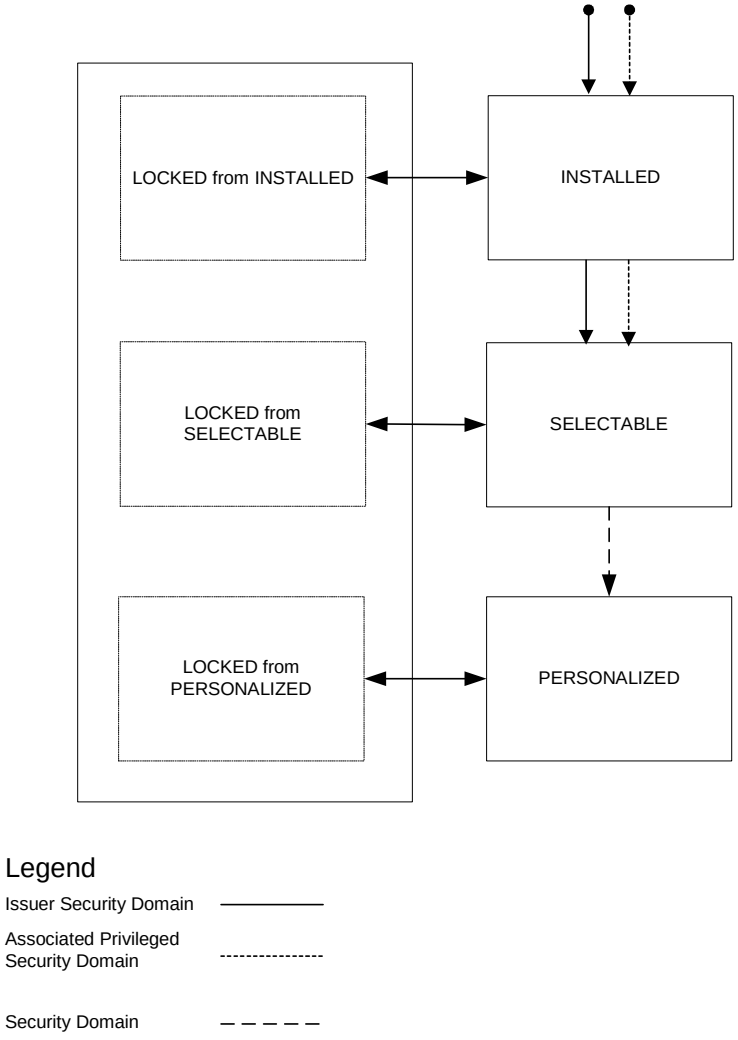


Figure 5-3: Security Domain Life Cycle State Transitions

5.4 Sample Life Cycle Illustration

This section provides a description of a sample Open Platform 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 Card Life 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 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 State SECURED. Application E is used until the card is terminated and as long as the Card Life 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 State SECURED. Application F is deleted some time during the card's lifetime.

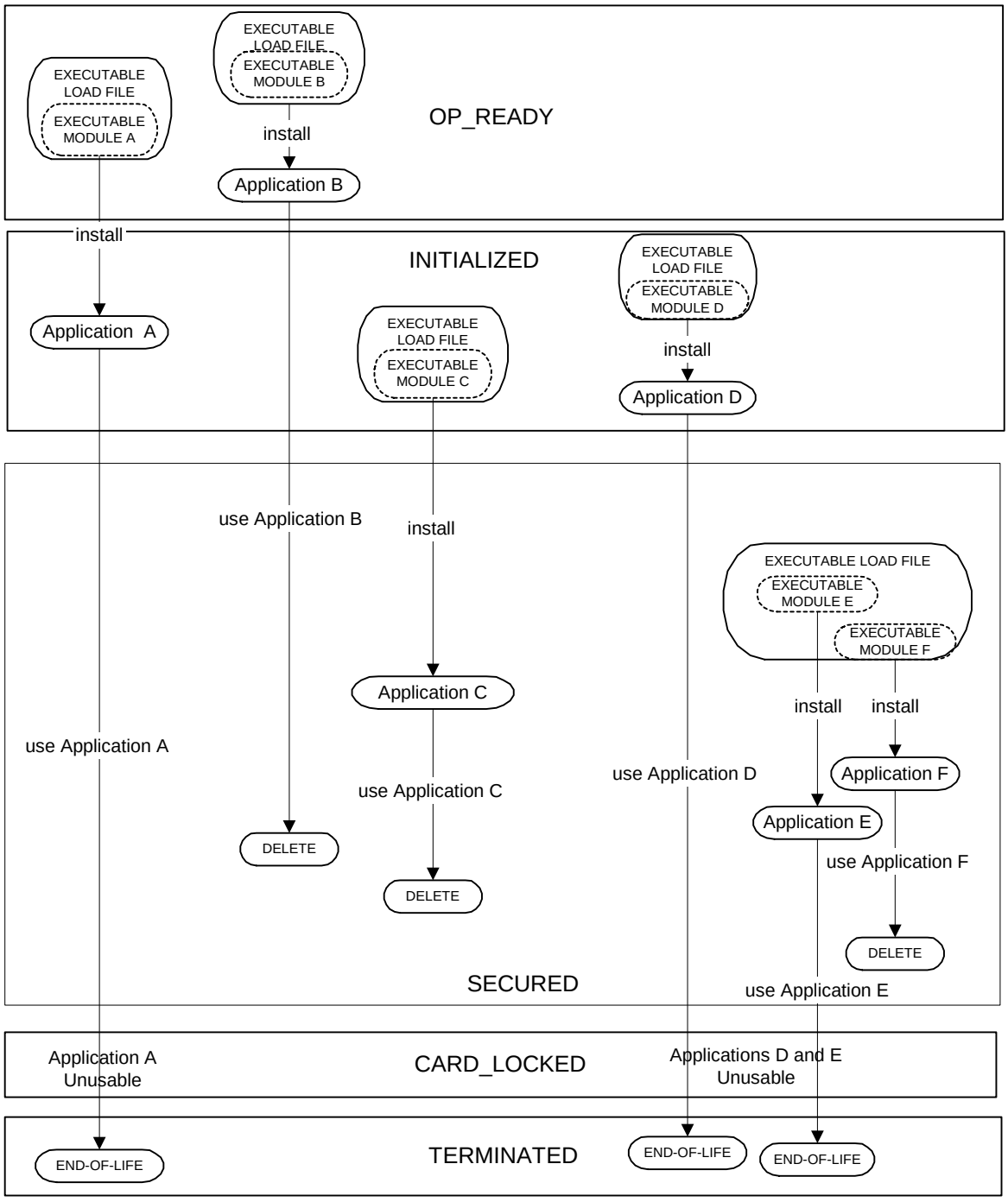


Figure 5-4: Sample Card Life Cycle and Application Life Cycles

6. Card Manager

6.1 Card Manager Overview

The Card Manager is the card component responsible for all card administration and card system service functions.

Section 3.2 - *Card Manager* provides a general overview of the Card Manager, the OPEN, the Issuer Security Domain and the CVM handler.

6.1.1 OPEN

The OPEN supports the following functions if the underlying runtime environment does not support them:

- **Command Dispatch**

  Application selection

  Command dispatching

- **Card Content Management**

  Content verification

  Content loading

  Content installation

  Content REMOVAL

- **Security Management**

  Security Domain locking

  Application locking

  Card locking

  Card termination

  Application privilege usage

  Security Domain privilege usage

  Tracing and event logging

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

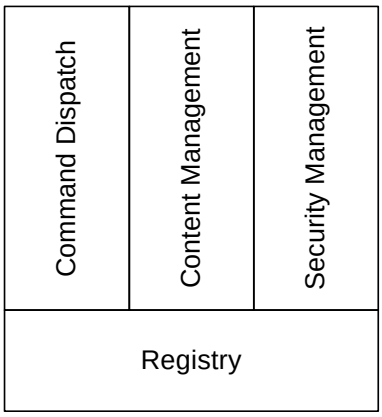


Figure 6-1: OPEN Architecture

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

Command Dispatch:

- Determine if an Application is present and available to respond to a SELECT command,
- Dispatch the 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 Card Issuer to audit the Card Content by retrieving status information related to any Application present on the card,
- Verify and request verification of the authenticity for Executable Load Files,
- Verify that Card Content functions are being initiated by Security Domains with the Delegated Management privilege,
- Verify that resource restrictions are respected during loading and installing of new content and during Application runtime execution,
- Verify an Application's accessibility to functionality that requires privileges.

The OPEN shall verify the Application privileges before accessing the CVM functionality.

Accessing the OPEN functions by an off-card entity is achieved through the Issuer Security Domain APDU interface and for on-card applications through an API. An Application shall use the OPEN API to communicate with the OPEN from within the card. However, the OPEN retains control of the services and determines on an individual request basis whether or not to grant each request.

6.1.2 Issuer Security Domain

The Issuer Security Domain, as implied by its name, has functionality very similar to a Security Domain and in addition to the functionality described herein (Chapter 6 - *Card Manager*), most of the functionality defined in Chapter *7 - Security Domains* also applies to the Issuer Security Domain. The 2 features that may apply to a Security Domain but do not apply to the Issuer Security Domain are Delegated Management and DAP Verification.

The Issuer Security Domain supports the following functions:

- Secure Communication for Card Content management
- Secure communication support for Applications during:
  - Application personalization
  - Application runtime

The Issuer Security Domain is responsible for the management of its data (e.g. the card's Issuer Identification Number).

The APDU interface, through the Issuer Security Domain, exists primarily to manage card administration functions, and in that capacity only the authenticated Card Issuer is allowed to access this interface through the Issuer Security Domain.

The API is the interface that Applications use to request services from the Issuer Security Domain. Only Applications associated with the Issuer Security Domain are allowed to use the Issuer Security Domain API services from within the card.

6.1.3 CVM Handler

The CVM handler is responsible for cardholder verification velocity checking.

The API is the interface that Applications use to request services from the CVM.

6.2 Card Manager Services

Appendix A.2 - *Open Platform on a Java Card* provides the Java Card™ implementation of the following interfaces.

Appendix A.3 - *Application API Interface on Windows Powered Smart Card* provides the Windows Powered Smart Card implementation of the following interfaces.

6.2.1 Application Access to OPEN Services

Applications may access and/or modify some content known or managed by the OPEN. The following services shall be provided by the OPEN:

- Retrieving the Application's own Life Cycle State stored by the OPEN in the Open Platform 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 (depending on the Application relevant privilege),
- Setting the content of the historical characters in the Answer-to-Reset (ATR) (depending on the Application relevant privilege),
- Transitioning the Application's own Application Life Cycle State stored by the OPEN in the Open Platform Registry,
- Transitioning the card Life Cycle State to TERMINATED (depending on the Application relevant privilege),

6.2.2 Application Access to CVM Services

This version of the Specification only supports one Cardholder Verification Method (CVM) Service that is the Global PIN.

The following services surrounding the CVM handler may be accessed by an Application. The following services shall be provided by the CVM:

- 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 Application having the relevant privilege,
- Requesting that the OPEN verify 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 Application having the relevant privilege.

6.2.3 Application Access to Issuer Security Domain Services

Applications associated with the Issuer Security Domain have the ability to access Issuer Security Domain services. By using these services, the Application may rely on cryptographic support from the Issuer Security Domain to ensure confidentiality and integrity during personalization and runtime. The implementation of these services is beyond the scope of this Specification.

The Issuer Security Domain services defined in this Specification are generic and shall encompass the following services.

- Initiating a Secure Channel,

- Verifying the off-card Card Issuer,
- Creating a Secure Channel upon successful verification of the off-card Card Issuer,
- Providing a method of mutual authentication between the card and a off-card Card Issuer,
- Unwrapping a command received within a Secure Channel,
- Verifying the integrity when unwrapping,
- Obtaining the original data in the case of confidentiality when unwrapping,
- Controlling the sequence of APDU commands,
- Decrypting and possibly verifying a secret data block,
- Closing a Secure Channel upon request, and
- Destroying any secret created by the act of setting up a Secure Channel.

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

- Wrapping a response sent within a Secure Channel (adding integrity and/or encrypting the original data in the case of confidentiality), and
- Controlling the sequence of APDU responses.

6.2.4 Issuer Security Domain Access to Applications

The Issuer Security Domain has the facility to receive a STORE DATA command destined to one of its associated Applications. The Issuer Security Domain pre-processes this command according to the current Secure Channel and prior to the command being forwarded to the Application.

6.3 Command Dispatch

The commands received by an Open Platform card shall either be processed by the OPEN or dispatched to the selected Application for processing.

When a SELECT [by name] command is received, the OPEN shall determine if the Application is present in the Open Platform Registry and is available for selection. If the Application is available for selection, the OPEN shall make this Application the selected Application. The response data for the SELECT command is determined by the application.

When a SELECT command is received and cannot result in an Application becoming the selected Application, the OPEN shall dispatch it to the currently selected Application. It is the responsibility of the Application to correctly reject SELECT commands that it does not recognize, expect or cannot process.

Any other type of command received shall be dispatched by the OPEN to the currently selected Application. It is the responsibility of the Application to correctly reject commands that it does not recognize, expect or cannot process.

6.3.1 Application Selection

The OPEN shall support Application selection via two processes:

- Implicit Selection,
- Explicit Selection.

The OPEN may also support additional selection processes. Partial AID Selection as defined in Section 6.3.1.2 - *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.

6.3.1.1 Implicit Selection

After the Answer-to-Reset (ATR) and before the first command is issued to the card, the OPEN shall determine which Application in the Open Platform Registry exists with the Default Selected privilege. This Application shall become the selected Application.

Runtime Behavior

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

- If the Card is in the Life Cycle States CARD_LOCKED or TERMINATED, the Issuer Security Domain is the selected Application and the OPEN shall not attempt to identify the Default Selected Application.
- In all other cases the OPEN shall search the Open Platform Registry for the Application that is marked with the "default selected" privilege. If this is an Application in the Life Cycle State LOCKED, the Issuer Security Domain shall become the selected Application.

6.3.1.2 Explicit Selection

At any time during a Card Session the OPEN may receive a request to select an Application (SELECT [first or only occurrence] command). The OPEN shall determine if the requested AID matches or partially matches an entry within the Open Platform Registry and whether this entry is selectable (i.e. an application in the correct Life Cycle State). If so this Application becomes the selected Application.

At any time during a Card Session that has already contained a SELECT [first or only occurrence] command, the OPEN may receive a request to select a next Application (SELECT [next occurrence] command). The OPEN shall determine if the requested AID matches or partially matches another entry within the Open Platform Registry and whether this entry is selectable (i.e. an application in the correct Life Cycle State). If so this next Application becomes the selected Application.

Runtime Behavior

The following requirements apply to the OPEN in the explicit Application selection (SELECT [by name]) process (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 is the Issuer Security Domain, the Issuer Security Domain is re-selected and a warning is returned to the off-card entity.
    - If any other Application is being selected, the Issuer Security Domain remains selected and an error is returned to the off-card entity.

- Only Applications that are not in the Life Cycle States INSTALLED or LOCKED are included in the search process.
- If a SELECT [first or only occurrence] or SELECT [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 [first or only occurrence] is received, the search always begins from the start of the Open Platform Registry.
- If a SELECT [next occurrence] is received, the search always begins from the entry following the currently selected Application in the Open Platform Registry.
- If a full or partial match is found, this Application shall become the currently selected Application and the SELECT command, whether [first or only occurrence] or [next occurrence], is dispatched to the Application.
- If no full or partial match is found, the currently selected Application shall remain the selected Application and
    - If the SELECT command has the [first or only occurrence] parameter set, the SELECT command is dispatched to the Application.
    - If the SELECT command has the [next occurrence] parameter set, the OPEN shall return the appropriate error to the off-card entity.

6.3.2 Application Command Dispatch

Once an Application becomes the selected Application, the responsibility for subsequent command dispatching still rests with the OPEN (i.e. all commands except SELECT [by name] commands are immediately dispatched to the Application). The processing of the command by the currently selected Application is beyond the scope of this Specification.

6.4 Card Content Management

Card Content management on an Open Platform card includes the capability for the loading, installation, and removal of Card Content. The OPEN is responsible for the Card Content management even though a Security Domain receives the initial requests (APDU commands).

The following sections describe the implementation and the OPEN requirements to support Card Content loading, installation and removal.

6.4.1 Card Content Loading and Installation

The Open Platform Card Content loading process is designed to allow the Card Issuer to add code to mutable persistent memory in the card. Card Content Loading is prohibited in the card Life Cycle State CARD_LOCKED.

The Issuer Security Domain 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.

The Open Platform 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 prohibited in the card Life Cycle State CARD_LOCKED.

All Card Content installation shall be combined within a load process or performed within an install process.

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 Windows® Powered Smart Card OPL format are examples of an expected Load File Data Block. 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. (See Section 7.7 - *Delegated Management Tokens and Receipts and DAP Verification* for more details).

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 Open Platform Registry for the Executable Load File. The OPEN shall also create an entry in the Open Platform Registry for each Executable Module present within an Executable Load File. However, Executable Modules are not yet ready for execution. Applications shall then be installed, resulting in another entry in the Open Platform Registry.

Figure 6-2 details the two possible phases of the Card Content loading and installation.

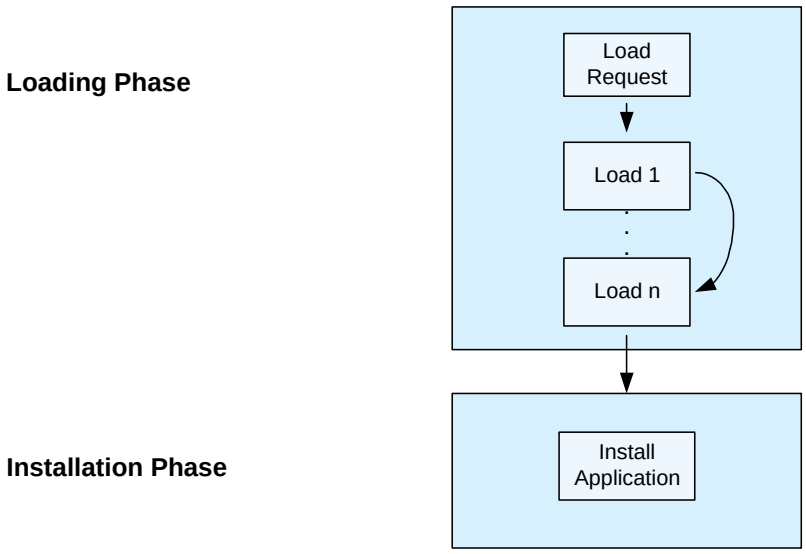


Figure 6-2: Loading and Installation process

The Issuer Security Domain has the implied privilege for loading any Load File to the card regardless of which Security Domain the Executable Load File is being associated with (this equates to extradition of the Load File). The Issuer Security Domain also has the implied privilege for installing any Application on the card regardless of which Security Domain the Executable Load File containing the Executable Module is associated with.

6.4.1.1 Card Content Loading

The phases in Figure 6-2 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] serves as the load request for load. 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.
- The INSTALL and LOAD commands are processed by the Issuer Security Domain before forwarding the load request and Load File to the OPEN for processing.

This Specification allows for two scenarios regarding an executable Application becoming present on a card:

- The Load File is loaded into the card and stored in memory as an Executable Load File. The installation phase occurs immediately following the loading phase.
- The Load File is loaded into the card and stored in memory as an Executable Load File. The installation phase occurs at some time in the future, separate from the loading process.

Runtime Behavior (loading)

The following runtime behavior requirements apply to the OPEN during the content loading process:

- On receipt of a load request (data contained in the INSTALL [for load] command), the OPEN shall:
  - Check that the AID of the Load File is not already present in the Open Platform Registry as an Executable Load File or Application,
  - If an associated Security Domain AID is present, check that this AID exists within the Open Platform Registry, that the associated Security Domain has the relevant privilege, and that the associated Security Domain is in a valid Life Cycle State (i.e. PERSONALIZED) (see Section 5.3.2 - *Security Domain Life Cycle States*). As this equates to the extradition of the Load File to a Security Domain other than that performing the load, request this Security Domain to indicate whether it accepts Content Extraditions.
- On receipt of the Load File (one or more LOAD commands), the OPEN shall:
  - Verify the resource requirements of the Load File,
  - Check in the Open Platform Registry if any Security Domain has the privilege for mandating authentication (Mandated DAP Verification privilege) of the Load File Data Block and if so:
    - ✓ Ensure that the required authentication data (DAP Block identifying the above Security Domain) is present in the Load File.
  - Check in the Open Platform Registry if the associated Security Domain has the privilege for authenticating (DAP Verification privilege) the Load File Data Block 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 from the Load File,
    - ✓ Check that each extracted authentication data relates to a Security Domain present in the Open Platform Registry,

- ✓ Request that Security Domain(s) associated with authentication data present in the Load File verify that the authentication data relates to the Load File Data Block Hash.

- On final receipt of the Load File (last LOAD command) the OPEN shall:
  - Verify the Load File Data Block Hash received in the load request if authentication data was present in the Load File,
  - Create an Executable Load File using the Load File Data Block,
  - Create an entry for the Executable Load File and its associated Security Domain in the Open Platform Registry,
  - Create an entry for each Executable Module within the Executable Load File in the Open Platform Registry. The associated Security Domain for the Executable Module shall be the same as the associated Security Domain for the Executable Load File.

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.

6.4.1.2 Card Content Installation

An INSTALL [for install] command is then used to request the installation of an application. The Issuer 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 Open Platform Registry regarding the Application Life Cycle State, Security Domain association, and Application privileges.

Runtime Behavior (Installation)

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

- On receipt of the install request (data contained within the INSTALL [for install] command), the OPEN shall:
  - Check that the Executable Module AID is present in the Open Platform Registry,
  - Check that the Instance AID (for future selection of the Application) is not already present in the Open Platform Registry as an Application or Executable Load File,
  - Verify the resource requirements of the Application,
  - 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 Application specific Install Parameters,
  - Create an entry in the Open Platform Registry for the Application and its associated Security Domain, Life Cycle State and privileges.

If the OPEN determines that card resources are insufficient for installing the Application, the OPEN shall terminate the installation process, shall return the appropriate error and shall reclaim any memory allocated to the install process.

Loading and Installation Flow

Figure 6-3 is an example of the loading and installing an Application to an Open Platform card. In this example 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.

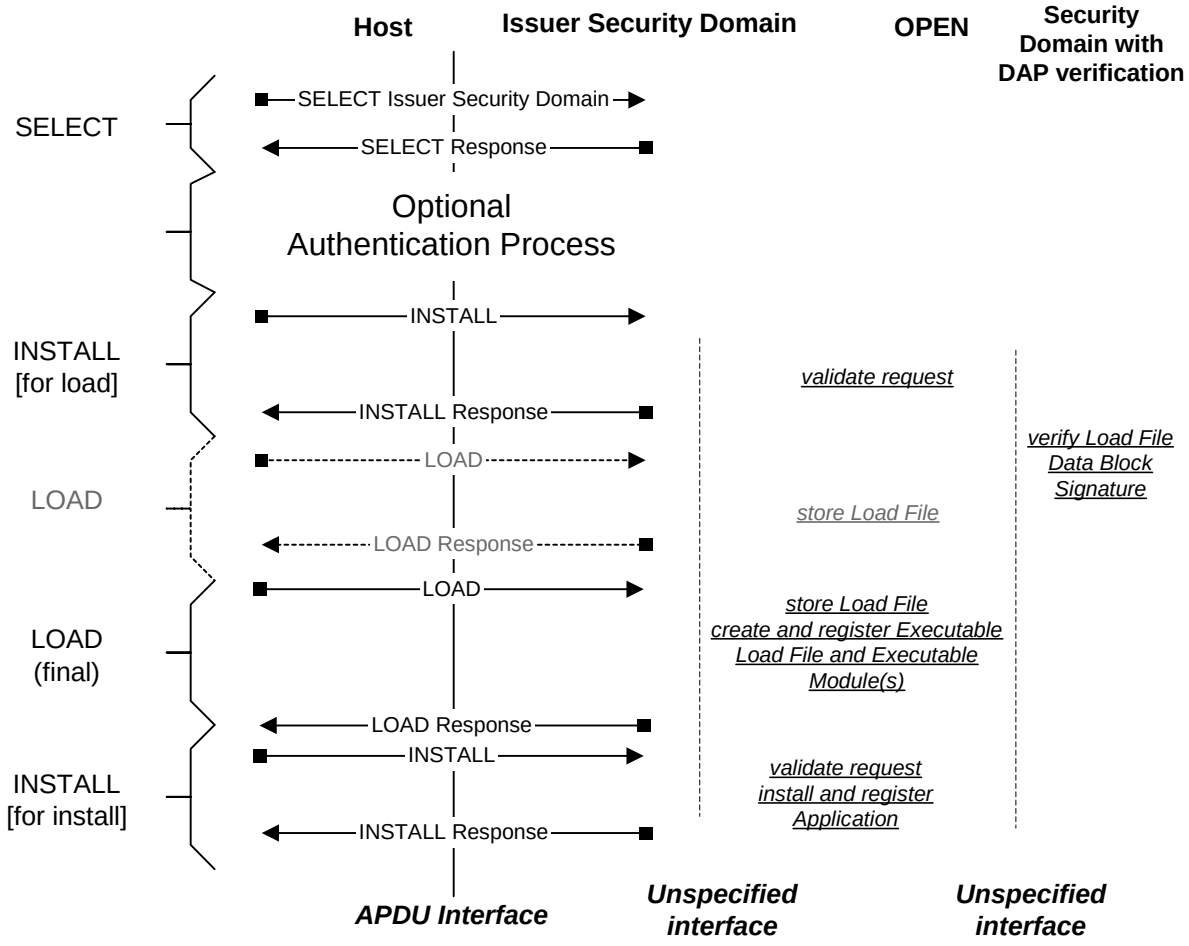


Figure 6-3: Load and Installation Flow Diagram

Installation Flow

Figure 6-4 is an example of installing an Application from an Executable Load File already present on the card:

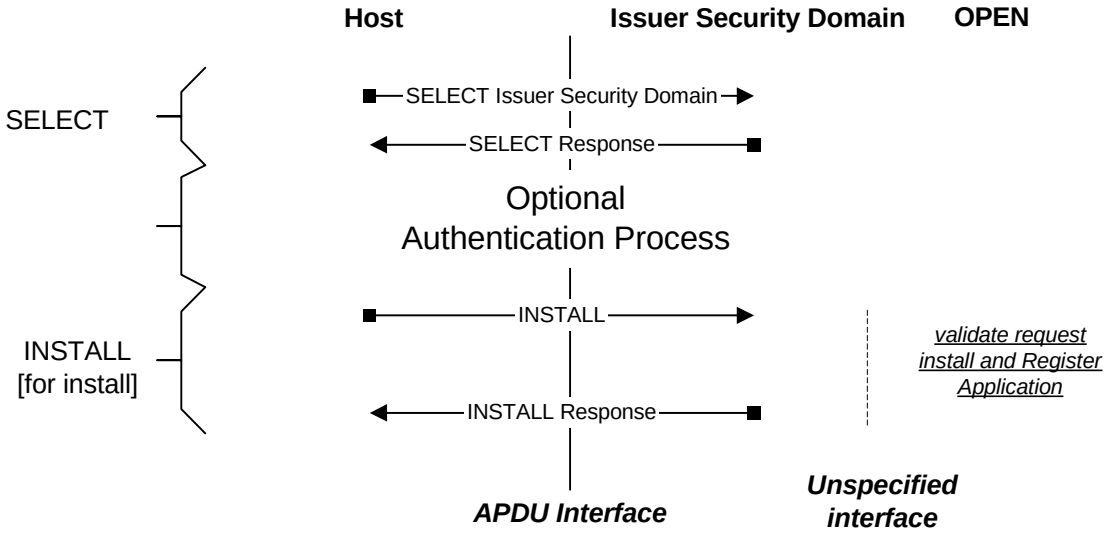


Figure 6-4: Install Flow Diagram

6.4.2 Content Removal

This section defines the content removal process that enables the Card Issuer to flexibly manage 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.

The DELETE command (see Section 9.2 - *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 Issuer Security Domain before forwarding the removal request to the OPEN for processing.

Depending on the memory location of the removed Applications 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.

The Issuer Security Domain has the implied privilege for deleting any Application or Executable Load File from the card regardless of which Security Domain the Application or Executable Load File is associated with.

6.4.2.1 Application Removal

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

Runtime Behavior

The following runtime behavior requirements apply to the OPEN during the Application Removal process. The OPEN shall:

- Determine if the Application being deleted has an entry within the Open Platform Registry,

- Determine if any other Applications present in the card make reference to this Application,
- Determine if any other Applications present in the card maintain references to any data within this Application,
- Release and mark as available any mutable persistent memory,
- Remove the entry for the Application within the Open Platform Registry,
- If the application has the Default Selected privilege, re-assign Default Selected privilege to the Issuer Security Domain.

If the OPEN determines that the content cannot be deleted or that resources within the content being deleted are still referenced, the OPEN shall terminate the delete process and shall inform the Issuer Security Domain to return the appropriate response.

6.4.2.2 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.

Runtime Behavior

The following runtime behavior requirements apply to the OPEN during the Executable Load File Removal process.

The OPEN shall:

- Determine if the Executable Load File being deleted has an entry within the Open Platform Registry,
- Determine if any Applications or other Executable Load Files present in the card maintain references to this Executable Load File,
- Release and mark as available any mutable persistent memory,
- Remove the entry for the Executable Load File and the entries for any Executable Modules present in the Executable Load File from within the Open Platform Registry.

If the OPEN determines that the content cannot be deleted or that resources within the content being deleted are still referenced, the OPEN shall terminate the removal process and shall inform the Issuer Security Domain to return the appropriate response.

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 Open Platform Registry.

Deletion Flow

Figure 6-5 is an example of deleting an Executable Load File:

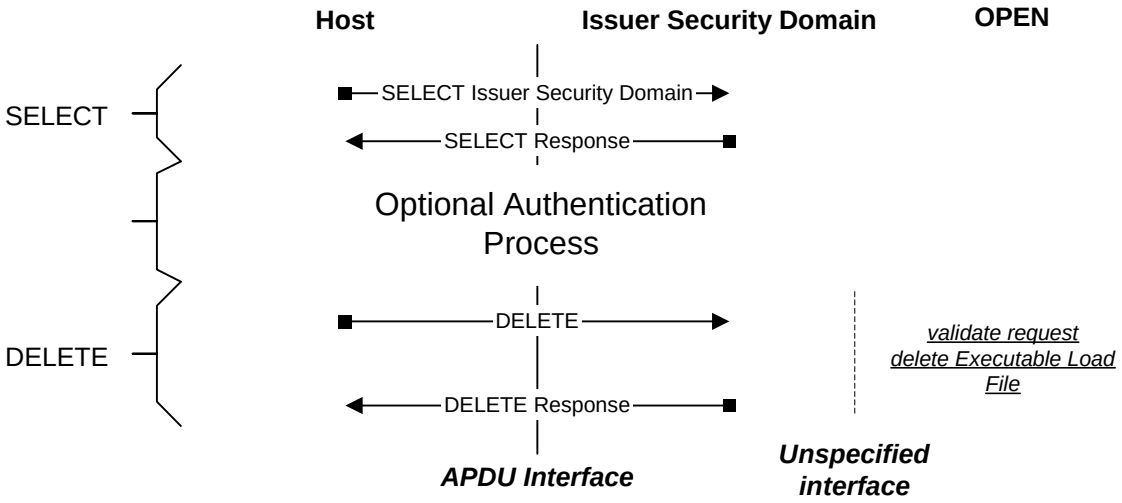


Figure 6-5: Executable Load File Deletion Flow

6.4.3 Content Extradition

The Open Platform Card Content extradition process is designed to allow the Card Issuer to associate to a different Security Domain a previously installed Application. Card Content extradition is prohibited in the card Life Cycle State CARD_LOCKED. The Issuer Security Domain shall verify the extradition request before the OPEN will allow the extradition.

Runtime Behavior

The following runtime behavior requirements apply to the OPEN during the Card Content Extradition process. The OPEN shall:

- Determine if the Application being extradited exists within the Open Platform Registry,
- Check that the Security Domain requesting the extradition is the Security Domain associated with the Application being extradited,
- Check that the Security Domain AID, to which the Application is being extradited, exists within the Open Platform Registry,
- Check that this Security Domain has the Security Domain privilege,
- Check that this Security Domain is in a valid state (i.e. PERSONALIZED),
- Request this Security Domain to indicate whether it accepts Card Content Extraditions.

Extradition Flow

Figure 6-5 is an example of extraditing an Application:

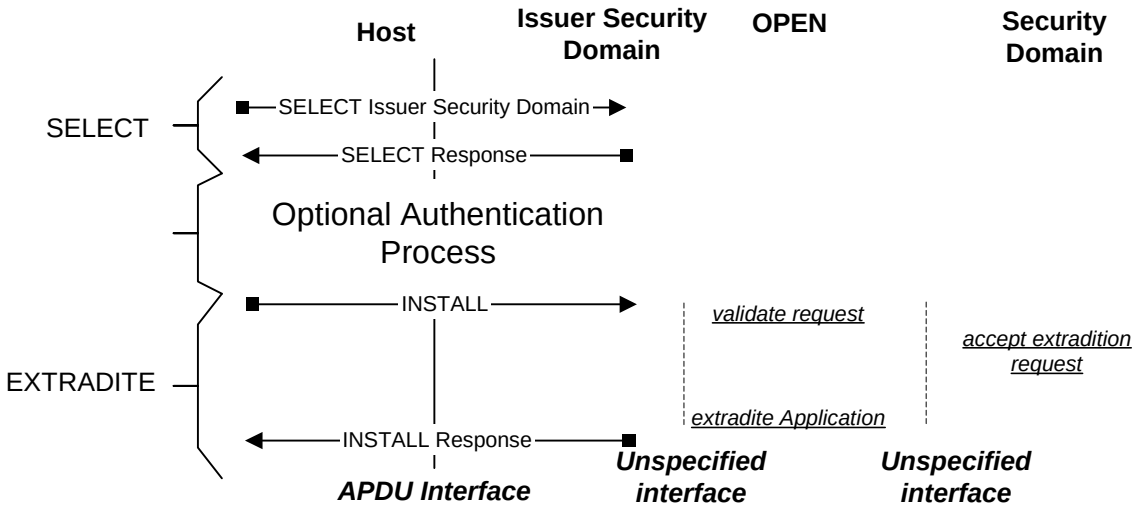


Figure 6-6: Application Extradition Flow

6.5 Delegated Management

The Open Platform is designed for providing maximum flexibility to the Card Issuer and its business partners regarding Card Content management while ensuring that the Card Issuer keeps control over the Card Content present on the card. Delegated Management provides this capability so that a Card Issuer can provide an Application Provider the capability to perform specific Card Content management.

Delegated Management is not a mandated feature of Open Platform and is only necessary for Card Issuers that choose to offer this flexibility and in order to achieve this, close co-operation between the OPEN and the Issuer Security Domain is required. Cryptographic security is required for Delegated Management and the Issuer Security Domain requires the knowledge of keys and algorithms used for Tokens and optionally for Receipts. If the Card Issuer's Security policy requires Receipt generation, the Issuer Security Domain shall also keep track of a Confirmation Counter that is incremented when generating each Receipt.

Delegated Management is a privilege that a Security Domain shall be granted during Installation. Content Management requires an Application Provider's Security Domain with this privilege to perform:

- Delegated Loading,
- Delegated Installation,
- Delegated Extradition,
- Delegated Deletion.

The Application Provider, instead of the Card Issuer, manages each of the above processes. However, it is important to note that the OPEN performs the physical loading and installation (See Section 7.6 - *Delegated Management*).

6.6 Open Platform Registry

The OPEN owns and manages information deemed necessary to perform the functionality defined by Open Platform. Exactly how this information is managed is beyond the scope of this Specification. For the purpose of this Specification it is assumed to be in the Open Platform Registry.

The Open Platform 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 (e.g., non-volatile memory allocation),
- Store Application Life Cycle information,
- Store card Life Cycle information,
- Track any counters associated with logs.

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

- A Card Issuer invoked action,
- An internal OPEN invoked action,
- An authorized Application invoked action.

The Open Platform Registry contains data elements for all Applications, including Security Domains and the Issuer Security Domain.

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 Open Platform services available to Applications.

6.6.1 Issuer Security Domain Data Elements Description

The Issuer Security Domain AID and card Life Cycle State are stored in the Open Platform Registry similarly to Application information.

The card Life Cycle State may be retrieved by an Application through the OPEN services or by an off-card entity through the Issuer Security Domain (See Section 9.4 - *GET STATUS Command*).

The following sections describe the possible Open Platform Registry data elements for the OPEN.

6.6.1.1 Issuer Security Domain AID

The Issuer Security Domain AID data element uniquely identifies the Issuer Security Domain.

One option of making the Issuer Security Domain the selected Application, is to specify this 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 Card Issuer is responsible for setting the value for the Issuer Security Domain AID.

6.6.1.2 Card Life Cycle State

The Issuer Security Domain inherits the Life Cycle State of the card.

6.6.2 Application/Executable Load File/Executable Module Data Elements

The following data elements are defined:

- Application/Executable Load File/Executable Module AID data element
- Application Life Cycle State data element
- Resource Allocation data element
- Application privileges data element
- Associated Security Domain AID data element

6.6.2.1 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 Open Platform 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.

6.6.2.2 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.

6.6.2.3 Resource Allocation

The Resource Allocation data element contains information about the resources that are available to an Application. It is a system-specific value and is used as a control mechanism by the OPEN to limit the amount of resources that an Application may claim during runtime.

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

Runtime Behavior

The OPEN shall terminate processing of the Application and shall return an appropriate response code if the additional resource requested by an Application exceeds its allocation limit.

The OPEN may choose to lock an Application that makes repeated attempts to allocate additional resources beyond its allocation limit.

6.6.2.4 Application Privileges

The Application Privileges data element indicates the privileges for each Application.

The following Application privileges are defined:

- Application is a Security Domain,
- Application is a Security Domain with DAP Verification privileges,
- Application is a Security Domain with Delegated Management privilege,
- Application is a Security Domain that mandates the presence of a DAP Block in all Load Files,
- Application has the privilege to lock the card,
- Application has the privilege to terminate the card,
- Application is the "default-selected" Application,
- Application has the privilege to manage the card CVM.

The following rules apply to the assignment of Application privileges:

- Only one Application or Security Domain in the card may be set with the Default Selected Application Privilege at a time (e.g. the Issuer Security Domain or a current legacy application),
- Once default selected privilege is assigned to an application, the privilege can only be reassigned to a new application by deleting the application which has the privilege.
- The Default Selected application privilege may be assigned only if the Issuer Security Domain has the Default Selected Application privilege,

Otherwise, the Application privileges are not mutually exclusive; therefore, one or more privileges may be marked as set for an Application.

The Issuer Security Domain, as the on-card representative of the Card Issuer, is the most privileged entity of the card as it is the only entity that performs Card Content management without having been explicitly delegated previously.

The Issuer Security Domain shall have the following set of privileges clearly identifying its functionality (i.e. a Security Domain with card lock, card terminate and CVM management privileges and possibly the Default Selected privilege) in addition to its implied unrestricted Card Content management privilege.

Runtime Behavior

The OPEN shall identify the Issuer Security Domain and use the Application Privileges data element for controlling the following runtime behavioral requirements:

- Identifying the Default Selected Application during the ATR sequence,
- Requesting Token verification and Receipt generation,
- Checking that a DAP Block is mandated in Load Files,
- Determining if an Application is a Security Domain,
- Determining if a Security Domain has DAP Verification privileges,
- Determining if a Security Domain has the Delegated Management privilege,
- Checking for the validity of a request to lock the card,
- Checking for the validity of a request to terminate the card,

- Ensuring that only one Application or Security Domain (including the Issuer Security Domain) is marked as the Default Selected Application. This privilege can be assigned to an Application only if the Issuer Security Domain currently has this privilege,
- Ensuring that only the Default Selected Application can successfully request that the Answer-to-Reset historical characters be modified,
- Checking for the validity of requests to change the CVM value and the CVM management parameters (e.g. CVM Retry Limit).

6.6.2.5 Associated Security Domain AID

The Associated Security Domain AID data element contains the AID of an Executable Load File's Application's associated Security Domain. The INSTALL [for load] command may specify the associated Security Domain that shall be linked to the Executable Load File and as such to each Executable Module within the Executable Load File. If no Security Domain is specified in the INSTALL [for load] command, the Security Domain performing the load is assumed to be the associated Security Domain.

An Application is installed through the Issuer Security Domain or through the Security Domain associated to the Executable Module. In both cases the Security Domain associated with the Executable Module is also associated to the Application.

Applications may use certain services of their associated Security Domains

Runtime Behavior

When the OPEN receives an Application request to use a service of the associated Security Domain the OPEN shall:

- Locate the Application's entry in the Open Platform Registry,
- Retrieve the Associated Security Domain.

If an Associated Security Domain is present, the entry for the associated Security Domain shall be located and the Application's request for service forwarded to this Security Domain.

The associated Security Domain shall be in an accessible Life Cycle State (i.e. PERSONALIZED) in order for its services to be usable.

6.7 Security Management

As described in Section 6.1 - *Card Manager Overview*, the OPEN is at the center of the security scheme and is responsible for controlling access to and executing all of the most trusted services on the card. In its role as administrator of the complete card, the OPEN shall also implement security monitoring and control functions in order to ensure card security and integrity during runtime execution. These functions are active at all times in the card regardless of which Application is currently selected. These functions shall include the following:

- Application locking,
- Card locking,
- Card termination

The OPEN may also provide the following function:

- Optional Tracing and Event Logging.

The OPEN is responsible for the security and the controls regarding the loading, the installation, the extradition, and the deletion of Card Content.

The OPEN shall:

- Verify the Load File Data Block Hash,
- Request the verification of the Delegated Management Token,
- Request the generation of the Delegated Management Receipt,
- Request the verification of the Load File Data Block Signature.

6.7.1 Application Locking

The Card Issuer has a mechanism to disable 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. The Card Issuer is the only entity that may initiate the locking of an Application.

Runtime Behavior

On receipt of a request to lock an Application (using a SET STATUS command received by the Issuer Security Domain), the OPEN shall:

- Check that an entry for the Application being locked is present in the Open Platform Registry,
- Set the Application Life Cycle State to LOCKED,
- Keep a record of the previous Application Life Cycle State to ensure that the Card Issuer can only transition the Application back to this previous state.

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

If the locked Application has the Default Selected Application privilege, the Issuer Security Domain shall act as the Default Selected Application until the Application is unlocked.

If at a later stage the Application is unlocked, the Default Selected privilege shall get reinstated.

Application Locking Flow

Figure 6-7 is an example of locking an Application:

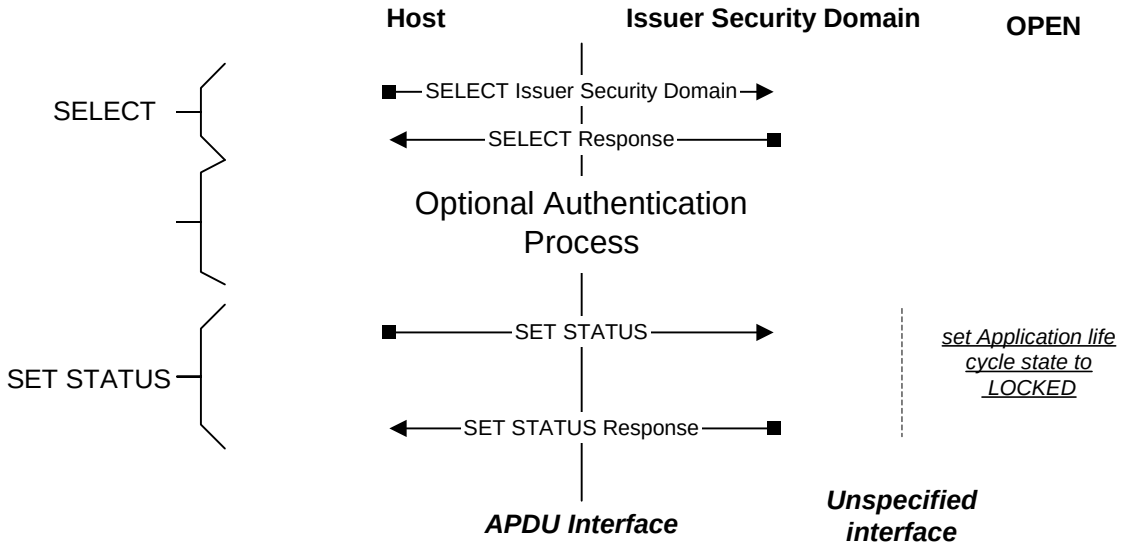


Figure 6-7: Application Locking Flow

6.7.2 Card Locking

Once the card Life Cycle State is CARD_LOCKED, all Applications except the Issuer Security Domain 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 an instruction from the Issuer Security Domain or a privileged Application instructing the OPEN to lock the card, the OPEN shall:

- Check within the Open Platform Registry that the Application requesting the locking of the card has the required card locking privilege,
- Check that the current card Life Cycle State is SECURED,
- Set the card Life Cycle State to CARD_LOCKED,

The card Life Cycle State CARD_LOCKED only applies functionally at the completion of the current Application Session, i.e. when an attempt is made to select an Application or the card is reset. Following this Application Session, the Issuer Security Domain is the only selectable Application and Card Content changes shall be disallowed.

Card Locking Flow

Figure 6-8 is an example of locking the card:

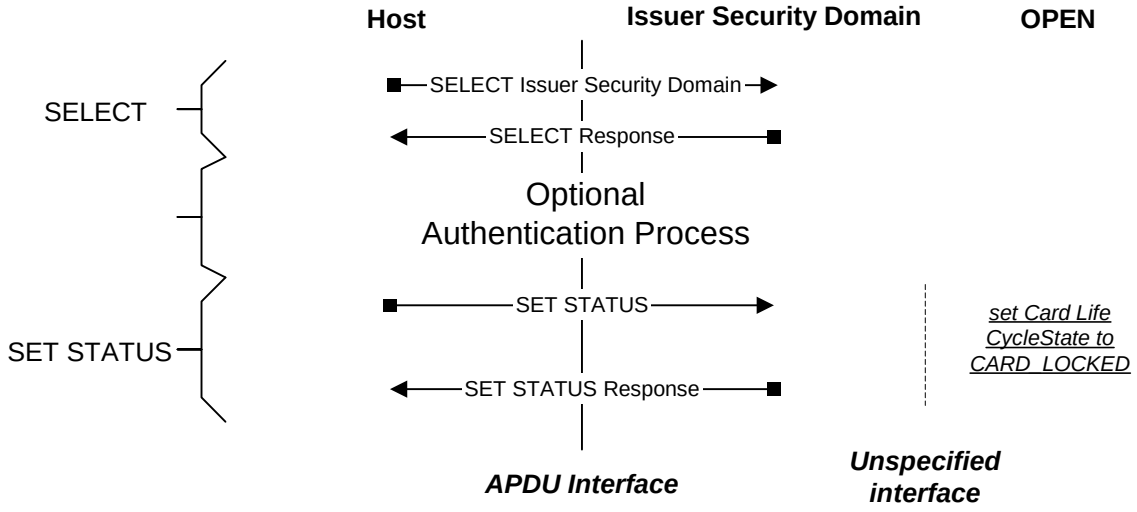


Figure 6-8: Card Locking Flow

6.7.3 Card Termination

In the card Life Cycle State TERMINATED, all communication to the card besides 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 by the Card Issuer:

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 termination privilege.

Runtime Behavior

On receipt of a SET STATUS command or an instruction from a privileged Application instructing the OPEN to terminate the card, the OPEN shall:

- Check within the Open Platform Registry that the Application requesting the termination of the card has the required card termination privilege,
- Set the card Life Cycle State to TERMINATED from its current state.

The card Life Cycle State TERMINATED only applies functionally at the completion of the current Application Session, i.e. when an attempt is made to select an Application or the card is reset. Following this Application Session, no Applications are selectable (i.e. the Issuer Security Domain is always implicitly selected) and only the GET DATA command is processed.

Terminate Card Flow

Figure 6-9 is an example of terminating the card:

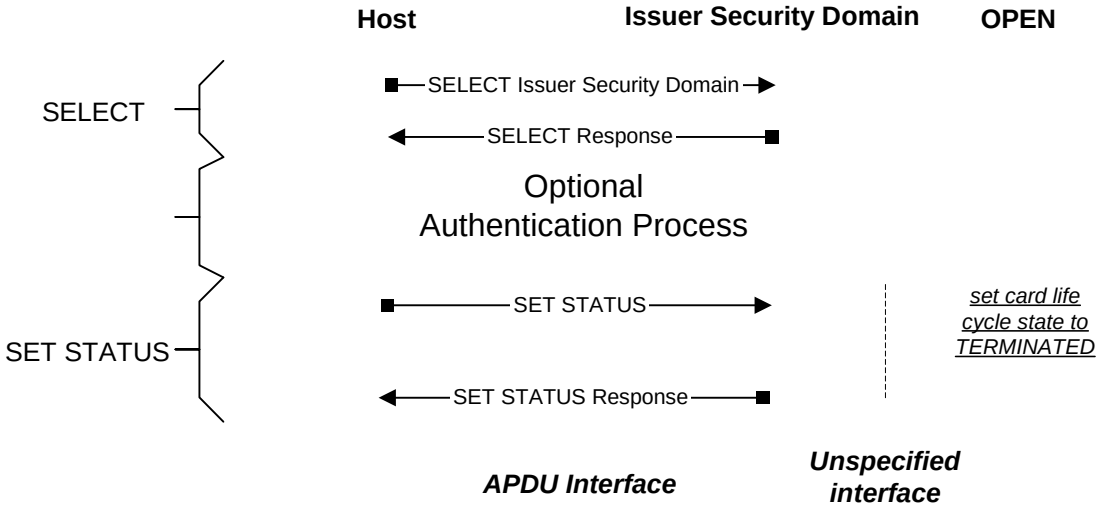


Figure 6-9: Terminate Card Flow

6.7.4 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.

6.7.4.1 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.

6.7.4.2 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 the Issuer 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 Open Platform Registry. The OPEN may choose to lock an Application that is exhibiting such behavior.

6.7.5 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, and are therefore considered extensions to the Issuer Security Domain.

6.7.6 Securing Content Loading and Installation

The following section details the security features that shall be available during Card Content loading and installation. The responsibility of ensuring that these controls are present and correct rests with the OPEN.

6.7.6.1 Load File Data Block Hash

The Load File Data Block Hash is a redundancy 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. Typically it should be a hash of the complete Load File Data Block. Mandating of the Load File Data Block Hash is only required when Delegated Management loading occurs or when a DAP Block is present in a Load File. The Load File Data Block Hash is not required when the loading occurs through the Issuer Security Domain and no DAP Blocks are required to be present in the Load File.

The OPEN, when receiving the complete Load File Data Block, should perform the same function as the off-card entity and compare its own result with the Load File Data Block Hash previously received during the processing of the INSTALL [for load] command.

See Appendix B.2.1 - *Secure Hash Algorithm (SHA-1)* for more details.

If the Load File Data Block Hash received by the card does not match the Load File Data Block Hash generated on card, any reference to the Executable Load File shall be discarded.

6.7.6.2 Tokens

Tokens relate specifically to Delegated Management and more detail is provided in Section 7.6 - *Delegated Management*. The Card Issuer provides a Token to the Application Provider performing a Delegated Management function.

During Delegated Management, Card Content management requests are transferred to the OPEN from a Security Domain that has the Delegated Management privilege.

The OPEN, on receiving a request from a Security Domain with Delegated Management privileges, shall request the Issuer Security Domain to verify the validity of the Token and shall act appropriately if the Issuer Security Domain cannot verify the Token.

6.7.6.3 Load File Data Block Signature

A Load File may contain one or more DAP (Data Authentication Pattern) Blocks that allow an entity other than the loading entity to verify the authenticity and the integrity of the Load File Data Block. A DAP Block is required when a Security Domain with the mandated DAP Verification privilege is present on the card. A DAP Block is also required when the on-card associated Security Domain has the DAP Verification privilege. Other DAP Blocks may be present in the Load File and shall be verified by the identified Security Domain(s). The signature within a DAP Block allows the Application Provider or Controlling Authority to ensure that only verified content is being loaded to an Open Platform card.

The Security Domain linked to each Load File Data Block Signature shall perform the verification. It is the responsibility of the OPEN to request that the Security Domain, linked to the Load File Data Block Signature, verify the signature and to act appropriately if the verification fails.

The verification of each Load File Data Block Signature is required in order to commit an Executable Load File to memory.

6.8 Issuer Security Domain

The Issuer Security Domain is regarded as the on-card representative of the Card Issuer. As such, the Issuer Security Domain contains the keys that the Card Issuer will use in support of cryptographic operations for card management functions, Card Issuer Applications management functions, and the Application Provider Applications if desired. As an example, these keys could be used for authentication and for cryptographic operations during the Card Content loading, installation and removal processes, as well as personalization.

The interface and distinction between the Issuer Security Domain and the OPEN is also not defined by the Open Platform Specifications (specifically relating to Card Content management, Token verification, and Receipt generation). It is assumed that the Issuer Security Domain cannot be developed in the same manner as a typical Security Domain or normal Application. Open Platform, therefore, does not mandate that the Issuer Security Domain be loaded in the same manner as Applications. The Issuer Security Domain, while viewed by the Open Platform Registry as an Application, has implementation specific behavior relating to how it becomes an active entity on the card.

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

- Issuer Identification Number (IIN),
- Card Image Number (CIN),
- Card Recognition Data,
- Issuer Security Domain Key Information,
- Other Card Issuer's proprietary data.

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

6.8.1 Issuer Identification Number

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

6.8.2 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 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.

6.8.3 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, what kinds of command it accepts, and what secure protocol it supports.

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

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 7816-6. Card Data can be retrieved using the GET DATA command. (Card Data may also be accessed through the ATR, if a suitable "command to perform" is included in the ATR historical characters.)

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.

6.8.4 On-Card Key Information

Keys are stored in mutable persistent memory and 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.

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

- The identity,
- The intended usage, and
- 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.

The Issuer 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, and a new Key Version Number.
- Replacing a key involves erasing the key slot of the key being replaced, updating the key slot with a new key value, and updating the associated Key Version Number. The Key Identifier remains the same.

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 Card Issuer's Security Policy.

6.9 CVM Management

The Card Manager may provide a mechanism for a Cardholder Verification Method (CVM) that may be used by all Applications on the card. This Specification provides a way for the implementation of a card global Personal Identification Number (PIN) service to support Cardholder Verification requirements. CVM management services shall only be accessible to privileged Applications. The CVM verification services may be accessed by any Application.

If the CVM service is supported, the CVM handler shall:

- Reserve memory space for the CVM management,
- Provide the CVM services to Applications,
- Change the CVM state,
- Change the Retry Limit and Retry Counter.

6.9.1 CVM States

The CVM state may be used by an Application to assist in managing PIN related functions. 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.

The CVM states are:

- ACTIVE
- INVALID_SUBMISSION
- VALIDATED
- BLOCKED

6.9.1.1 CVM State Active

The CVM state shall first become ACTIVE when a privileged Application initializes the CVM value and sets the Retry Limit. 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 affect the Retry Counter.

6.9.1.2 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,
- A privileged Application either unblocks the CVM, or blocks the CVM or changes the CVM value,
- Subsequent CVM verification(s) fail causing the CVM state to transition to BLOCKED.

6.9.1.3 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,
- A privileged Application either blocks the CVM, or changes the CVM value.

6.9.1.4 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. The CVM verification shall always fail in the BLOCKED state.

7. Security Domains

7.1 Overview

The Issuer Security Domain contains keys that the Card Issuer uses in support of cryptographic operations for the Card Issuer's Applications. It is possible to associate Applications owned by an Application Provider to the Issuer Security Domain. However it is not recommended.

An Application Provider Security Domain (Security Domain) is a special type of Application that is responsible for managing and providing cryptographic services for keys that are completely separate from, and not under the control of, the Card Issuer.

These Security Domains are privileged Applications established on an Open Platform card to represent Application Providers who require a level of key separation from the Card Issuer.

These Security Domains shall contain keys capable of providing support to Applications during personalization . Security Domain keys may also be used in support of runtime cryptography for Applications.

In addition to the Application privileges, a Security Domain may support three additional privileges that are referred to as DAP Verification Privilege, Mandated DAP Verification Privilege and Delegated Management Privilege. (These privileges are not available to the Issuer Security Domain.) A Security Domain may support one or more of these additional privileges.

Mandated DAP Verification allows a Controlling Authority to own a Security Domain that always requires to authorize a load process. This ensures that only Load File Data Blocks authorized by the Controlling Authority may be loaded on cards that contain this Security Domain.

DAP Verification allows an Application Provider to own a Security Domain that requires authorizing a load process. This ensures that when associating a Load File to this Security Domain, the Application Provider must have authorized the Load File Data Block. This authorization may also serve as a means for a Security Domain to control the access to some of its services.

Delegated Management allows Application Providers to perform Card Content changes (load, install and extradite) with pre-authorization from the Card Issuer. Applications Providers can also delete Executable Load Files and Applications associated to their on-card Security Domains without pre-authorization from the Card Issuer.

The keys and associated cryptography for all Security Domains including the Issuing Security Domain 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.

Security Domains with DAP Verification privilege are used for:

- Verifying a Load File Data Block signature: a specific key within the Security Domain shall be identified as a DAP Verification key;

Security Domains with Delegated Management privilege can be used for:

- Delegated Loading,
- Delegated Installation,
- Delegated Extradition,
- Delegated Deletion.

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. Security Domains should apply to their own keys the key management principles described for the Issuer Security Domain (see Section 6.8.4 - *On-Card Key Information*).

An Issuer Security Domain shall be present on all Open Platform cards to manage the Card Issuer keys and to optionally provide personalization and cryptographic services to Applications.

A Security Domain is required if Application Provider's Applications are to be personalized using the Application Provider keys.

A Security Domain may either accept or reject any Card Content Extradition request from the OPEN.

During Application installation, indication that the Application has the privileges of a Security Domain is included in the INSTALL command. (See Section 9.5 - *INSTALL Command* for a description of the data elements.)

As the Open Platform does not define the interface between a Security Domain and the OPEN (specifically relating to Delegated Management and DAP Verification), the assumption is made that a Security Domain cannot be developed in the same manner as a normal Application. Open Platform therefore does not mandate that Security Domains be loaded in the same manner as Applications. It is assumed however that installation of a Security Domain is similar to Application installation with the additional setting of the relevant Security Domain privileges.

7.2 Security Domain Services

Appendix A.2 - *Open Platform on a Java Card* provides the Java Card™ implementation of the following interfaces.

Appendix A.3 - *Application API Interface on Windows Powered Smart Card* provides the Windows Powered Smart Card implementation of the following interfaces.

Appendix D - *Secure Channel Protocol '01'* and Appendix E - *Secure Channel Protocol '02'* provide more details on Secure Channel Protocols implementing the following interfaces.

7.2.1 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,
- Verifying an off-card entity,

- Creating a Secure Channel upon successful verification of an off-card entity,
- Providing a method of mutual authentication between the card and a off-card entity,
- Unwrapping a command received within a Secure Channel,
- Verifying the integrity when unwrapping,
- Obtaining the original data in the case of confidentiality when unwrapping,
- Controlling the sequence of APDU commands,
- Decrypting a secret data block,
- Closing a Secure Channel upon request,
- Destroying any secret created by the act of setting up a Secure Channel.

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 (adding integrity and/or encrypting the original data in the case of confidentiality),
- Controlling the sequence of APDU responses.

7.2.2 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 pre-processes this command according to the current Secure Channel and prior to the command being forwarded to the Application.

7.3 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 2 ways i.e. one being to utilize the Runtime Messaging Support described in Section 7.4 - *Runtime Messaging Support* or the other being to utilize the Security Domain's ability to access the Application.

Using this second method 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.

Personalization Flow

Figure 7-1 is an example of an Application receiving data from its associated Security Domain during personalization.

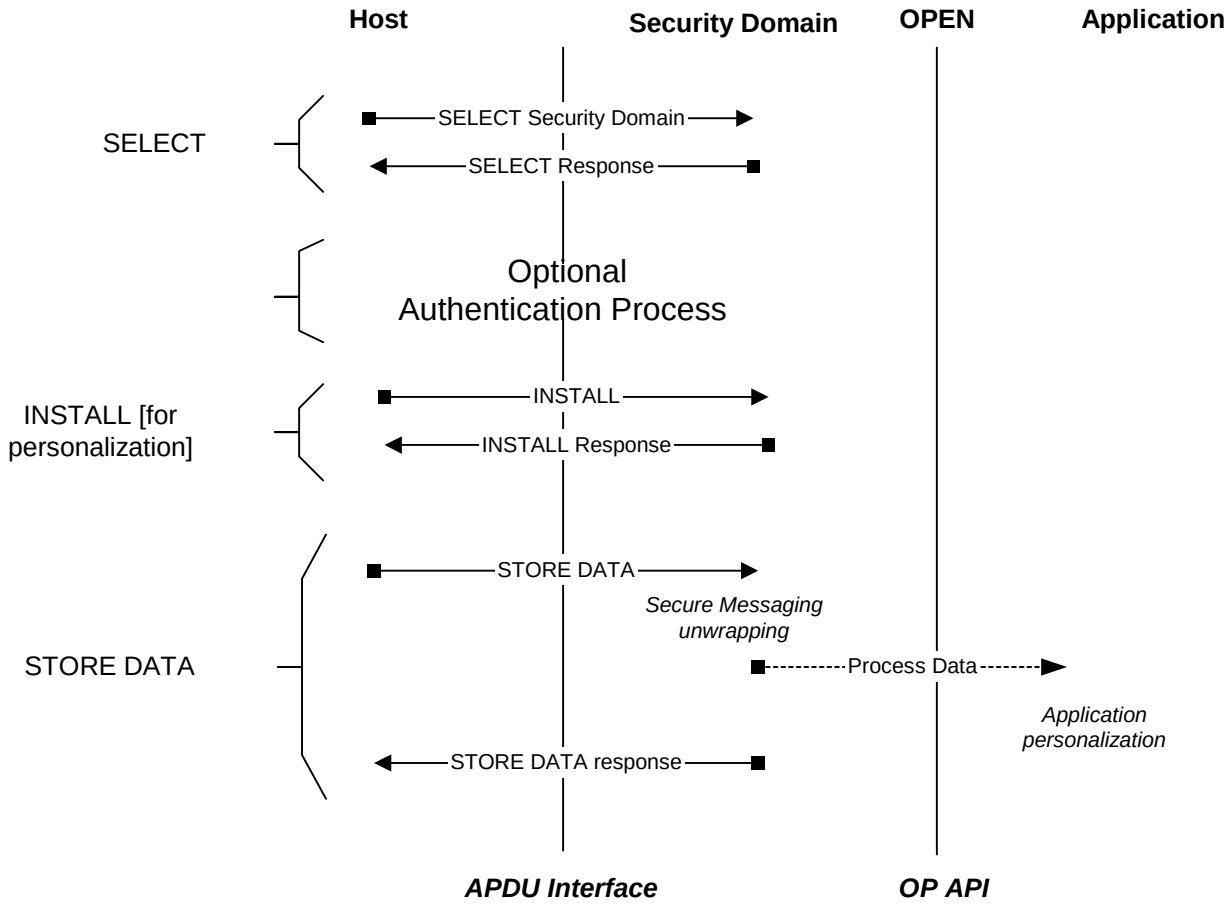


Figure 7-1: Application Personalization through Associated Security Domain

7.4 Runtime Messaging Support

Security Domains may provide runtime support for secure communication to their 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

Figure 7-2 is an example of an Application using the services of its associated Security Domain:

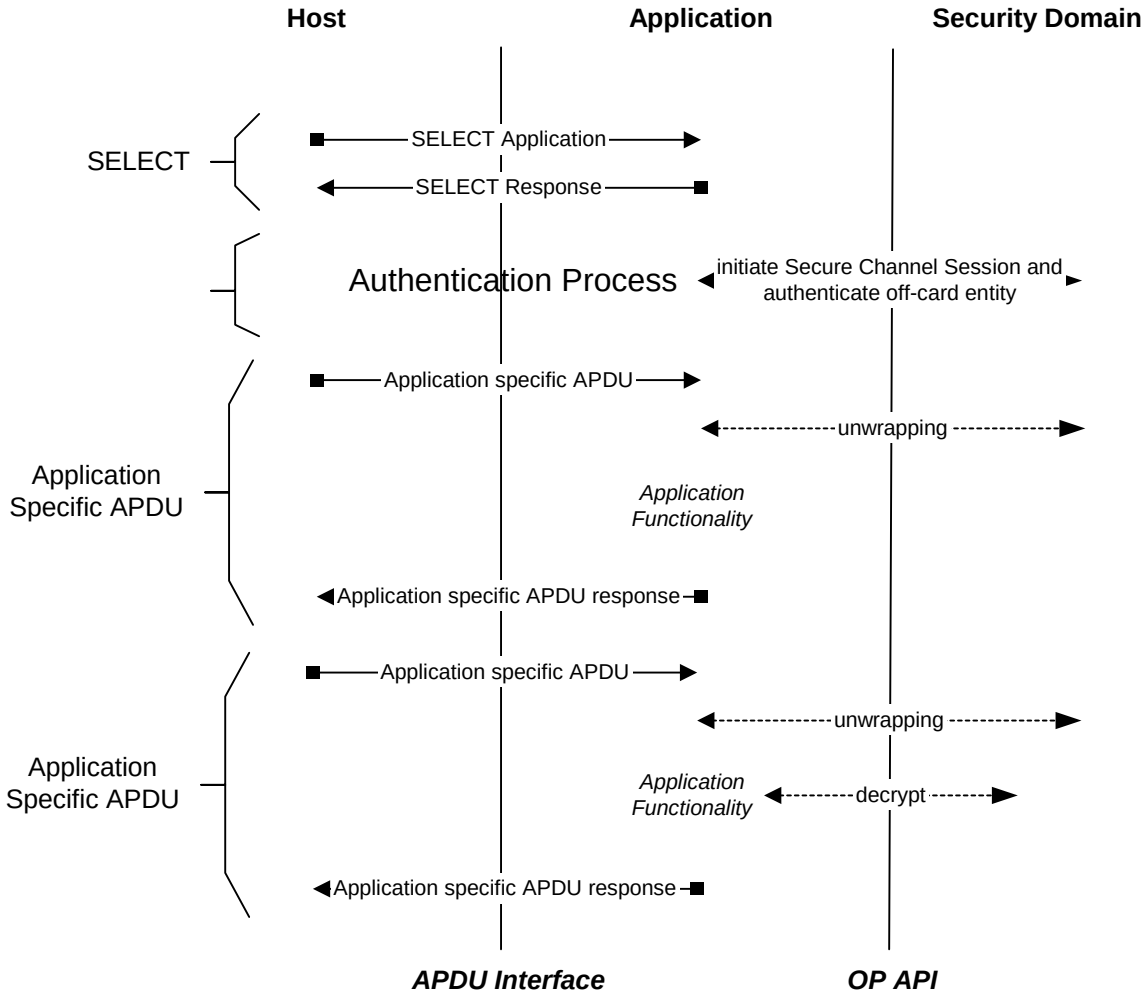


Figure 7-2: Runtime Messaging Flow

7.5 DAP Verification

If the Application Provider does not have a Security Domain with Delegated Management to load Application code to the card, it may rely on the loading services of the Card Issuer or a third party (Controlling Authority or Application Provider) that has the loading privilege on the card. A Controlling Authority may require that all Application code to be loaded to the card shall be verified by the Security Domain of the Controlling Authority.

The Application Provider may require a check or the Controlling Authority may mandate a check of Application code integrity and authenticity before the Application code is loaded, installed and made available to the cardholder. The DAP Verification Privilege for a Security Domain detailed in this Specification is to provide this service on behalf of an Application Provider. The Mandated DAP Verification privilege provides this service on behalf of a Controlling Authority.

Verification of a Load File Data Block Signature is required when one or more DAP Blocks are present in the Load File. Each DAP Block contains the identifier of a Security Domain and a Load File Data Block Signature.

The OPEN shall perform the controls and verifications described in Section 6.4.1 - *Card Content Loading and Installation*.

The OPEN shall send the Load File Data Block Hash, the Load File Data Block Signature and the AID of the Load File to the linked Security Domain for verification.

It is the Security Domain's responsibility to inform the OPEN whether the actual signature is valid or not.

The presence of a DAP block within the Load File implies that the owner of the Security Domain identified in the DAP block verified off-card the content of the Load File Data Block. The Load File Data Block Hash, signed to create the Load File Data Block Signature, creates this link between the Load File Data Block, the Security Domain and its owner.

Security Domains with DAP Verification Privileges shall know the key(s) required to verify a Load File Data Block Signature.

7.6 Delegated Management

The design of the Open Platform 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.

The concept of Delegated Management defined in this Specification gives the Card Issuer the possibility of empowering partnered Application Providers the ability to initiate approved and pre-authorized Card Content changes (loading, installing or extraditing).

This approval, which is central to the concept of Delegated Management, ensures that only Card Content changes that the Card Issuer has authorized will be accepted and processed by the OPEN. This delegation of control in the Card Content changes gives the Application Provider more flexibility in managing its Application.

The following functions shall be supported in Delegated Management:

- Delegated Loading (requires a pre-authorization)
- Delegated Installation (requires a pre-authorization)
- Delegated Extradition (requires a pre-authorization)
- Delegated Deletion (no pre-authorization required)

Delegated Management is not allowed when the card Life Cycle State is CARD_LOCKED.

7.6.1 Delegated Loading

Delegated loading allows an Application Provider to establish a Secure Channel, for transferring Load Files directly to its Security Domain.

The load process comprises one 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 Issuer Security Domain verification, to ensure that the Card Issuer authorized the load process and the loading of the content of the Load File Data Block.

The Load Token implies that the Card Issuer pre-authorized the data required for the delegated load process. The pre-authorized data includes most of the INSTALL [for load] command and the Load File Data Block Hash. 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 Card Issuer 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.

Runtime Behavior

The OPEN shall provide the following additional services (see Section 6.4.1.1 - *Card Content Loading* for basic requirements) during the delegated loading process:

- Check that the Life Cycle of the Security Domain performing the load is in the state of PERSONALIZED,
- Check that the Security Domain performing the load has the Delegated Management privilege,
- Request the Issuer Security Domain to verify the Load Token generated by the Card Issuer,
- On completion of the load procedure the OPEN shall:
  - Request the Issuer Security Domain to generate a Load Receipt.

7.6.2 Delegated Installation

Delegated Installation allows an Application Provider to establish a Secure Channel, for installing an Application from an Executable Load File that is associated to the Application Provider's Security Domain.

The installation process comprises one 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 Issuer Security Domain verification, to ensure that the Card Issuer authorized the installation process.

The Install Token implies that the Card Issuer pre-authorized the data required for the delegated installation process. The pre-authorized data includes most of the INSTALL [for install] command.

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 Card Issuer 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.

Runtime Behavior

The OPEN shall provide the following additional services (see Section 6.4.1.2 - *Card Content Installation* for basic requirements) during the Delegated Installation process:

- Check that the Life Cycle of the Security Domain performing the installation is in the state of PERSONALIZED,
- Check that the Security Domain performing the installation has the Delegated Management Privilege,
- Check that the Security Domain performing the installation is the Security Domain associated with the Executable Module from which the Application is being installed,
- Request that the Issuer Security Domain verify the Install Token.
- On completion of the install procedure the OPEN shall:
  - Request the Issuer Security Domain to generate an Install Receipt.

Delegated Loading and Installation Flow

The processes of delegated loading and delegated installation may occur in a single Application Session where an Application is loaded and then installed immediately. Alternatively, the Delegated Loading and Delegated Installation processes may occur in separate Card Sessions.

Figure 7-3 is an example of loading and installing an Application on the card through a Security Domain with the Delegated Management privilege.

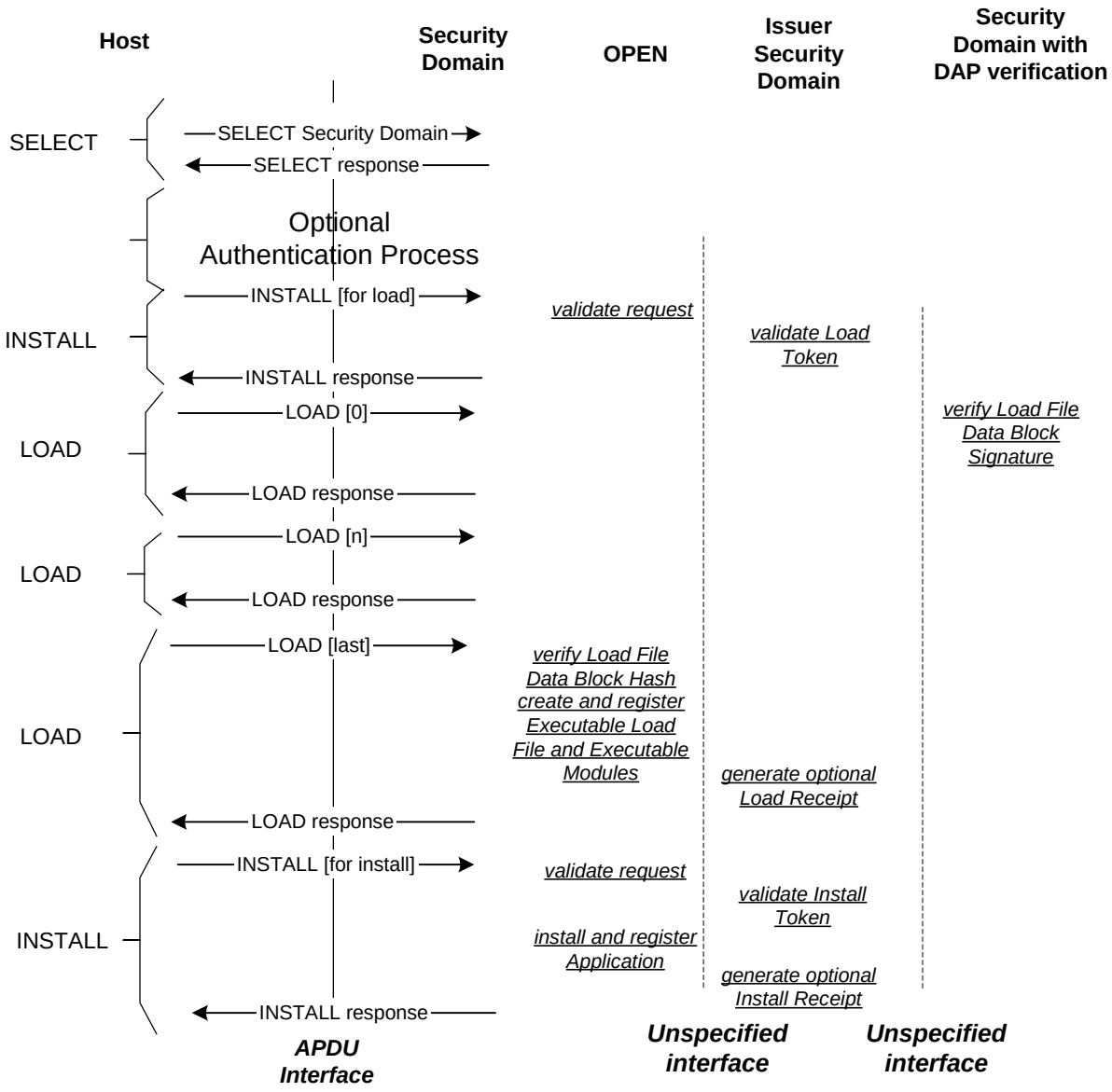


Figure 7-3: Delegated Loading and Installation

7.6.3 Delegated Extradition

Delegated Extradition allows an Application Provider to establish a Secure Channel for extraditing one of its Applications. The extradition may apply at any time during the Application Life Cycle.

The extradition process comprises one INSTALL [for extradition] command processed by the 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 Issuer Security Domain verification, to ensure that the Card Issuer authorized the extradition process.

The Extradition Token implies that the Card Issuer pre-authorized the data required for the delegated extradition process. The pre-authorized data includes most of the INSTALL [for extradition] command.

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 Card Issuer 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.

Runtime Behavior

The OPEN shall provide the following additional services (see Section 6.4.3 - *Content Extradition* for basic requirements) during the delegated extradition process:

- Check that the Life Cycle of the Security Domain performing the extradition is in the state of PERSONALIZED,
- Check that the Security Domain performing the extradition has the Delegated Management privilege,
- Request that the Issuer Security Domain verifies the Extradition Token,

On completion of the extradition procedure the OPEN shall provide the following service:

- Request that the Issuer Security Domain generates an optional Extradition Receipt.

7.6.4 Delegated Deletion

The Application Provider may instruct the OPEN to delete its associated Applications and Executable Load Files. The OPEN shall honor the deletion request without pre-authorization from the Card Issuer, i.e. no Token is required. The DELETE command is used for deleting Applications and Executable Load Files.

The delegated deletion process comprises one DELETE command processed by the Security Domain. The Security Domain then passes the deletion request to the OPEN for additional verification and processing.

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 Card Issuer 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.

Runtime Behavior

The OPEN shall provide the following services during the delegated deletion process:

- Check that the Life Cycle of the Security Domain performing the deletion is in the state of PERSONALIZED,
- Check that the Security Domain performing the deletion has the Delegated Management privilege,
- Check that the Security Domain performing the delete is the Security Domain associated with the Executable Load File or Application being deleted.

On completion of the deletion procedure the OPEN shall provide the following service:

- Request the Issuer Security Domain to generate an optional Delete Receipt.

Delegated Deletion Flow

Figure 7-4 is an example of deleting an Application or Executable Load File on the card through a Security Domain with the Delegated Management privilege:

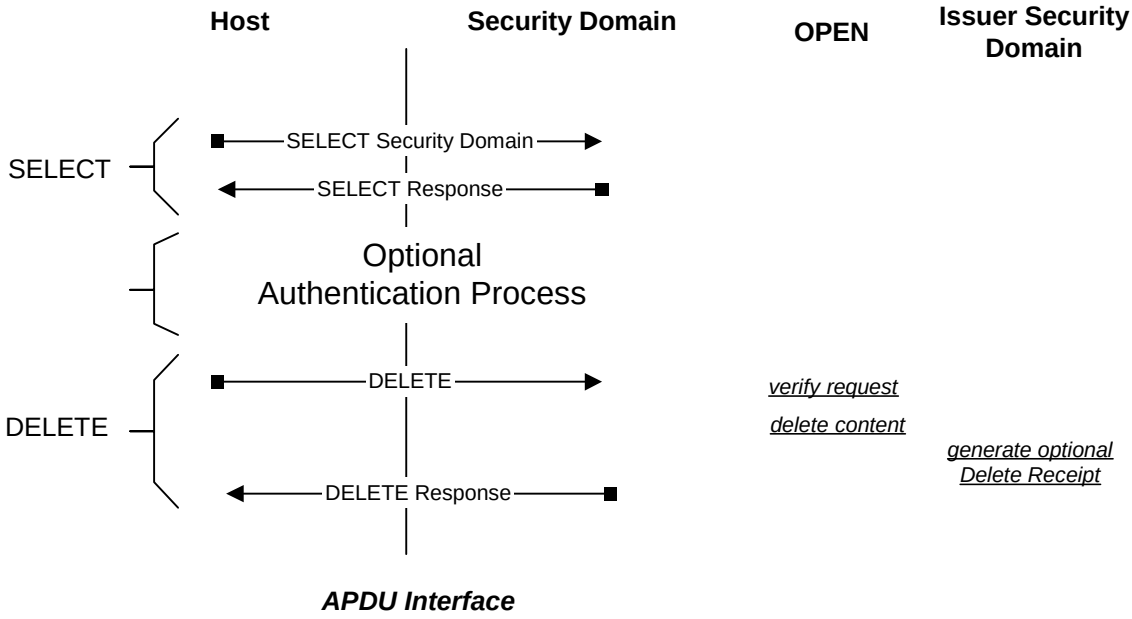


Figure 7-4: Delegated Deletion

7.7 Delegated Management Tokens and Receipts and DAP Verification

This Section defines the mechanisms, data elements and the format of Tokens and Receipts. It also describes the mechanisms and the data elements required for DAP Verification.

The algorithm for Token signatures is defined in Appendix B.3.

Token may be generated for use on multiple cards, depending on the Card Issuer's security policy

7.7.1 Load Token

The Load Token allows for the verification of the load process prior to actually processing the INSTALL [for load] command. The OPEN shall request the Issuer Security Domain to verify the Load Token.

The Load Token is a signature on most of the INSTALL [for load] command including the Load File Data Block Hash. The Token is appended to the INSTALL [for load] command.

The Load File Data Block Hash is included in the calculation to ensure that the Load File Data Block that has been approved by the generation of a Load Token is the same Load File Data Block that is subsequently received by the OPEN through the series of LOAD commands that follow the INSTALL [for load] command.

Once the complete Load File has been received, the OPEN shall verify that the Load File Data Block is valid using the Load File Data Block Hash. If the Load File Data Block Hash is not valid the OPEN shall abort the load process at this stage.

Figure 7-5 details the Load Token calculation performed by the Card Issuer.

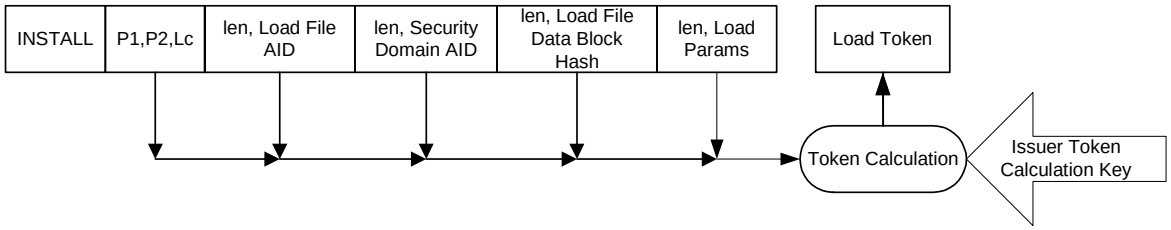


Figure 7-5 Load Token Calculation

7.7.2 Load Receipt

If the Card Issuer's security policy requires the generation of Receipts, the Load Receipt provides confirmation from the card that a successful load has occurred through the delegated loading process. The Load Receipt is comprised of data related to the delegated load process including Card Unique Data and generated by the Issuer Security Domain. The Receipt is included in the response message for the last LOAD command issued in a sequence of LOAD commands to the Security Domain.

Figure 7-6 details the Load Receipt calculation performed by the Issuer Security Domain.

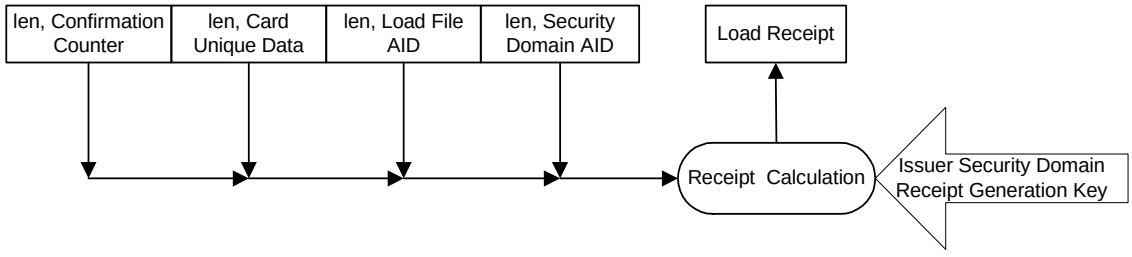


Figure 7-6: Load Receipt Calculation

7.7.3 Install and Extradition Tokens

The Install Token allows for the verification of the installation process prior to actually processing the INSTALL [for install] or [for make selectable] command. The Extradition Token allows for the verification of the extradition process prior to actually processing the INSTALL [for extradition] command. The OPEN shall request the Issuer Security Domain to verify the Install and Extradition Tokens. The Install and Extradition Tokens are a signature of the following fields within an INSTALL command and is appended to the INSTALL command. The same Token applies to the INSTALL [for install] command, the INSTALL [for make selectable] command and the INSTALL [for extradition] command.

Figure 7-7 details the Install and Extradition Tokens calculation performed by the Card Issuer.

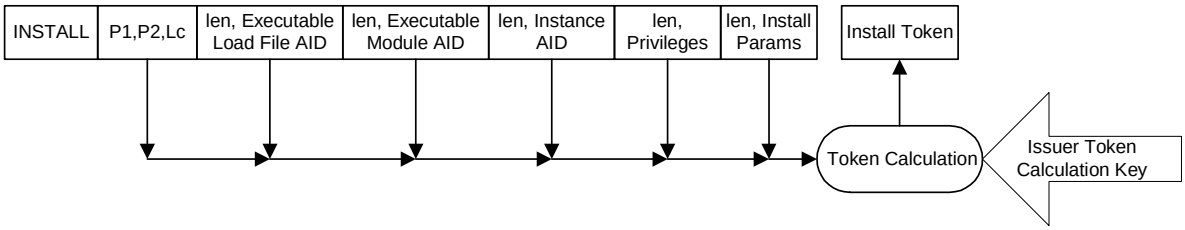


Figure 7-7: Install//Extradition Token Calculation

7.7.4 Install Receipt

If the Card Issuer's security policy requires the generation of Receipts, the Install Receipt provides confirmation from the card that a successful installation has occurred through the delegated installation process. The Install Receipt is comprised of data related to the delegated installation process including Card Unique Data generated by the Issuer Security Domain. The Install Receipt is returned in the response message to the INSTALL [for install] command sent to the Security Domain.

Figure 7-8 details the Install Receipt calculation performed by the Issuer Security Domain.

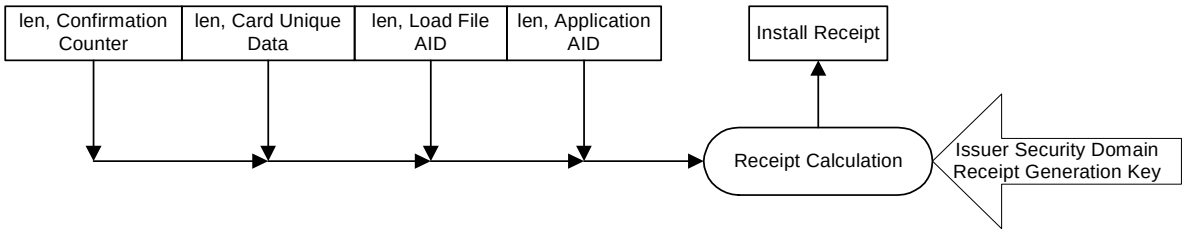


Figure 7-8: Install Receipt Calculation

7.7.5 Extradition Receipt

If the Card Issuer's security policy requires the generation of Receipts, the Extradition Receipt provides confirmation from the card that a successful extradition has occurred through the delegated extradition process. The Extradition Receipt is comprised of data related to the delegated extradition process including Card Unique Data generated by the Issuer Security Domain. The Extradition Receipt is returned in the response message to the INSTALL [for extradition] command issued to the Security Domain.

Figure 7-9 details the Extradition Receipt calculation performed by the Issuer Security Domain.

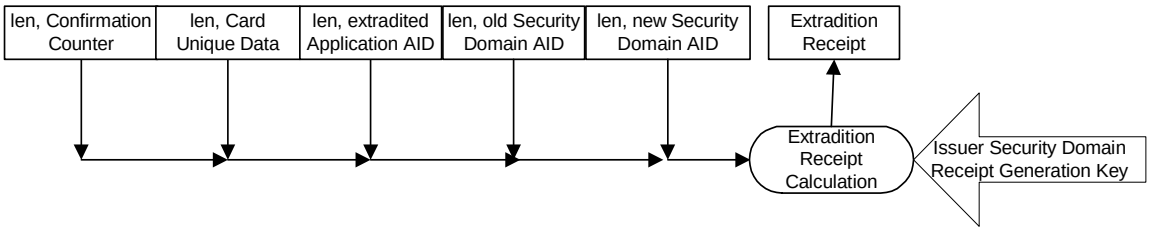


Figure 7-9: Extradition Receipt Calculation

7.7.6 Delete Receipt

If the Card Issuer's security policy requires the generation of Receipts, the Delete Receipt provides confirmation from the card that a successful deletion has occurred through the delegated deletion process. The Delete Receipt is comprised of data related to the delegated deletion process including Card Unique Data generated by the Issuer Security Domain. The Delete Receipt is returned in the response message to the DELETE command issued to the Security Domain.

Figure 7-10 details the Delete Receipt calculation performed by the Issuer Security Domain.

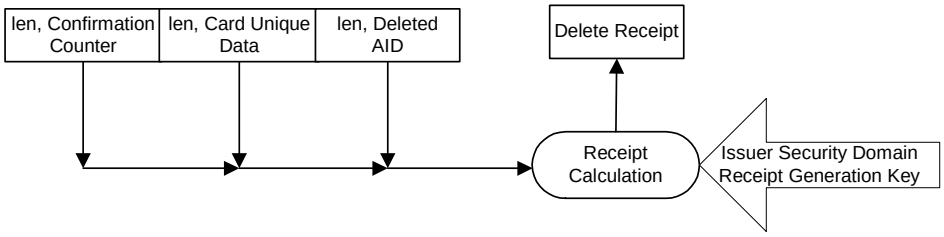


Figure 7-10: Delete Receipt Calculation

7.7.7 Load File Data Block Hash

The Load File Data Block Hash provides integrity of the Load File Data Block following receipt of the complete Load File Data Block. The OPEN shall verify the integrity of the Load File Data Block prior to creating an Executable Load File.

The Load File Data Block Hash is a message digest of the Load File Data Block. The Load File Data Block Hash is appended to the INSTALL [for load] command.

Figure 7-11 details the Load File Data Block Hash Calculation performed by the Issuer Security Domain, an Application Provider and a Controlling Authority.

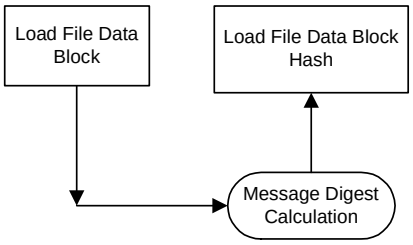


Figure 7-11: Load File Data Block Hash Calculation

7.7.8 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 7-12 details the Load File Data Block Signature Calculation performed by an Application Provider or Controlling Authority.

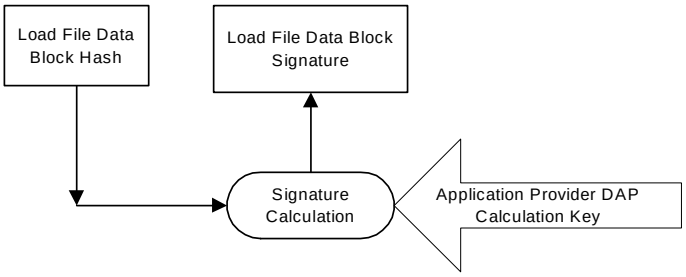


Figure 7-12: Load File Data Block Signature Calculation

8. Secure Communication

The Open Platform documentation broadly defines the notion of secure communication over and above that defined in ISO specifications. 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 imply that the ability to create a Secure Channel 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 the Issuer Security Domain, a Security Domain, or an Application may communicate with an off-card entity within a logically secure environment.

8.1 Secure Channel

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

A Secure Channel session is divided into three sequential phases:

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

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.

8.2 Explicit / Implicit Secure Channel

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

8.2.1 Explicit Secure Channel Initiation

Initiating the Secure Channel 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 (integrity and/or confidentiality). They also give the off-card entity the possibility of selecting the keys to be used.

8.2.2 Implicit Secure Channel Initiation

The Secure Channel 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.

8.2.3 Secure Channel Termination

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

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

- The card receives the first APDU command with an erroneous cryptographic protection;
- The card receives an APDU without the required cryptographic protection set up during Secure Channel Initiation;
- The on-card Application is de-selected, for instance another Application is selected (i.e. the Application Session ends); It may be the responsibility of the Application to initiate the termination of the Secure Channel when this occurs;
- The card is reset or powered off (i.e. the Card Session ends).

8.3 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.

8.4 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 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.

8.5 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.

8.6 Secure Channel Protocol Identifier

The Secure Channel Protocol Identifier 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'** – Secure Channel Protocol 01 defined in Appendix D *Secure Channel Protocol '01'*
- **'02'** – Secure Channel Protocol 02 defined in Appendix E *Secure Channel Protocol '02'*
- **'03' to '7F'** – Reserved for Future Use by GlobalPlatform
- **'80' to 'EF'** – Reserved for use by individual schemes registered by GlobalPlatform
- **'F0' to 'FF'** – Reserved for proprietary use and not registered by GlobalPlatform

The Secure Channel Protocol '01' is backward compatible with the Open Platform Card Specification version 2.0.1'.

The 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 Secure Channel Protocol '01'.

Part IV
APDU
Command
Reference

9. APDU Command Reference

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

Table 9-1 summarizes the APDU commands that shall be supported by the Issuer Security Domain and the requirements for support of these APDU commands by other Security Domains.

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED			TERMINATED		
	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD
DELETE Load File															
DELETE Application	✓			✓			✓								
DELETE Key															
GET DATA	✓			✓			✓			✓			✓		
GET STATUS	✓			✓			✓			✓					
INSTALL [for load]															
INSTALL [for install] (*)	✓	✓		✓	✓		✓	✓							
INSTALL [for make selectable](*)	✓	✓		✓	✓		✓	✓							
INSTALL [for extradition]															
INSTALL [for personalization]															
LOAD															
PUT KEY	✓			✓			✓								
SELECT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓					
SET STATUS	✓			✓			✓			✓					
STORE DATA	✓			✓			✓								

Table 9-1: Authorized OP Commands per Card Life Cycle State

Legend of Table 9-1:

ISD: Issuer Security Domain.

DM SD: Application Provider Security Domain with Delegated Management Privilege.

SD: Application Provider Security Domain without Delegated Management Privilege.

✓ : Support required.

Blank cell: Support optional.

Striped cell: Support prohibited.

(*) Note: The support of the combined [install and make selectable] options within the same INSTALL APDU command is also required

Table 9-2 summarizes the minimum security level required for the APDU commands.

Command	Minimum Security Level
DELETE	Secure Channel Initiation
GET DATA	None
GET STATUS	Secure Channel Initiation
INSTALL	Secure Channel Initiation
LOAD	Secure Channel Initiation
PUT KEY	Secure Channel Initiation
SELECT	Not Applicable
SET STATUS	Secure Channel Initiation
STORE DATA	Secure Channel Initiation

Table 9-2: **Minimum Security Requirements for OP Commands**

9.1 General Coding Rules

In all the following tables, an 'x' in a cell means a value that the card shall ignore.

The following details the coding rules for:

- The Life Cycle States,
- The Application privileges,
- The general errors returned in the APDU responses,
- The class byte of the APDU commands.
- The Key Type

9.1.1 Life Cycle Status 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 9-3: 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 4-7, and the coding of these bits is undefined in this specification

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

Table 9-4: 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	X	X	1	1	LOCKED

Table 9-5: 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 9-6: Issuer Security Domain Life Cycle Coding

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

9.1.2 Application Privileges Coding

Application privileges are coded on one byte as described in the following table (See Section 6.6.2.4 - *Application Privileges* for additional details):

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	X	X	X	X	X	X	X	Security Domain
1	1	X	X	X	X	X	0	DAP Verification
1	X	1	X	X	X	X	X	Delegated Management
X	X	X	1	X	X	X	X	Card lock
X	X	X	X	1	X	X	X	Card terminate
X	X	X	X	X	1	X	X	Default Selected
X	X	X	X	X	X	1	X	CVM management
1	1	X	X	X	X	X	1	Mandated DAP Verification

Table 9-7: Application Privileges

An Application may support one or more of these Application privileges.

The Application Privileges individual bits have the following meaning:

b8=1 indicates that the Application is a Security Domain.

b7=1 indicates that the Security Domain has DAP Verification capability.

b6=1 indicates that the Security Domain has Delegated Management Privilege.

b5=1 indicates that the Application has the privilege to lock the card.

b4=1 indicates that the Application has the privilege to terminate the card.

b3=1 indicates that the Application has the Default Selected privilege.

b2=1 indicates that the Application has CVM management privilege.

b1=1 indicates that the Security Domain has mandated DAP Verification capability.

9.1.3 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
'69'	'82'	Security status not satisfied
'69'	'85'	Conditions of use not satisfied
'6A'	'86'	Incorrect P1 P2
'6D'	'00'	Invalid Instruction
'6E'	'00'	Invalid Class

Table 9-8: General Error Conditions

9.1.4 Class Byte Coding

The Class Byte of all Open Platform commands shall conform to ISO 7816-4 and shall be coded according to the following table:

CLA	Meaning
'00'	Command defined in ISO 7816
'80'	Proprietary command
'84'	Proprietary command with Secure Messaging

Table 9-9: CLA Byte Coding

Note that as support for Logical channels has not been defined in this version of the Specification, the class byte does not reflect logical channels as defined in ISO.

Note: In this version of the Specification only MACs are defined for integrity.

9.1.5 APDU command and response data

All OP commands comply with ISO 7816 short message lengths i.e. one byte.

9.1.6 Key Type Coding

The key type is coded on one byte according to the following table:

Value	Meaning
'00'-'7F'	Reserved for private use
'80'	DES – mode (EBC/CBC) implicitly known
'81'-'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
'A7'	RSA private key - Chinese Remainder DP1 component
'A8'	RSA private key - Chinese Remainder DQ1 component
'A9'-'FE'	RFU (asymmetric algorithms)
'FF'	Not Available

Table 9-10: Key Type Coding

9.1.7 Optional Receipts in Delegated Management Response Messages

If a Delegated Management Card Content APDU contains an optional confirmation containing a Receipt, the Receipt shall be structured according to the following table:

Length	Data Element
1	Length of Receipt
0-n	Receipt
1	Length of Confirmation Counter
1-n	Confirmation Counter
1	Length of Card Unique Data
1-n	Card Unique Data

Table 9-11: Confirmation Structure

Note: Card Unique Data is the concatenation without delimiters of the IIN and the CIN.

9.2 DELETE Command

9.2.1 Definition and Scope

The DELETE command is used to delete a uniquely identifiable object such as an Executable Load File, an Application, or a key. However, in order to delete an object, the object shall be uniquely identifiable by the selected Application, Security Domain or Issuer Security Domain.

9.2.2 Command Message

The DELETE command message is coded according to the following table:

Code	Value	Meaning
CLA	'80' or '84'	
INS	'E4'	DELETE
P1	'xx'	Always '00'
P2	'00	Always '00'
Lc	'xx'	Length of data field
Data	'xxxx…'	TLV coded objects (and MAC if present)
Le	'00'	

Table 9-12: DELETE Command Message

9.2.2.1 Reference Control Parameter P1

Reference Control Parameter P1 shall always be set to '00'.

9.2.2.2 Reference Control Parameter P2

Reference Control Parameter P2 is always set to '00'.

9.2.2.3 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(s) to be deleted.

Deleting an Executable Load File or an Application shall be specified using the tag for an AID '4F' followed by a length and the AID of the Executable Load File or Application to be deleted.

Deleting a key shall be specified using the following tags:

Key management	Tag	Length	Value
Delete a Key ID	‘D0’	‘01’	Key Identifier
Delete a Key Version	‘D2’	‘01’	Key Version Number

Table 9-13: DELETE [key] Command Message Data Field

A single key is deleted when both the Key Identifier (‘D0’) and the Key Version (‘D2’) are provided in the DELETE command message data field.

9.2.3 Response Message

A response message shall always be returned. In the case where the Issuer Security Domain is processing the DELETE '00' shall be returned indicating that no additional response data is present. In the case of Delegated Management a DELETE Confirmation may be returned.

9.2.3.1 Data Field Returned in the Response Message

The data field of the response message may contain the confirmation of the delete procedure. A Delete Confirmation may be present for Delegated Management depending on the security policy of the Card Issuer.

The two following tables describe the structure of the DELETE Confirmation.

Presence	Length in Bytes	Name
Mandatory	1	Length of delete confirmation
Conditional	0-n	Delete confirmation (see Section 9.1.7)

Table 9-14: DELETE Confirmation Processing State Returned in the Response Message

A successful execution of the command shall be coded by '90' '00'.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'65'	'81'	Memory failure
'69'	'85'	Referenced data cannot be deleted
'6A'	'88'	Referenced data not found
'6A'	'82'	Application not found
'6A'	'80'	Incorrect values in command data

Table 9-15: DELETE Error Conditions

9.3 GET DATA Command

9.3.1 Definition and Scope

The GET DATA command is used to retrieve a single data object. P1 and P2 coding is used to define the specific data object tag. The data object may be a key.

9.3.2 Command Message

The GET DATA command message is coded according to the following table:

Code	Value	Meaning
CLA	‘00’, '80' or '84'	
INS	'CA'	GET DATA
P1	'xx'	'00' or high order tag value
P2	'xx'	Low order tag value
Lc	'xx'	Not present or length of MAC
Data	'xxxx…'	MAC if present
Le	'00'	

Table 9-16: GET DATA Command Message

9.3.2.1 CLA Byte

A value ‘00’ of the CLA Byte defines the GET DATA Command with the ISO 7816-4 structure.

A value ’80’ or ‘84’ defines the Open Platform structure of the GET DATA command.

9.3.2.2 Parameter P1 and P2

Parameters P1 and P2 define the tag of the data object to be read.

The Issuer Security Domain shall support at least the following data object tags:

- Tag ‘42’: Issuer Identification Number
- Tag ‘45’: Card Image Number
- Tag ‘66’: Card Recognition Data
- Tag ‘E0’: Key Information Template
- Tag ‘C1’ – Sequence Counter of the default Key Version Number

An Application Provider Security Domain shall support at least the following data object tag:

- Tag ‘E0’: Key Information Template
- Tag ‘C1’ – Sequence Counter of the default Key Version Number

9.3.2.3 Data Field Sent in the Command Message

The data field of the GET DATA command message shall be empty unless a MAC is required.

9.3.3 Response Message

9.3.3.1 Data Field Returned in the Response Message

For an Open Platform type GET DATA command, the GET DATA Response Data Field shall contain the TLV coded data object referred to in parameters P1 and P2 of the command message.

For an ISO type GET DATA command, the GET DATA Response Data Field shall contain only the value of the TLV coded data object referred to in parameters P1 and P2 of the command message.

When retrieving Key Information for the currently selected Application, the Key Information is returned in the template ‘E0’ where each Key Information data is introduced by the tag ‘C0’ and is structured as follows:

Parameter	Length	Coding
Key Id	1 Byte	Proprietary
Key Version Number	1 Byte	Proprietary
Key type of first (or only) key component	1 Byte	See Section 9.1.6 - *Key Type Coding*
Length of first (or only) key component	1 Byte	1-255
Key type of last key component	1 Byte	See Section 9.1.6 - *Key Type Coding*
Length of last key component	1 Byte	1-255

Table 9-17: Key Information Data Structure

9.3.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be coded by '90' '00'.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or the following error conditions.

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table 9-18: GET DATA Error Conditions

9.4 GET STATUS Command

9.4.1 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.

GET STATUS is the complementary command to SET STATUS.

9.4.2 Command Message

The GET STATUS command message shall be coded according to the following table.

Code	Value	Meaning
CLA	'80' or '84'	
INS	'F2'	GET STATUS
P1	'xx'	Reference Control Parameter P1
P2	'xx'	Reference Control Parameter P2
Lc	'xx'	Search criteria (and MAC if present)
Le	'00'	

Table 9-19: GET STATUS Command Message

9.4.2.1 Reference Control Parameter P1

Reference Control Parameter P1 is used to select a subset of statuses to be included in the response message.

The following values of the Reference Control Parameter may apply:

'80' – Issuer Security Domain only. In this case the search criteria is ignored and the Issuer Security Domain information is returned.

'40' – Applications and Security Domains only.

'20' – Executable Load Files only.

'10' – Executable Load Files and their Executable Modules only.

Other values for this Reference Control Parameter may be assigned in the future.

The ability to differentiate a Security Domain from an Application is achieved via the Application privileges.

9.4.2.2 Reference Control Parameter P2

The Reference Control parameter P2 controls the number of consecutive GET STATUS command. It shall be coded according to Table 9-20.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
X	X	X	X	X	X	X		RFU
							0	Get first or all occurrence(s)
							1	Get next occurrence(s)

Table 9-20: GET STATUS Reference Control parameter P2

The Get Next Occurrence does not apply when retrieving only the status of the Issuer Security Domain.

9.4.2.3 Data Field Sent in the Command Message

The GET STATUS command message data field contains the TLV coded search qualifier(s).

The tag '4F' shall be used to indicate the Application Identifier (AID). 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 Applications with the same RID.

9.4.3 Response Message

9.4.3.1 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 Parameters P1 and P2, multiple occurrences of the data structure in the following tables may be returned.

Length	Value	Meaning
1	'xx'	Length of AID
1-n	'xxxx…'	AID
1	'xx'	Life Cycle State
1	'xx'	Application Privileges

Table 9-21: Issuer Security Domain, Application and Executable Load File Information Data

Length	Value	Meaning
1	'xx'	Length of Executable Load File AID
1-n	'xxxx…'	Executable Load File AID
1	'xx'	Executable Load File Life Cycle State
1	'xx'	Application Privileges
1	'xx'	Number of associated Executable Modules
1	'xx	Length of Executable Module AID
1-n	'xxxx…'	Executable Module AID
		…
1	'xx'	Length of Executable Module AID
1-n	'xxxx…'	Executable Module AID

Table 9-22: Executable Load File and Executable Module Information Data

The Issuer Security Domain, Application, Executable Load File Life Cycle State and the Application Privileges are coded according to the general coding rules as defined in Section 9.1 - *General Coding Rules*.

9.4.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be coded by '90' '00'.

The command may return the following warning condition to indicate that a subsequent GET STATUS [get next occurrence(s)] may be issued to retrieve additional data :

SW1	SW2	Meaning
'63'	'10'	More data available

Table 9-23: GET STATUS Warning Conditions

This command may either return a general error condition as listed in Section 9.1.3 - *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 9-24: GET STATUS Error Conditions

9.5 INSTALL Command

9.5.1 Definition and Scope

The INSTALL command is issued to a Security Domain to initiate or perform the various steps required for Card Content management.

9.5.2 Command Message

The INSTALL command message shall be coded according to the following table.

Code	Value	Meaning
CLA	'80' or '84'	
INS	'E6'	INSTALL
P1	'xx'	Reference control parameter P1
P2	'00'	Unused'
Lc	'xx'	Length of data field
Data	'xxxx…'	Install data (and MAC if present)
Le	'00'	

Table 9-25: INSTALL Command Message

9.5.2.1 .Reference Control Parameter P1

The Reference Control Parameter P1 of the INSTALL command is coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
		1	0	0	0	0	0	For personalization
		0	1	0	0	0	0	For extradition
		0	0	1	X	0	0	For make selectable
		0	0	X	1	0	0	For install
		0	0	0	0	1	0	For load
X	X							RFU

Table 9-26: INSTALL Command Reference Control Parameter P1

b6 to b1 shall be coded as follows:

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 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

Combination of the [for install] and [for make selectable] options may apply.

9.5.2.2 Reference Control Parameter P2

The Reference Control Parameter of the INSTALL command is RFU and shall be coded '00'.

9.5.2.3 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.

9.5.2.3.1 Data Field for INSTALL [for load]

The following table details the INSTALL [for load] command data field.

Presence	Length in Bytes	Name
Mandatory	1	Length of Load File AID
Mandatory	5-16	Load File AID
Mandatory	1	Length of Security Domain AID
Conditional	0-16	Security Domain AID
Mandatory	1	Length of Load File Data Block Hash
Conditional	0-n	Load File Data Block Hash
Mandatory	1	Length of Load Parameters field
Conditional	0-n	Load Parameters field
Mandatory	1	Length of Load Token
Conditional	0-n	Load Token

Table 9-27: INSTALL [for load] Command Data Field

The Load File Data Block Hash and Load Token are mandatory for Delegated Management. The Load File Data Block Hash is mandatory if the Load File contains one or more DAP Blocks.

9.5.2.3.2 Data Field for INSTALL [for install]

The following table details the INSTALL [for install] command data field.

Presence	Length in Bytes	Name
Mandatory	1	Length of Executable Load File AID
Mandatory	5-16	Executable Load File AID
Mandatory	1	Length of Executable Module AID
Mandatory	5-16	Executable Module AID
Mandatory	1	Length of Application AID
Mandatory	5-16	Application AID
Mandatory	1	Length of Application Privileges
Mandatory	1	Application Privileges
Mandatory	1	Length of Install Parameters field
Mandatory	2-n	Install Parameters field
Mandatory	1	Length of Install Token
Conditional	0-n	Install Token

Table 9-28: INSTALL [for install] Command Data Field

The Install Token is mandatory for Delegated Management. The install token shall not be present if Delegated Management is not used.

The Executable Module AID is the AID of the Executable Module previously loaded. The Executable Module shall be present within an Executable Load File.

Open Platform cards use the instance AID to indicate the AID with which the installed Application will be selected.

This version of the Open Platform Card Specification requires a single byte for Application Privileges. If an Application is only being installed and not made selectable with the same INSTALL command the Default Selected privilege cannot be set.

The instance AID, the Application Privileges and the Application specific Install Parameters shall be made known to the Application.

9.5.2.3.3 Data Field for INSTALL [for make selectable]

The following table details the INSTALL [for make selectable] command data field.

Presence	Length in Bytes	Name
Mandatory	1	Length = '00'
Mandatory	1	Length = '00'
Mandatory	1	Length of Application AID
Mandatory	5-16	Application AID
Mandatory	1	Length of Application Privileges
Mandatory	1	Application Privileges
Mandatory	1	Length = '00'
Mandatory	1	Length of Install Token
Conditional	0-n	Install Token

Table 9-29: INSTALL [for make selectable] Command Data Field

If the Default Selected privilege is set in the Application Privileges field, the Open Platform Registry shall be updated according to the rules defined in Section 6.6.2.4 - *Application Privileges*. Any other privilege set in the Application Privileges field is ignored by the card.

The Install Token is mandatory for Delegated Management for an INSTALL [for make selectable] command.

9.5.2.3.4 Data Field for INSTALL [for extradition]

The following table details the INSTALL [for extradition] command data field.

Presence	Length in Bytes	Name
Mandatory	1	Length of Security Domain AID
Mandatory	5-16	Security Domain AID
Mandatory	1	Length = '00'
Mandatory	1	Length of Application AID
Mandatory	5-16	Application AID
Mandatory	1	Length = '00'
Mandatory	1	Length = '00'

Presence	Length in Bytes	Name
Mandatory	1	Length of Extradition Token
Conditional	0-n	Extradition Token

Table 9-30: INSTALL [for extradition] Command Data Field

The Security Domain AID indicates to which Security Domain this Application is being extradited. The Security Domain to which this Application is currently associated is the currently selected Application.

9.5.2.3.5 Data Field for INSTALL [for personalization]

The following table details the INSTALL [for personalization] command data field.

Presence	Length in Bytes	Name
Mandatory	1	Length = '00'
Mandatory	1	Length = '00'
Mandatory	1	Length of Application AID
Mandatory	5-16	Application AID
Mandatory	1	Length = ‘00’
Mandatory	1	Length = '00'
Mandatory	1	Length = ‘00’

Table 9-31: INSTALL [for personalization] Command Data Field

9.5.2.3.6 INSTALL [for load] and INSTALL [for install] 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.

The following table identifies the possible tags for use in the Load Parameter field:

Tag	Length	Value (Name)	Presence
'EF'	Variable	System specific parameters	Conditional
'C6'	2	Non volatile code space limit	Optional
'C7'	2	Volatile data space limit	Optional
‘C8’	2	Non volatile data space limit	Optional

Table 9-32: Load Parameter Tags

The presence of tags ‘C6’, ‘C7’, or ‘C8’, in the INSTALL [for load] command indicates the minimum size of the resources that are requested to be available on the card. This indication enables the OPEN to reject the load request if the remaining card resources are insufficient for either the load or a subsequent installation.

The following table identifies the possible tags for use in the Install Parameter field:

Tag	Length	Value (Name)	Presence
'C9'	Variable	Application specific parameters	Mandatory
'EF'	Variable	System specific parameters	Conditional
'C7'	2	Volatile data space limit	Optional
'C8'	2	Non volatile data space limit	Optional

Table 9-33: Install Parameter Tags

When present in the INSTALL [for Install] command, the information contained in tag 'C9' (length + data) shall be passed to the Application being installed. The presence of tags 'C7' or 'C8' in the INSTALL [for Install] command indicates the maximum size of card resources allocated to the Application. This information may be used by the OPEN to validate Application's resources requests.

9.5.3 Response Message

A data field shall always be returned in the response message. The content of the response message data field is only relevant in the case of Delegated Management.

In the case where the Issuer Security Domain processes the INSTALL, a single byte of '00' shall be returned indicating that no additional data is present in the response message data field.

If an INSTALL [for load] or [for personalization] is being issued to a Security Domain with Delegated Management privileges, a single byte of '00' shall also be returned indicating that no additional response data is present.

9.5.3.1 Data Field Returned in the Response Message

For INSTALL commands sent during Delegated Management, the data field of the response message contains the confirmation of the install procedure. An install confirmation may be present for Delegated Management depending on the security policy of the Card Issuer. The overall length of the response message shall not exceed 256 bytes.

The following table describes the structure of the INSTALL response data field.

Presence	Length in Bytes	Name
Mandatory	1	Length of install confirmation
Conditional	0-n	Install confirmation (see Section 9.1.7)

Table 9-34: INSTALL Response Message Data Field

9.5.3.2 Processing State Returned in the Response Message

A successful execution of the command shall coded by '90' '00'. This command returns general error conditions.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'65'	'81'	Memory failure
'6A'	'80'	Incorrect parameters in data field

SW1	SW2	Meaning
'6A'	'84'	Not enough memory space
'6A'	'88'	Referenced data not found

Table 9-35: INSTALL Error Conditions

9.6 LOAD command

9.6.1 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 ICC internal handling or storage of the Load File is beyond the scope of this specification.

Multiple LOAD APDU 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 APDU 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.

9.6.2 Command Message

The LOAD command message is coded according to the following table.

Code	Value	Meaning
CLA	'80' or '84'	
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 9-36: LOAD Command Message Structure

The overall length of the command message shall not exceed 256 bytes.

9.6.2.1 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 9-37: LOAD Command Parameter P1

9.6.2.2 Reference Control Parameter P2

Reference Control Parameter P2 contains the block number, and shall be coded sequentially from '00' to 'FF'.

9.6.2.3 Data Field Sent in the Command Message

The data field of the command message contains a portion of the Load File.

A complete Open Platform Load File is structured as defined in the following table:

Tag	Length	Value (Name)
'E2'	Variable	DAP Block
'4F'	5-16	Security Domain AID
'C3'	Variable	Load File Data Block Signature
:	:	:
:	:	:
'E2'	Variable	DAP Block
'4F'	5-16	Security Domain AID
'C3'	Variable	Load File Data Block Signature
'C4'	Variable	Load File Data Block

Table 9-38: Load File Structure

The data objects defined in **Error! Reference source not found.** shall be coded according to ASN.1 Basic Encoding Rules.

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.

9.6.3 Response Message

A data field shall always be returned in the response message.

9.6.3.1 Data Field Returned in the Response Message

The data field is one byte of zeros except where there is a Load Confirmation to be returned. A Load confirmation may be returned on the last block of a Load command during Delegated Management. Whether it is actually returned depends on the Card Issuer’s security policy. The overall length of the response message shall not exceed 256 bytes.

The following table defines the structure of the LOAD response data field.

Presence	Length in bytes	Name
Mandatory	1	Length of load confirmation
Conditional	0-n	Load confirmation (see Section 9.1.7)

Table 9-39: LOAD Response Data Field

9.6.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be coded by '90' '00'.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'65'	'81'	Memory failure
'6A'	'84'	Not enough memory space
'6A'	'86'	Incorrect P1/P2
'69'	'85'	Conditions of use not satisfied

Table 9-40: LOAD Error Conditions

9.7 PUT KEY command

9.7.1 Definition and Scope

The PUT KEY command is used to either:

- Replace an existing key with a new key: The new key has 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 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; or
- 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.

9.7.2 Command Message

The PUT KEY command message is coded according to the following table:

Code	Value	Meaning
CLA	'80' or '84'	
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 9-41: PUT KEY Command Message

The overall length of the command message shall not exceed 256 bytes.

9.7.2.1 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
	*	*	*	*	*	*	*	Key Version Number

Table 9-42: PUT KEY Reference Control Parameter P1

9.7.2.2 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, P2 indicates the Key Identifier of this key. When multiple keys are contained in the command message data field, 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
	*	*	*	*	*	*	*	Key Identifier

Table 9-43: PUT KEY Reference Control Parameter P2

9.7.2.3 Data Field Sent in the Command Message

9.7.2.3.1 Format 1

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):

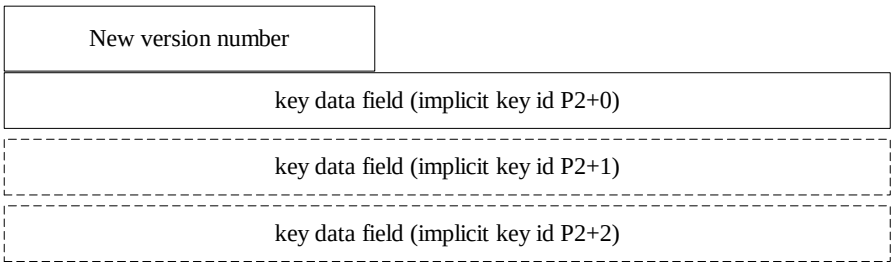


Table 9-44: Key Version Number Diagram

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 key data field is structured according to the following table:

Length	Meaning
1	Key type
1	Length of key or key component
Variable: 1-n	Key or key component data value
1	Length of key check value
Variable: 0-n	Key check value (if present)

Table 9-45: KEY Data Field

9.7.2.3.2 Format 2

Reserved for Future Use

9.7.2.3.3 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 8 of P1 = 0) in an atomic manner.

The PUT KEY command creates or updates the Key Information data structured as in Table Table 9-17 and contained in the tag ‘C0’.

The modulus component of a 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.

9.7.3 Response Message

9.7.3.1 Data Field Returned in the Response Message

9.7.3.1.1 Format 1

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).

9.7.3.1.2 Format 2

Reserved for Future Use

9.7.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be coded by '90' '00'.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
‘6A’	‘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 9-46: PUT KEY Error Conditions

9.8 SELECT command

9.8.1 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 other options shall be passed to the selected Application.

9.8.2 Command Message

The SELECT command is coded according to the following table:

Code	Value	Meaning
CLA	'00'	General command
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 9-47: SELECT Command Message

9.8.2.1 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
					1			Select by name

Table 9-48: SELECT Reference Control Parameter P1

9.8.2.2 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	First or only occurrence
						1	0	Next occurrence

Table 9-49: SELECT Reference Control Parameter P2

9.8.2.3 Data Field Sent in the Command Message

The data field of the command shall contain the AID of the Application to be selected. The data field may be omitted if the Issuer Security Domain is being selected.

9.8.3 Response Message

9.8.3.1 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 F for detailed coding)	Optional
'9F6E'	Application production life cycle data	Optional
'9F65'	Maximum length of data field in command message	Mandatory

Table 9-50: File Control Information

9.8.3.2 Return Processing State

A successful execution of the command shall be coded by '90' '00'.

The command may return the following warning condition when the Issuer Security Domain is being selected.

SW1	SW2	Meaning
'62'	'83'	Card Life Cycle State is CARD_LOCKED

Table 9-51: SELECT Warning Condition

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'6A'	'81'	Function not supported e.g. card Life Cycle State is CARD_LOCKED
'6A'	'82'	Selected Application / file not found

Table 9-52: SELECT Error Conditions

9.9 SET STATUS command

9.9.1 Definition and Scope

The SET STATUS command shall be used to modify the card Life Cycle State or the Application Life Cycle State.

9.9.2 Command Message

The SET STATUS command message is coded according to the following table.

Code	Value	Meaning
CLA	'80' or '84'	
INS	'F0'	SET STATUS
P1	'xx'	Status Type parameter
P2	'xx'	State control parameter P2
Lc	Lc	Length of data field
Data	'xxxxx…'	AID of Application (and MAC if present)
Le		Not present

Table 9-53: SET STATUS Command Message

9.9.2.1 Reference Control Parameter P1 - Status Type Parameter

The Status Type Parameter of the SET STATUS command message indicates if the change in the Life Cycle State applies to the Issuer Security Domain or an Application. Security Domains other than the Issuer Security Domain are considered as Applications in this context. The Status Type Parameter shall be coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0							Indicate Issuer Security Domain
0	1							Indicate Application
		X	X	X	X	X	X	RFU

Table 9-54: SET STATUS – Status Type (P1)

9.9.2.2 Reference Control Parameter P2 - State Control Parameter

The State Control Parameter of the SET STATUS command message indicates the State Transition required and shall be coded according to Section 9.1.1 - *Life Cycle Status Coding*.

For transitions to a higher Life Cycle State, only the bit indicating the Life Cycle State to which the Issuer Security Domain or the Application shall transition shall be set.

For the Issuer Security Domain (card), the parameter shall be coded according to Table 9-6 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 transition is to the PERSONALIZED state: the parameter shall be coded according to Table 9-5.

For an Application setting its own Life Cycle State using this command, the parameter shall be coded according to Table 9-4 and abide by the transitioning rules as diagramed in Figure 5-2.

For the Card Issuer setting the Life Cycle State of an 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 8 (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.

9.9.2.3 Data Field Sent in the Command Message

The data field shall contain the AID of the Application. If P1 is ‘80’ the content of the command data field shall be ignored.

9.9.3 Response Message

9.9.3.1 Data Field Returned in the Response Message

The data field of the response message shall not be present.

9.9.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be coded by '90' '00'.

This command may either return a general error condition as listed in Section 9.1.3 - *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 9-55: SET STATUS Error Conditions

9.10 STORE DATA command

9.10.1 Definition and Scope

The STORE DATA command is used to transfer data to an Application or the Security Domain processing the command.

The Issuer Security Domain or 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.

9.10.2 Command Message

The STORE DATA command message shall be coded according to the following table.

Code	Value	Meaning
CLA	'80' or '84'	
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	‘00’	

Table 9-56: STORE DATA Command Message

The overall length of the command message shall not exceed 256 bytes.

9.10.2.1 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
	X	X	X	X	X	X	X	RFU

Table 9-57: STORE DATA Reference Control Parameter P1

9.10.2.2 Reference Control Parameter P2

Reference control parameter P2 shall contain the block number coded sequentially from ‘00’ to ‘FF’. The Security Domain shall check the sequence of commands.

9.10.2.3 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, this data except the MAC (if present) shall be transparent to the Security Domain.

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')

9.10.3 Response Message

9.10.3.1 Data Field Returned in the Response Message

The data field of the response message shall not be present.

9.10.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be coded by '90' '00'.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or the following error condition.

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in command data

Table 9-58: STORE DATA Error Condition

Appendices

A. OPEN PLATFORM API

This section contains both the API required for backwards compatibility with Open Platform 2.0.1' Java Cards as well as Open Platform 2.1 Java Cards. Over time the Open Platform 2.0.1' API will be deprecated but in the short term, implementations intended to support Applications coded to this previous API shall contain both the old API as well as the new API.

The following is a list of changes between the Open Platform 2.0.1' API and the Open Platform 2.1 API:

The package naming has changed.

- The CVM methods have been expanded and moved into an optional interface.
- The management of Secure Channels has been modified in an attempt to make it more generic and support multiple Secure Channel Protocols.
- A new sharable interface has been added in order to support Security Domains ability to pre-process data on behalf of an Application.

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.

A.1 Deprecated Open Platform Java Card API

Open Platform Specific

Open Platform and the definition of the Card Manager integrate with the standard Java Card™ 2.1 specifications with the following minor exceptions.

Installation

The Java Card JCRE specification defines the parameters for the Applet.install() method to identify the following information:

- the application specific parameters.

The Open Platform requires the parameters for the Applet.install() method to identify the following information:

- the instance AID;
- the Application privileges; and,
- the Application specific parameters.

This requirement does not require a change to the API but rather to the Runtime Environment (in this case the Card Manager) and the expectation of an Open Platform Application.

The install() method parameter fields identify the above information retrieved from the INSTALL (for install) command in the following format:

- a buffer containing the following consecutive LV coded data;
  - length of the instance AID;
  - the instance AID;
  - length of the application privileges;
  - the application privileges;
  - length of application specific parameters; and
  - the application specific parameters.
- an offset within the buffer pointing to the length of the instance AID; and
- a length indicating the total length of the above data.

The application is required to utilize the instance AID as a parameter when invoking the Java Card register(byte[] bArray, short bOffset, byte bLength) method.

Selection

Open Platform does not require the Applet.select() method to return any other value but true. In addition if the select() method does fail, an '6999' response code is not expected from an Open Platform card and there should not be a scenario that causes no application to be selected (i.e. if no other application can be selected, the Card Manager is the selected application).

Cryptographic Algorithm

Open Platform cards supporting RSA cryptography require support for an additional algorithm not defined in the Java Card™ API specification. This additional algorithm will fulfill the requirement identified in section **Error! Reference source not found.** 'Cryptographic support'.

The RSA_NO_PAD algorithm is required in the javacardx.crypto.cipher class.

Application Programmer Interface

The following package details an example of the API required for an Open Platform 2.0.1' card or a Open Platform 2.1 card that intends to be backwards compatible with Open Platform 2.0.1' Applications.

Invocation of Open Platform methods

Some Open Platform methods require the application invoking the method to be the selected application at the time the method is invoked. These are primarily the methods that require the Card Manager to locate the invoking application's entry in the registry.

This restriction applies directly to the install() method of the application which is restricted in the Open Platform methods it can invoke.

The following is a list of methods that can be invoked from the install() method:

- getTriesRemaining;
- getCardManagerState; and
- verifyPin.

All other Open Platform methods can only be invoked when the application is the selected application. This of course also relates to all Security Domain methods which, while in themselves, do not require the location of the application's entry in the registry, do need to be preceded by the getSecurityDomain() method.

Package Index

Other Packages

- package openplatform

package openplatform

Interface Index

- ProviderSecurityDomain

Class Index

- OPSystem

Class Hierarchy

class java.lang.Object

- class openplatform.OPSystem
- interface openplatform.ProviderSecurityDomain

Interface openplatform.ProviderSecurityDomain

public abstract interface ProviderSecurityDomain

This defines the interface of a privileged system class that represents an Application Provider on a card. The class implementing this interface shall be declared a Shareable Interface Object (see the JCRE document in Java Card™ 2.1). This class offers cryptographic services, key management services, runtime messaging support, and secure loading services to applets from the same Application Provider. Prior to using this interface, an Application is required to obtain a handle to it's associated Security Domain by invoking the OPSystem.getSecurityDomain() method.

Method Summary	
Void	closeSecureChannel(byte channel) This method is used to close a Secure Channel.
Boolean	decryptVerifyKey (byte channel, APDU apdu, short offset) This method is used to decrypt and verify a new key.
Byte	openSecureChannel (APDU apdu) This method opens a Secure Channel.
Void	unwrap (byte channel, APDU apdu) This method is used to process an APDU buffer received within a Secure Channel.
Void	verifyExternalAuthenticate (byte channel, APDU apdu) This method is used to authenticate an off-card entity.

Method Detail

closeSecureChannel

public void closeSecureChannel(byte channel)

This method is used to close the Secure Channel that was previously opened with the openSecureChannel() method specifically to erase any secure information relating to the Secure Channel.

Parameters:

channel - byte

---

decryptVerifyKey

public boolean decryptVerifyKey(byte channel,
APDU apdu,
short offset)

This method is used to decrypt and verify a key received by the Application within a Secure Channel.

Parameters:

channel - byte Secure Channel number

apdu – APDU The APDU handle

offset - short Offset within the APDU buffer where the Key Set data field can be retrieved.

Returns:

TRUE if a key has been verified, FALSE otherwise.

openSecureChannel

public byte openSecureChannel(APDU apdu)

This method opens a Secure Channel for an Application and returns the newly opened channel number. The supplied APDU shall contain the command used to retrieve data from the card that will be used by the off-card entity to authenticate the card.

This method prepares the response to this command within the APDU. The Security Domain that the applet belongs to is responsible for the channel number allocation.

Parameters:

apdu - APDU

Returns:

channel number

unwrap

public void unwrap(byte channel,

APDU apdu)

This method is used to process the APDU content after receiving it from an off-card entity and within a Secure Channel. The processing is according to the requirements for integrity and confidentiality that are established when a Secure Channel is opened. The resultant APDU contains the command as if it where received outside of a Secure Channel.

Parameters:

channel - byte

apdu - APDU

verifyExternalAuthenticate

public void verifyExternalAuthenticate(byte channel,

APDU apdu)

This method is used to authenticate the off-card entity by verifying the contents of the APDU command.

Parameters:

channel - byte

apdu - APDU

Class openplatform.OPSystem

java.lang.Object

|

+---- openplatform.OPSystem

public final class OPSystem

extends Object

The OPSystem class exposes a subset of the behavior of the Card Manager to the outside world. It extends the functionality of the JCRE 2.1 API by providing card management services to Applications. It implements and enforces a Card Issuer's security policies. This class provides the functionality of a runtime environment running at the JCRE 'system' (privileged) context. The details of the JCRE 'system' context implementation are left to the implementer. This class's public interface is composed of static methods visible to all applets importing the Open Platform package. This is a 'singleton' class (only one instance is created for the lifetime of the card).

Field Summary	
Static byte	APPLET_BLOCKED Application Life Cycle State indicating the Application is BLOCKED = 0x7F
Static byte	APPLET_PERSONALIZED Application Life Cycle State indicating the Application is PERSONALIZED = 0x0F
Static byte	APPLET_SELECTABLE Application Life Cycle State indicating the Application is SELECTABLE = 0x03
Static byte	CARD_INITIALIZED Card Manager Life Cycle State indicating the Card Manager is INITIALIZED = 0x07
Static byte	CARD_SECURED Card Manager Life Cycle State indicating the Card Manager is SECURED = 0x0F
Static byte	CARD_LOCKED Card Manager Life Cycle State indicating the Card Manager is CARD_LOCKED = 0x7F
Static byte	CARD_OP_READY Card Manager Life Cycle State indicating the Card Manager is OP_READY = 0x01

Method Summary	
Static byte	getCardContentState() This method returns the Life Cycle State of the selected Application.
Static byte	getCardManagerState() This method returns the Life Cycle State of the Card Manager.
Static ProviderSecurityDomain	getSecurityDomain() This method returns a handle to the Application's associated Security Domain.
Static byte	getTriesRemaining() This method returns the PIN tries remaining for the CVM.
Static boolean	lockCardManager() This method allows an applet to lock the card.
Static boolean	setATRHistBytes (byte [] buffer, short bOffset, byte bLength) This method sets the historical bytes contained in the ATR (Answer To Reset).
Static boolean	setCardContentState(byte state) This method allows an Application to change its own life cycle state.
Static boolean	setPin(APDU apdu, short offset) This method is used to initialize the CVM
Static Boolean	terminateCardManager() This method allows an Application to terminate the card.
Static boolean	verifyPin(APDU apdu, short offset) This method allows an Application to check the validity of a PIN.

Field Detail

APPLET_SELECTABLE

public static final byte APPLET_SELECTABLE

The applet has reached the Life Cycle State of SELECTABLE and is available to receive SELECT commands from outside the card.

APPLET_SELECTABLE = 0x07

APPLET_PERSONALIZED

public static final byte APPLET_PERSONALIZED

The applet has been loaded with Application specific data and has transitioned itself to the Life Cycle State of PERSONALIZED.

APPLET_PERSONALIZED = 0x0F

APPLET_BLOCKED

public static final byte APPLET_BLOCKED

The Application, due to some off-card or internal event, has transitioned itself to the Life Cycle State of BLOCKED.

APPLET_BLOCKED = 0x7F

CARD_OP_READY

public static final byte CARD_OP_READY

The card is in the Open Platform state of OP_READY.

CARD_OP_READY = 0x01

CARD_INITIALIZED

public static final byte CARD_INITIALIZED

The Card Manager is in the Life Cycle State of INITIALIZED.

INITIALIZED = 0x07

CARD_SECURED

public static final byte CARD_SECURED

The Card Manager is in the Life Cycle State of SECURED.

CARD_SECURED = 0x0F

CARD_LOCKED

public static final byte CARD_LOCKED

The Card Manager has transition to the Life Cycle State of CARD_LOCKED.

CARD_LOCKED = 0x7F

Method Detail

getCardContentState

public static byte getCardContentState()

This method returns the Life Cycle State of the selected Application. The Card Manager locates the AID of the selected Application in the Open Platform Registry and returns the Life Cycle State.

Returns:

Life cycle state

See Also:

APPLET_SELECTABLE, APPLET_PERSONALIZED,

APPLET_BLOCKED

getCardManagerState

public static byte getCardManagerState()

This method returns the current Life Cycle State for the Card Manager.

Returns:

Life cycle state

See Also:

CARD_OP_READY, CARD_INITIALIZED, CARD_ SECURED,

CARD_LOCKED

getSecurityDomain

public static ProviderSecurityDomain getSecurityDomain()

This method returns the handle of the Application's associated Security Domain. The Card Manager locates the AID of the selected Application in the Open Platform Registry and determines the Application's associated Security Domain.

Returns:

ProviderSecurityDomain

---

getTriesRemaining

byte public static getTriesRemaining()

This method returns the PIN tries remaining for the CVM.

Returns:

PIN tries remaining.

---

lockCardManager

public static boolean lockCardManager()

This method allows an applet to lock the card. If the calling applet has been authorized to perform this operation and the operation is successful, this method returns TRUE. The Card Manager locates the AID of the selected Application in the Open Platform Registry and determines if the Application has the required privilege.

Returns:

TRUE if Card Manager locked, FALSE otherwise

setATRHistBytes

public static boolean setATRHistBytes (byte[] buffer,
short bOffset,
byte bLength)

This method sets the historical bytes contained in the ATR (Answer To Reset). The sequence of bytes will be set on a subsequent power-up or reset. Only the "default selected" Application may invoke this method. The Card Manager locates the AID of the selected Application in the Open Platform Registry and determines if the Application has the required privilege.

Parameters:

buffer – byte[] Array containing the ATR historical bytes.

bOffset – short Offset within the buffer where ATR historical bytes begin.

bLength – byte Length of the ATR historical bytes in the buffer.

Returns:

TRUE if ATR bytes set, FALSE otherwise.

setCardContentState

public static boolean setCardContentState(byte state)

This is used by an Application to transition its own Life Cycle State. The Card Manager locates the AID of the selected Application in the Open Platform Registry and changes the Application's Life Cycle State. The Card Manager is responsible for enforcing the state transition rules.

Parameters:

state – transition to this state.

Returns:

TRUE if the operation is successful, FALSE otherwise

See Also:

APPLET_SELECTABLE, APPLET_PERSONALIZED,

APPLET_BLOCKED

setPin

public static boolean setPin(APDU apdu,
short pOffset)

This method is used to change the value of the CVM. The length of the PIN is retrieved from the PIN structure. The Card Manager locates the AID of the selected Application in the Open Platform Registry and determines if the Application has the required privilege. If the PIN is changed, the PIN try counter shall be reset.

Parameters:

apdu - APDU

pOffset – identifies the location of starting byte of a structure containing the

PIN information.

Returns:

TRUE if the PIN value was changed, FALSE otherwise.

---

terminateCardManager

public static boolean terminateCardManager()

This method allows an applet to terminate the card. If the calling applet has been authorized to perform this operation and the operation is successful, this method does not return to the calling Application. The Card Manager locates the AID of the selected Application in the Open Platform Registry and determines if the Application has the required privilege.

Returns:

FALSE if the operation was not performed.

verifyPin

public static boolean verifyPin(APDU apdu,

short pOffset)

This method allows an Application to request the Card Manager to compare a PIN block with the CVM. If the comparison is successful, the retry counter shall be reset. If the comparison is unsuccessful, the retry counter shall be updated.

Parameters:

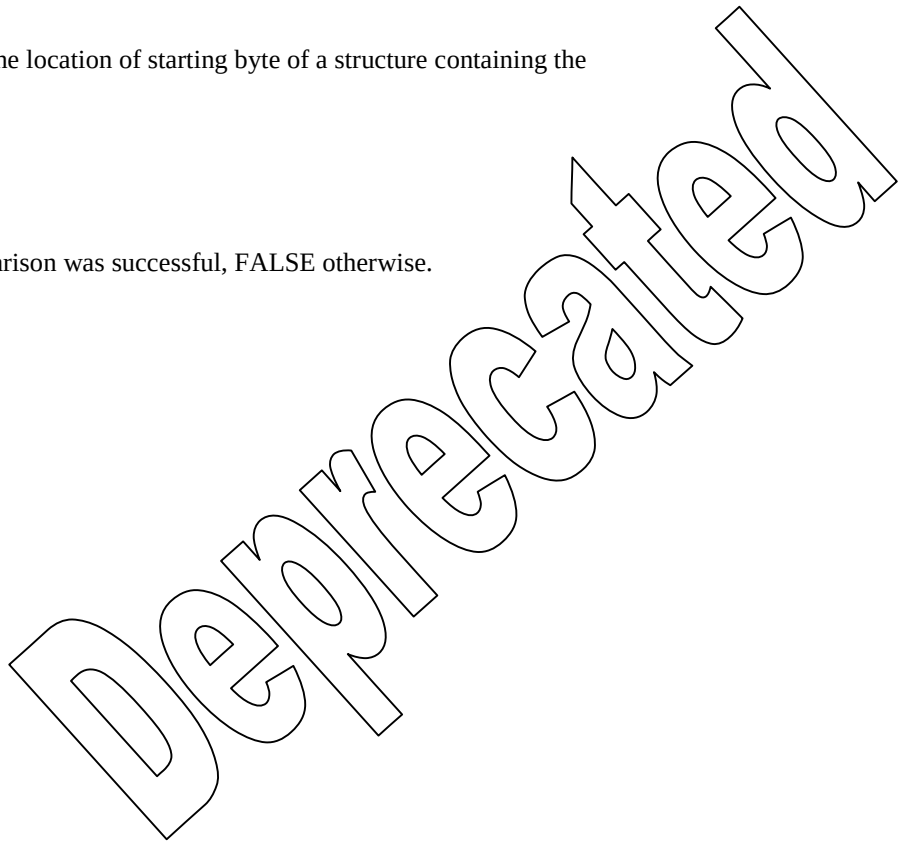
apdu - APDU

offset – identifies the location of starting byte of a structure containing the

PIN information.

Returns:

TRUE if the comparison was successful, FALSE otherwise.



A.2 Open Platform on a Java Card

Open Platform Specific Requirements

In order to ensure the highest level of interoperability of Open Platform implementations, GlobalPlatform also adopts the order defined in Section 6.2 Installation of the Java Card™ 2.1.1 Virtual Machine Specification.

The following 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* exist in an Open Platform implementation.

GlobalPlatform package AID

Each GlobalPlatform package AID will be a concatenation of a RID and a PIX. The exact values of the AID of each package will be defined once the RID mentioned in Appendix F - *GlobalPlatform Data Values and Card Recognition* Data have been assigned.

Installation

In *Section 3.1 The Method* `install` of the *Java Card™ 2.1.1 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.

In the definition of the install method of the Class Applet of the *Java Card™ 2.1.1 Application Programming Interface*, the install parameters to be supplied must be in the format defined by the applet.

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 an Open Platform card. It does not affect the definition of the install method of the Class Applet of the *Java Card™ 2.1.1 Application Programming Interface* specification.

The install parameters shall identify the following data, present in the INSTALL [for install] command (see section 9.5.2.3.2 - *Data Field for INSTALL [for install]*):

- The instance AID,
- The Application privileges and,
- The Application specific parameters[1].

[1] While the APDU 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.

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 Application privileges,
- The Application privileges,
- Length of the Application specific parameters and,
- 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™ 2.1.1 Application Programming Interface* specification.

T=0 Transmission Protocol

Open Platform cards are intended to be functional in the widest range of environments (i.e. Card Acceptance Devices). Currently the *Java Card™ 2.1.1 Runtime Environment (JCRE) Specifications* describes the behavior for case 2 commands (when using the T=0 protocol) in contradiction to EMV 2000. Open Platform mandates that the JCRE shall handle this case of command in accordance with ISO 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 stated of Open Platfrom API must conform to a transaction in progress.

All operations performed by this API, except the `Application.processData()` method shall be executed atomically.

Cryptographic Algorithms

Open Platform cards supporting RSA cryptography shall 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, shall be available.

Level of trust

The current Java Card 2.1.1 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™ 2.1.1 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™ 2.1.1 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. Open Platform 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 have a level of trust.

Invocation of Open Platform methods

The Application Programming Interface defined below is accessible to any Java Card applet developed with the intention of being present on an Open Platform 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™ 2.1.1 Application Programming Interface.* As this specification does not define exactly when the instance of an applet becomes an entry in the Card Open Platform Registry, an applet developer can only assume that this has occurred following the successful completion of the install method. To ensure interoperability, Open Platform API methods that require access to the Card Open Platform Registry of the applet invoking the method, shall not be invoked from within the constructor or `install()` method.

The following is a list of methods that may be invoked from within the constructor or the `install()` method:

- `getCardState`, and

  `getCVM`.

Class Hierarchy

- class java.lang.Object
    - interface org.globalplatform.**Application**
    - interface org.globalplatform.**SecureChannel**
    - class org.globalplatform.**GPSystem**
    - interface org.globalplatformx.**CVM**

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 SecureChannel

public interface **SecureChannel** extends javacard.framework.Shareable

This class 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 by invoking the GP`System.getSecureChannel()` method. The Secure Channel Interface shall only be exposed through the GP`System.getSecureChannel` method.

The byte value returned by the `getSecurityLevel()` method is a bit map indicating whether Entity Authentication has occurred and what level of security will be applied when invoking the wrap and unwrap methods. This byte value is coded according to the following table. Note that more than one security level may be set.

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

Table A- 1: Open Platform on a Java Card: Security Level

Field Summary

static byte	**AUTHENTICATED** Entity authentication has occurred (0x80).
static byte	**C_DECRYPTION** The unwrap method will decrypt incoming command data (0x02).
static byte	**C_MAC** The unwrap method will verify the MAC on an incoming command (0x01).
static byte	**NO_SECURITY_LEVEL** Entity Authentication has not occurred (0x00).
static byte	**R_ENCRYPTION** The wrap method will encrypt the outgoing response data (0x20).
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.
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 Words according to the security level.

Field Detail

AUTHENTICATED

`public static final byte` **AUTHENTICATED**

Entity authentication has occurred (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.*

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*

NO_SECURITY_LEVEL

`public static final byte` **NO_SECURITY_LEVEL**

Entity authentication has not occurred (0x00).

Notes:

- *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.*

Method Detail

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 instruction or only invoke this method for instructions it does not recognize.

Notes:

- *The method is responsible for receiving the data field of commands that are recognized.*
- *The applet is responsible for recognizing instructions that the method refused to process ('6E' and '6D').*
- *The applet is responsible for outputting status words 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 words may be returned):

- `ISO7816.SW_WRONG_CLA` class byte is not recognized by the method.
- `ISO7816.SW_WRONG_INS` class byte is not recognized by the method.
- `ISO7816.SW_INS_NOT_SUPPORTED` instruction 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 is used to apply additional security processing to outgoing response data and Status Words according to the security level.

Notes:

- *The applet is able to ensure that the security level it requires (`R_MAC`, `R_ENCRYPTION`) will be applied by invoking the `getSecurityLevel()` method.*
- *The `getSecurityLevel()` method invocation may also indicate that entity authentication (`AUTHENTICATION`) has previously occurred.*
- *If `NO_SECURITY_LEVEL` is indicated, this method will do no processing.*

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 wrap
`sLength` - the length of the data to wrap

Returns:

new length of wrapped data

Throws:

`javacard.framework.ISOException` - security mechanism related status words might be returned
`java.lang.ArrayIndexOutOfBoundsException` - if wrapping would cause access of data outside array bounds

---

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 security level.

Notes:

- *The applet is able to query what level of security will be assumed (`C_MAC`, `C_DECRYPTION`) to be present by the Security Domain by invoking the `getSecurityLevel()` method.*
- *The `getSecurityLevel()` method invocation may also indicate that entity authentication (`AUTHENTICATION`) has previously occurred.*

- *If `NO_SECURITY_LEVEL` is indicated, this method will do no processing.*
- *If the class byte does not indicate secure messaging (according to ISO 7816-4), this method will do no processing.*
- *The applet is responsible for receiving the data field of the command.*
- *Correct 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.*
- *Incorrect processing of the `unwrap()` will result in the information relating to the current Secure Channel being reset.*

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 wrap

Returns:

the length of the unwrapped data i.e. the length of the command data

Throws:

`javacard.framework.ISOException` - with the following reason code (other security mechanism related status words may be returned):

- `ISO7816.SW_WRONG_CLA` class byte is not recognized by the method.

---

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:

- *The Security Domain implicitly knows the key used for decryption.*
- *The Security Domain is implicitely 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.*

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` - 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
```

This method is used to encrypt data located in the input buffer.

Notes:

- *The Security Domain is implicitely 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.*

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.

---

resetSecurity

```
public void resetSecurity()
```

This method is used to reset information relating to the current Secure Channel.

Notes:

- *Applets using the services of a Security Domain, must invoke this method in the `Applet.deselect()` method.*
- *The Security Domain will reset all information relating to the current Secure Channel 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 information is present.*

---

getSecurityLevel

```
public 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.

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 e.g. an `R_MAC` session may be initiated during a `C_MAC` session*

Returns:

`NO_SECURITY_LEVEL(0x00)` indicating that entity authentication has not occurred and that the `wrap()` and `unwrap()` methods will not apply any cryptographic processing to command or response data or a bitmap of the security level as follows:

- `AUTHENTICATION` (0x80): Entity authentication has occurred.
- `C_MAC` (0x01): The `unwrap()` method will verify the MAC on an incoming command.
- `R_MAC` (0x10): The `wrap()` method will generate a MAC for outgoing response data.
- `C_DECRYPTION` (0x02): The `unwrap()` method will decrypt incoming command data.
- `R_ENCRYPTION` (0x20): The `wrap()` method will encrypt the outgoing response data.

org.globalplatform
Class GPSystem

java.lang.Object
|
+-**org.globalplatform.GPSystem**

---

public class **GPSystem** extends java.lang.Object

The System 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	
static byte	**APPLICATION_INSTALLED** The current applet context is in the Life Cycle State of INSTALLED (0x03).
static byte	**APPLICATION_SELECTABLE** The current applet context is in the Life Cycle State of SELECTABLE (0x07).
static byte	**CARD_INITIALIZED** The card is in the Life Cycle State of INITIALIZED (0x07).
static byte	**CARD_LOCKED** The card is in the Life Cycle State of CARD_LOCKED(0x7F).
static byte	**CARD_OP_READY** The card is in the Life Cycle State of OP_READY (0x01).
static byte	**CARD_SECURED** The card is in the Life Cycle State of SECURED (0x0F).
static byte	**CARD_TERMINATED** The card is in the Life Cycle State of TERMINATED (0xFF).
static byte	**CVM_GLOBAL_PIN** Indicates that the CVM interface required is a Global PIN (0x11).
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.globalplatformx.CVM	**getCVM**(byte bCVMIdentifier) This method returns a handle to the CVM interface.
static org.globalplatform.SecureChannel	**getSecureChannel**() This method returns a handle to the SecureChannel interface.
static boolean	**lockCard**() This method locks the card.
static boolean	**setATRHistBytes**(byte[] baBuffer, short sOffset, byte bLength) This method sets the historical bytes of the ATR (Answer To Reset) string.
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_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.*

SECURITY_DOMAIN_PERSONALIZED

`public static final byte` **SECURITY_DOMAIN PERSONALIZED**

The current Security Domain is in the Life Cycle State of PERSONALIZED (0x0F).

CARD_OP_READY

`public static final byte` **CARD_OP_READY**

The card is in the Life Cycle State of OP_READY (0x01).

CARD_INITIALIZED

`public static final byte` **CARD_INITIALIZED**

The card is in the Life Cycle State of INITIALIZED (0x07).

CARD_SECURED

`public static final byte` **CARD_SECURED**

The card is in the Life Cycle State of SECURED (0x0F).

CARD_LOCKED

`public static final byte` **CARD_LOCKED**

The card is in the Life Cycle State of CARD_LOCKED (0x7F).

CARD_TERMINATED

`public static final byte` **CARD_TERMINATED**

The card is in the Life Cycle State of TERMINATED (0xFF).

CVM_GLOBAL_PIN

`public static final byte` **CVM_GLOBAL_PIN**

Indicates that the CVM interface required is a Global PIN (0x11).

Method Detail

getCardContentState

`public static byte` **getCardContentState**()

This method returns the Life Cycle State of the current applet context.

Notes:

- *This method shall not be invoked from the `Applet.install()` method.*
- *The OPEN locates the entry of the current applet context in the Open Platform Registry and retrieves the Life Cycle State.*

Returns:

the Life Cycle State of the current applet context.

See Also:

`APPLICATION_INSTALLED, APPLICATION_SELECTABLE, SECURITY_DOMAIN_PERSONALIZED`

---

getCardState

`public static byte` **getCardState**()

This method 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

`public static org.globalplatformx.CVM` **getCVM**`(byte bCVMIdentifier)`

This method returns a handle to the CVM interface.

Parameters:

`bCVMIdentifier` - identifies the required CVM interface.

Returns:

the CVM interface object reference.

See Also:

`CVM_GLOBAL_PIN`

getSecureChannel

`public static org.globalplatform.SecureChannel` **getSecureChannel**()

This method returns a handle to the `SecureChannel` interface.

Notes:

- *This method shall not be invoked from the `Applet.install()` method.*
- *The OPEN locates the entry of the current applet context in the Open Platform Registry to determine the Application's associated Security Domain.*

Returns:

the `SecureChannel` interface object reference.

---

lockCard

`public static boolean` **lockCard**()

This method locks the card.

Notes:

- *This method shall not be invoked from the `Applet.install()` method.*
- *The OPEN locates the entry of the current applet context in the Open Platform Registry and verifies that the Application has the card lock privilege.*

Returns:

`true` if card locked, `false` otherwise

---

terminateCard

`public static boolean` **terminateCard**()

This method terminates the card.

Notes:

- *This method shall not be invoked from the `Applet.install()` method.*
- *The OPEN locates the entry of the current applet context in the Open Platform Registry and verifies that the Application has the card terminate privilege.*

Returns:

`true` if card terminated, `false` otherwise

---

setATRHistBytes

```
public static boolean setATRHistBytes(byte[] baBuffer,
                                       short sOffset,
                                       byte bLength)
```

This method sets the historical bytes of the ATR (Answer To Reset) string. The sequence of bytes will be visible on a subsequent power-up or reset.

Notes:

- *This method shall not be invoked from the `Applet.install()` method.*
- *The OPEN locates the entry of the current applet context in the Open Platform Registry and verifies that the Application has* the *Default Selected privilege.*
- *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 ATR historical bytes. Must be a global array.
`sOffset` - offset of the ATR historical bytes within the source byte array.
`bLength` - the number of historical bytes.

Returns:

`true` if ATR bytes set, `false` otherwise

---

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:

- *This method shall not be invoked from the `Applet.install()` method.*
- *The OPEN shall reject any transition request to the Life Cycle States INSTALLED or LOCKED.*
- *The OPEN locates the entry of the current applet context in the Open Platform 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`

---

org.globalplatformx
Interface CVM

public interface **CVM** extends javacard.framework.Shareable

This defines the interface of a privileged system class that represents a Card Holder Verification Method. 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 by invoking the GP`System.getCVM()` method.

Field Summary

static short	**CVM_FAILURE** The CVM value comparison failed (-1).
static short	**CVM_SUCCESS** The CVM value comparison was successful (0).
static byte	**FORMAT_ASCII** The CVM value is formatted as ASCII bytes (0x01).
static byte	**FORMAT_BCD** The CVM value is formatted as numerical digits, coded on a nibble (4 bits) and left justified (0x02).
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	**resetState**() This method resets the state of the CVM to active.
boolean	**resetAndUnblockState**() This method resets the state of the CVM from BLOCKED 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_SUCCESS

`public static final short` **CVM_SUCCESS**

The CVM value comparison was successful (0).

CVM_FAILURE

`public static final short` **CVM_FAILURE**

The CVM value comparison failed (-1).

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

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 (at least) ACTIVE, `false` otherwise (i.e. does not exist).

---

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 (at least) SUBMITTED, `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.

---

isBlocked

`public boolean` **isBlocked**()

This method indicates whether the CVM is currently BLOCKED.

Returns:

`true` if the CVM state is 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.

---

update

```
public boolean update(byte[] baBuffer,
                      short sOffset,
                      byte bLength,
                      byte bFormat)
```

This method changes the value of the CVM value.

Notes:

- *The OPEN locates the entry of the current applet context in the Open Platform Registry and verifies that the Application has the CVM Management privilege.*
- *The applet is responsible for identifying the format of the CVM value.*
- *The CVM try counter is reset when changing the CVM value.*

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.

---

resetSate

`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 BLOCKED.*

Returns:

`true` if the CVM state was reset, `false` otherwise.

---

blockState

`public boolean` **blockState**()

This method sets the state of the CVM to BLOCKED.

Note:

- *The OPEN locates the entry of the current applet context in the Open Platform Registry and verifies that the application has the CVM Management privilege.*

Returns:

`true` if the CVM state was set to BLOCKED, `false` otherwise.

---

resetAndUnblockState

`public boolean` **resetAndUnblockState**()

This method resets the state of the CVM from BLOCKED to ACTIVE.

Notes:

- *The OPEN locates the entry of the current applet context in the Open Platform Registry and verifies that the application has the CVM Management privilege.*
- *The CVM try counter is reset when unblocking the CVM.*

Returns:

`true` if the CVM state was reset to ACTIVE, `false` otherwise.

---

setTryLimit

`public boolean` **setTryLimit**`(byte bTryLimit)`

This method sets the Retry Limit for the CVM.

Notes:

- *The OPEN locates the entry of the current applet context in the Open Platform Registry and verifies that the application has the CVM Management privilege.*
- *The CVM Retry Counter is reset when setting the maximum number of tries.*

Parameters:

`bTryLimit` - the Retry Limit for the CVM.

Returns:

`true` if the Retry Limit was set, `false` otherwise.

---

verify

```
public short verify(byte[] baBuffer,
                    short sOffset,
                    byte bLength,
                    byte bFormat)
```

This method compares the CVM with the 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 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.*
- *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 within source byte array
`bLength` - the length of the CVM
`bFormat` - the format of the CVM

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.

A.3 Open Platform on Windows Powered Smart Card

Application API Interface on Windows Powered Smart Card

The OPEN provides an API interface that Applications may use to get services. The OPEN retains the control of the services and determines whether or not to grant each request.

The following APIs are provided to access OPEN services. They are shown in the standard Visual Basic™ function prototype format.

OpGetCardContentState

Function OpGetCardContentState (ByRef AppState as Byte) as Byte

This API is used to retrieve the Life Cycle State of the currently selected Application. The OPEN locates the AID of the currently selected Application in the Open Platform Registry, and returns the Life Cycle State

In addition to application specific values, the following predefined values may be returned by this function:

- APPLICATION_INSTALLED = &H3
- APPLICATION_SELECTABLE = &H7
- SECURITY_DOMAIN_PERSONALIZED = &HF

OpSetCardContentState

Function OpSetCardContentState (ByVal state as Byte) as Byte

This API allows the currently selected Application to set its own Life Cycle State.

An Application may use this function in order to change its own Life Cycle State to an application specific value.

OpSetCardState

Function OpSetCardState (ByVal state as Byte) as Byte

This API changes the state of the card to the state specified.

The states may be:

- CARD_LOCKED = &H7F
- CARD_TERMINATED = &HFF

OPEN locates the registry entry of the currently selected Application and verifies that the application has the privilege to change the card state to the specified one.

OpGetCardState

Function OpGetCardState (ByRef CardState as Byte) as Byte

This API returns the Life Cycle State of the card

One of the following four states will be returned by this function:

- CARD_OP_READY = &H1
- CARD_INITIALIZED = &H7

- CARD_SECURED = &HF
- CARD_LOCKED = &H7F

OpSetATRHistBytes

Function OpSetATRHistBytes (ATRData() as Byte, ByVal offset as Byte, ByVal length as Byte) as Byte

Paramenters:

- ATRData() – Buffer byte array containing the ATR historical bytes.
- offset – byte: Offset within the buffer where ATR historical bytes begin.
- length – byte Length of the ATR historical bytes in the buffer.

This function sets the Historical Bytes contained in the ATR. The sequence of bytes will be valid within an ATR on a subsequent reset. This service is granted only if the currently selected Application has the Default Selected privilege.

An Application may call this function in order to set the value of the Historical Bytes of the ATR. This functionality is only accessible to the implicitly selectable Application. The Application sets the value of up to 15 Historical Bytes as well as the number of Historical Bytes within the ATR. This function may be invoked at any time in the life of the Application subsequent to the Application being set to a selectable state with the Default Selected privilege.

Security Domain API Interface on Windows Powered Smart Card

The following sections describe the API available for an application to obtain services from its associated security domain. There will be mechanism whereby CLA, INS, P1, P2, and LC can be passed back and forth between an application and its security domain. In addition, the security domain will be able to pass back a 2 two-byte ResultCode and a 1 byte CurrentSecurityLevel byte on each call. The CurrentSecurityLevel byte will be encoded as follows:

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

Table A- 2: Open Platform on Windows Powered Smart Card: Security Level

OpProcessSecureChannel

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 applet 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.

Note:

- The Security Domain will retrieve the data portion of the APDU command from the card input communication buffer, and will place any command response data in the card output communication buffer.
- The application will be responsible to output the ResultCode that was set by the Security Domain.

OpResetSecurity

This function is used to erase any secure information relating to the current secure session that may have been established.

OpDecryptData

This function is used to decrypt data that an application has placed in the card communication buffer.

Note:

- The Security Domain will retrieve the data to be decrypted from the card input communication buffer and place the results in the card output communication buffer. The decrypted data will then be retrieved by the calling application.

OpUnwrap

This function is used to process and verify the secure messaging of an incoming command.

Note:

- If the class byte indicate secures messaging (ISO 7816-4), the Security Domain will retrieve the data portion of the APDU command from the input communication buffer and will place the reformatted APDU command data in the output communication buffer. The Security Domain will have reformatted the APDU to have all data relating to the Secure Messaging removed.
- The Application is responsible for checking the ResultCode set by the Security Domain to verify that the Security Domain was able to process the APDU according to the requirements for integrity and confidentiality that were specified when the secure channel was established.
- If the Security Domain was unable to correctly process the APDU command, the Security Domain will have reset all information relating to the current secure session.

OpWrap

This function is used to apply additional security processing to outgoing response data and Status Words.

Note:

- The application must place the data to be wrapped into the card output communication buffer.

OpEncryptData

This function is used to encrypt data that an application has placed in the card communication buffer.

Note:

- The Security Domain will retrieve the data to be encrypted from the card input communication buffer and place the results in the card output communication buffer. The encrypted data will then be retrieved by the calling application.

B. Algorithms (Cryptographic and Hashing)

An Open Platform card may support many types of security functions for use by Applications. This section contains examples of several of the cryptographic algorithms and hashing method that are possible for Open Platform.

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. DES and triple DES are defined in FIPS PUB 46-3. Triple DES as used in Open Platform uses keying option 2 as defined in FIPS PUB 46-3.

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 ‘00h 00h 00h 00h 00h 00h 00h 00h’.

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 [IS9797].

B.1.2.1 Full Triple DES MAC

The Full Triple DES MAC is as defined in [ISO 9797-1] as MAC Algorithm 1 with output transformation 1, 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 1 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.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 Open Platform 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 length of the Key Modulus is 1024 bits for this Specification.

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, 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.

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 section defines the different methods that shall be used to secure the changes and/or modifications to the content of an Open Platform card. At this point in time, these methods are the only ones specified for Open Platform cards. Depending on the implementation of the card, any subset of these methods may be supported.

C.1 Keys

In order to perform the Content Management operations described in the subsequent sub-sections different keys are required.

C.1.1 Issuer Security Domain Keys

The Issuer Security Domain 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	Length	Remark
Token	Token verification (RSA Public Key)	1024 bits	Delegated Management only
Receipt	Optional Receipt Generation (DES)	16 bytes	Delegated Management only

Table C-1: Issuer Security Domain Keys

C.1.1.1 Token key

If the card supports Delegated Management, the Issuer Security Domain shall have a 1024 bit RSA public key to be used to verify a Load, Install, or Extradition Token.

C.1.1.2 Receipt Key

If the card supports Delegated Management and specifically Receipt generation, the Issuer Security Domain shall have an unambiguously recognized double length DES key to be used to generate Load, Install, Extradition, or Delete Receipts.

C.1.2 Security Domain Keys

If the Security Domain supports DAP Verification, the Security Domain shall have either a DAP Verification Public Key or a DAP Verification DES Key that will be used to verify Load File Data Block Signatures.

Key	Usage	Length	Remark
DAP Verification	Load File Data Block Signature verification (RSA Public Key)	1024 bits	DAP Verification only
DAP Verification	Load File Data Block Signature verification (DES Key)	16 Bytes	DAP Verification only

Table C- 2: Additional Security Domain Key

C.2 Load File Data Block Hash

The Load File Data Block Hash is a field that is conditionally present in the INSTALL [for load] command. The purpose of this field is to provide a 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.

This hash is required in the command if Delegated Management loading is occurring and/or the Load File contains one or more DAP blocks:

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.

The hash present in the INSTALL [for load] command is generated according to Section 6.7.6.1 - *Load File Data Block Hash.*

In order for the chain of trust to be complete, the card is responsible for verifying the hash on receipt of the complete Load File Data Block.

A Security Domain's owner (e.g. a Controlling Authority or an Application Provider) generates a SHA-1 digest across the complete Load File Data Block. 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).

C.3 Tokens

Tokens are used when Delegated Management is being performed and they are Public Key signatures of the token data defined in Section 7.7 - *Delegated Management Tokens and Receipts and DAP Verification.* Tokens are the proof that the Card Issuer has authorized the Delegated Management operation being performed and they are generated and verified according to this Appendix.

All cards supporting Delegated Management shall perform token verification and the Issuer Security Domain shall have knowledge of a token verification public key to perform this verification. The signature scheme for tokens is as defined in Appendix B.3, Public Key Cryptography Scheme 1.

C.3.1 Load Token

A Card Issuer generates a Load Token. This is a signature authorizing the transmission of application code to the card.

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; and,
- 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 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)	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
Load Parameters field	Var

Table C- 3: Data Elements Included in the Load Token

This RSA signature is the encryption (with the token private key) of the SHA-1 message digest of the above data. Padding of the data is as defined by the SHA-1 and PKCS#1 mechanisms.

The OPEN receives each of the Delegated Management load process commands (INSTALL [for load] command followed by multiple LOAD commands) from the Security Domain. The Issuer Security Domain (at the request of the OPEN) is responsible for verifying the Load Token (signature) using the token verification public key. The OPEN is responsible for ensuring that the hash (Load File Data Block Hash) of the Load File Data Block present in the INSTALL [for load], command is a valid SHA-1 message digest of the Load File Data Block.

C.3.2 Install Token

A Card Issuer generates an Install Token. This is a signature authorizing the installation to the card of an Application (Executable Module) contained in a previously loaded file (Executable Load File).

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; and,
- Only the parameters included in this signature may be used to install the Application.

The Install Token is an RSA signature of the following data that will be included in an INSTALL [for install] command

Note: An INSTALL [for install] command may also include the option to make the Application selectable.

Name	Length
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	5-16
Executable Module AID length	1
AID within the executable Load File	5-16
Instance AID length	1
Instance AID	5-16
Application privileges length	1
Application privileges	1
Install Parameters field length	1
Install Parameters (system and Application) field	Var

Table C- 4: Data Elements Included in the Install Token

This RSA signature is the encryption (with the token private key) of the SHA-1 message digest of the above data. Padding of the data is as defined by the SHA-1 and PKCS#1 mechanisms.

The Security Domain passes the Delegated Management command (INSTALL [for install] command) to the OPEN. The Issuer Security Domain is responsible for verifying the Install Token using the Issuer Security Domain's token verification public key.

C.3.3 Extradition Token

A Card Issuer generates an Extradition Token. This is a signature authorizing the extradition of an Application from one Security Domain to another.

Generating an Extradition Token ensures the following:

- The instance of the Application may only be associated with Security Domain included in the signature;

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
Reference Control Parameter P1	1
Reference Control Parameter P2	1
Length of the following data fields (including the length fields)	1
Security Domain AID length	1
Security Domain AID	5-16
Length = 0	1
Instance AID length	1
Instance AID	5-16

Length = 0	1
Length = 0	1

Table C- 5: Data Elements Included in the Extradition Token

This RSA signature is the encryption (with the token private key) of the SHA-1 message digest of the above data. Padding of the data is as defined by the SHA-1 and PKCS#1 mechanisms.

The Security Domain passes the Delegated Management command (INSTALL [for extradition] command) to the OPEN. The Issuer Security Domain is responsible for verifying the Extradition Token using the Issuer Security Domain's token verification public key.

C.4 Receipts

Receipts are also optionally used when Delegated Management is being performed and are DES signatures of the receipt data defined in Section 7.7 - *Delegated Management Tokens and Receipts and DAP Verification.* Receipts, if required by the Card Issuers Security Principles, are intended to be confirmation for the Card Issuer that a Delegated Management operation has been successfully completed. They are generated according to this Appendix.

The card generates receipts during Delegated Management operations as proof to the Card Issuer that the operation (load, install, extradition or deletion) was successfully performed.

Cards supporting Delegated Management may generate receipts and in order to achieve this the Issuer Security Domain would require knowledge of a double length DES receipt key.

Each receipt is generated using the receipt key, an ICV of binary zeroes and the signature method described in Appendix B.1.2.1 - *Full Triple DES.*

In addition to the receipt key, the Issuer Security Domain also keeps track of a Confirmation Counter. The confirmation number is a 16-bit value that is incremented by one (1) following each receipt generation. The Confirmation Counter is initialized to zero.

C.4.1 Load Receipt

The Load Receipt, if used, provides proof to the Card Issuer that the identified Security Domain performed a load of application code to a specific card.

A Load Receipt is a triple DES signature of the following data:

Name	Length
Confirmation Counter length	1
Confirmation Counter	2
Card Unique data length	1
Card Unique data	10
Executable Load File AID length	1
Executable Load File AID	5-16
Security Domain AID length	1
Security Domain AID	5-16

Table C- 6: Data Elements Included in the Load Receipt

Prior to generating the signature, the data shall be padded according to B.4- *DES Padding*

The signature method, defined in Appendix B.1.2.1 - *Full Triple DES*, is applied across the padded data and the signature along with other required data may be returned as the response message to the last LOAD command.

The entity performing the Delegated Management function will forward this information to the Card Issuer. The Card Issuer verifies the Load Receipt using the same above procedure with the additional comparison step.

C.4.2 Install Receipt

The Install Receipt, if used, provides proof to the Card Issuer that the identified Security Domain performed the installation of an Application on a specific card.

An Install Receipt is a triple DES signature of the following data:

Name	Length
Confirmation Counter length indicator	1
Confirmation Counter	2
Card Unique data length indicator	1
Card Unique data	10
Executable Load File AID length	1
Executable Load File AID	5-16
Instance AID length indicator	1
Instance AID	5-16

Table C- 7: Data Elements Included in the Install Receipt

Prior to generating the signature the data shall be padded according to B.4- *DES Padding*

The signature method, defined in Appendix B.1.2.1 - *Full Triple DES*, is applied across the padded data and the signature along with other required data may be returned as the response message to the INSTALL [for install] command.

The entity performing the Delegated Management function will forward this information to the Card Issuer. The Card Issuer verifies the Install Receipt using the same above procedure with the additional comparison step.

C.4.3 Delete Receipt

The Delete Receipt, if used, provides proof to the Card Issuer that the identified Security Domain deleted an Application or Executable Load File from a specific card.

A Delete Receipt is a triple DES signature of the following data:

Name	Length
Confirmation Counter length indicator	1
Confirmation Counter	2
Card Unique data length indicator	1
Card Unique data	10
AID length indicator	1
Application instance or Executable Load file AID	5-16

Table C- 8: Data Elements Included in the Delete Receipt

Prior to generating the signature the data shall be padded according to B.4 - *DES Padding.*

The signature method, defined in Appendix B.1.2.1 - *Full Triple DES*, is applied across the padded data and the signature along with other required data may be returned as the response message to the DELETE command.

The entity performing the Delegated Management function will forward this information to the Card Issuer. The Card Issuer verifies the Delete Receipt using the same above procedure with the additional comparison step.

C.4.4 Extradition Receipt

The Extradition Receipt, if used, provides proof to the Card Issuer that the identified old Security Domain extradited an Application to the new Security Domain on a specific card.

An Extradition Receipt is a DES signature of the following data:

Name	Length
Confirmation Counter length indicator	1
Confirmation Counter	2
Card Unique data length indicator	1
Card Unique data	10
AID length indicator	1
Old Security Domain AID	5-16
AID length indicator	1
Application instance or Executable Load file AID	5-16
AID length indicator	1
New Security Domain AID	5-16

Table C- 9: Data Elements Included in the Extradition Receipt

Prior to generating the signature the data shall be padded according to B.4 - *DES Padding.*

The signature method, defined in Appendix B.1.2.1 - *Full Triple DES*, is applied across the padded data and the signature along with other required data may be returned as the response message to the INSTALL [for extradition] command.

The entity performing the Delegated Management function will forward this information to the Card Issuer. The Card Issuer verifies the Extradition Receipt using the same above procedure with the additional comparison step

C.5 DAP Verification

Load File Data Block Signature Verification is typically used by an entity other than the Card Issuer or the entity responsible for loading the Application to the card, 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 a 1024 bit public key or a 16 byte DES key.

Exactly how the OPEN and Security Domain interface during the DAP Verification process is beyond the scope of this Specification. However the verification operation mandates that the signature be verified using the Security Domain's DAP Verification key.

C.5.1 PKC Scheme

An entity generates a signature of the Load File Data Block Hash as defined in Appendix B.3 - *Public Key Cryptography Scheme 1 (PKCS#1)*, i.e. computing a SHA-1 message digest of the Load File Data Block Hash and apply the DAP Verification private key to obtain the signature. This RSA signature is the encryption 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 the SHA-1 and PKCS#1. The signature is provided to any entity responsible for loading the signed Application to an Open Platform card. The AID of the Security Domain representing the Controlling Authority along with this 128-byte signature are placed in the Load File as a DAP Block to be transmitted to the card.

C.5.2 DES Scheme

A Security Domain's owner generates a signature of the Load File Data Block Hash. This signature is the MACing 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 to an Open Platform card. The AID of the Security Domain representing the Security Domain's owner along with this signature are placed in the Load File as a DAP Block to be transmitted to the card.

D. Secure Channel Protocol '01'

The following section defines the usage of Secure Channel Protocol 01 (SCP01).

D.1 Secure Communication

D.1.1 SCP01 Secure Channel

The 3 levels of security provided by the SCP 01 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; and,

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.

Only explicit Secure Channel initiation is supported in SCP 01. Both implicit and explicit Secure Channel termination are supported in SCP 01.

In SCP01 the card shall support the following implementation option defined by "i" (see Appendix F - *GlobalPlatform Data Values and Card Recognition Data*):

- "i" = '50': Initiation Mode explicit, C-MAC on modified APDU, ICV set to zero, 3 Secure Channel Keys.

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 an Open Platform Java Card.

The explicit Secure Channel initiation allows the off-card entity to instruct the card (see Appendix D.4.2 - *EXTERNAL AUTHENTICATE Command-Response APDU*) 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 Set and Key Index 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-Response APDU*) 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-Response APDU*).

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.

SCP 01 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.

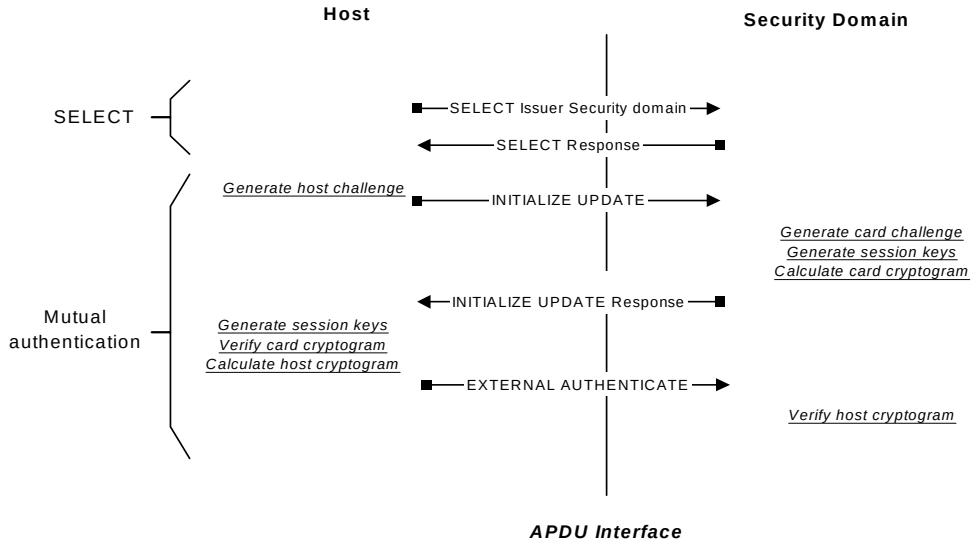


Figure D- 1: SCP 01 Mutual Authentication Flow (Security Domain)

Expanding the authentication process shown in the flow described in Section 7.4 - *Runtime Messaging Support*, it can be seen how an Application would use the services of an associated Security Domain to achieve mutual authentication.

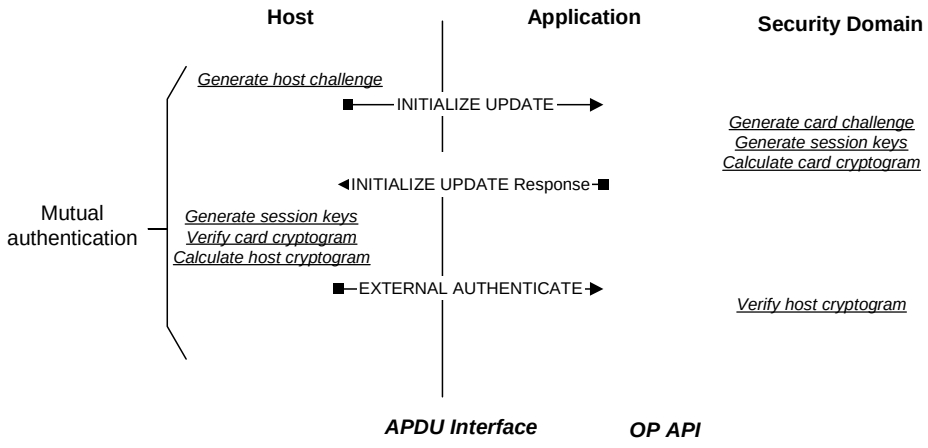


Figure D- 2: SCP 01 Mutual Authentication Flow (using services of Security Domain)

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 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.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 key 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 and Hashing 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 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; and,
- Creating the Secure Channel MAC session key.

The DES operation used to generate these keys is always triple DES in ECB mode.

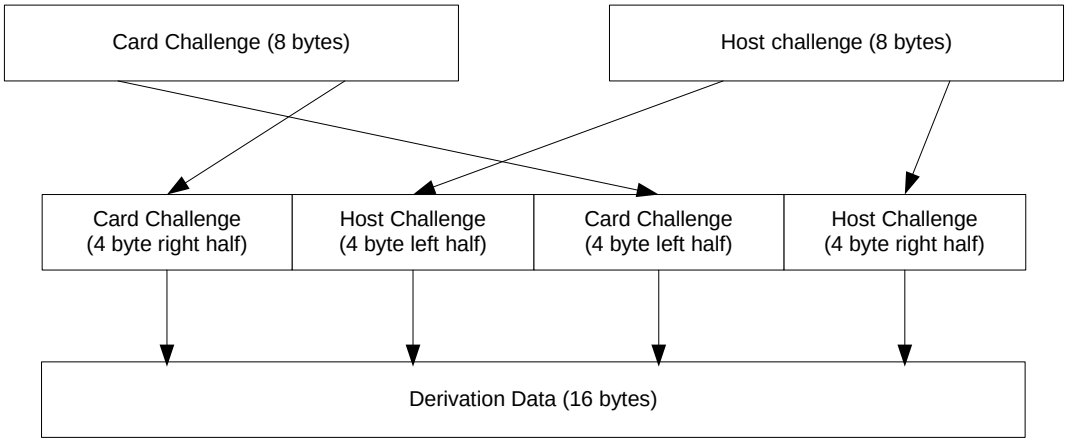


Figure D- 3: SCP01 Session Key - Step 1 - Generate Derivation data

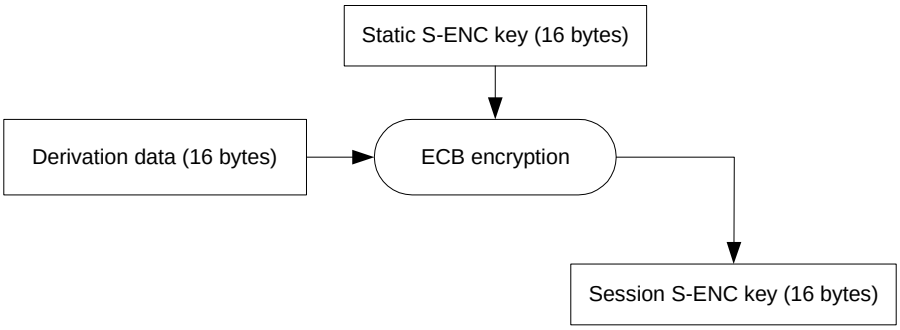


Figure D- 4: SCP01 Session Key - Step 2 - Create S-ENC Session Key

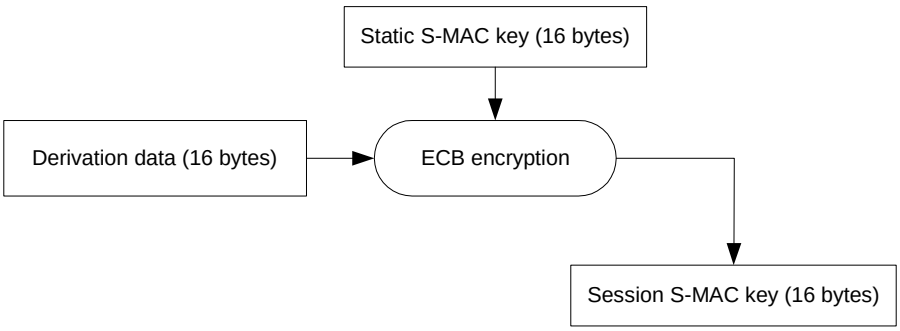


Figure D- 5: SCP01 Session Key - Step3 - Create S-MAC Session Key

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 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 includes secure messaging. This is achieved by setting bit 3 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 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.

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 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).

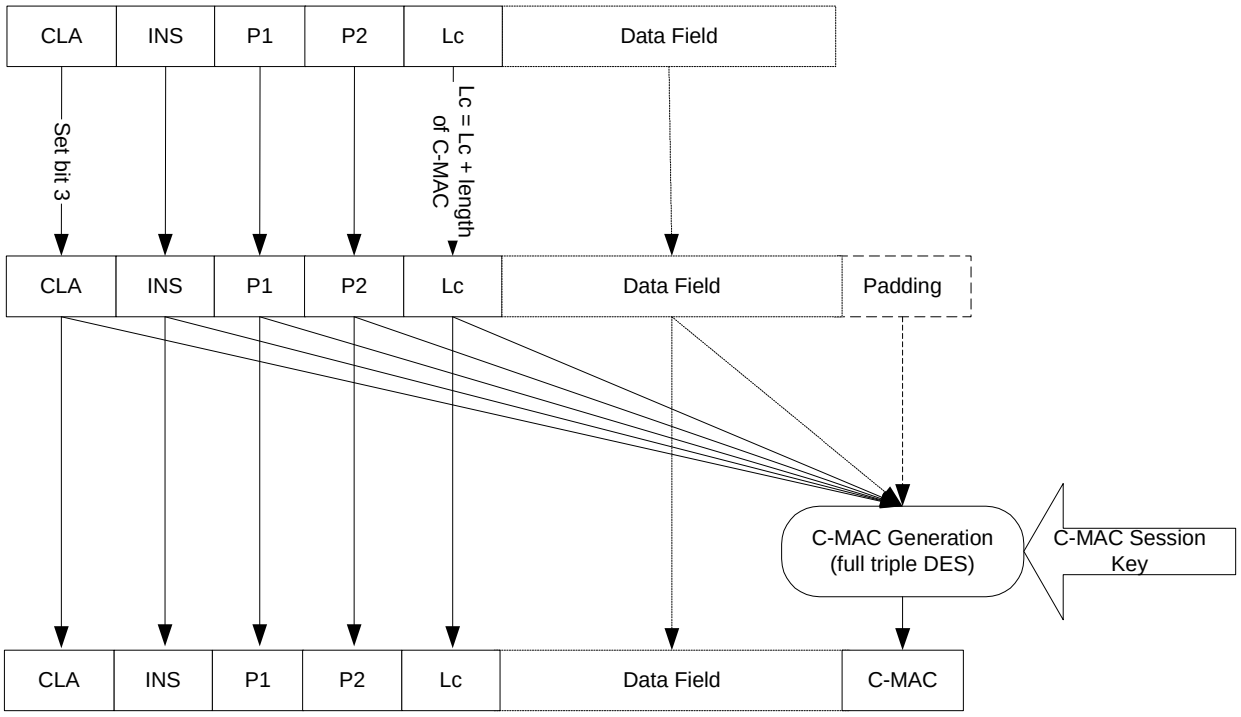


Figure D- 6: SCP01 - APDU Command MAC Generation and Verification

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).

D.3.4 APDU Data Field Encryption and Decryption

Depending on the 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 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.1.2 *MACing*

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.

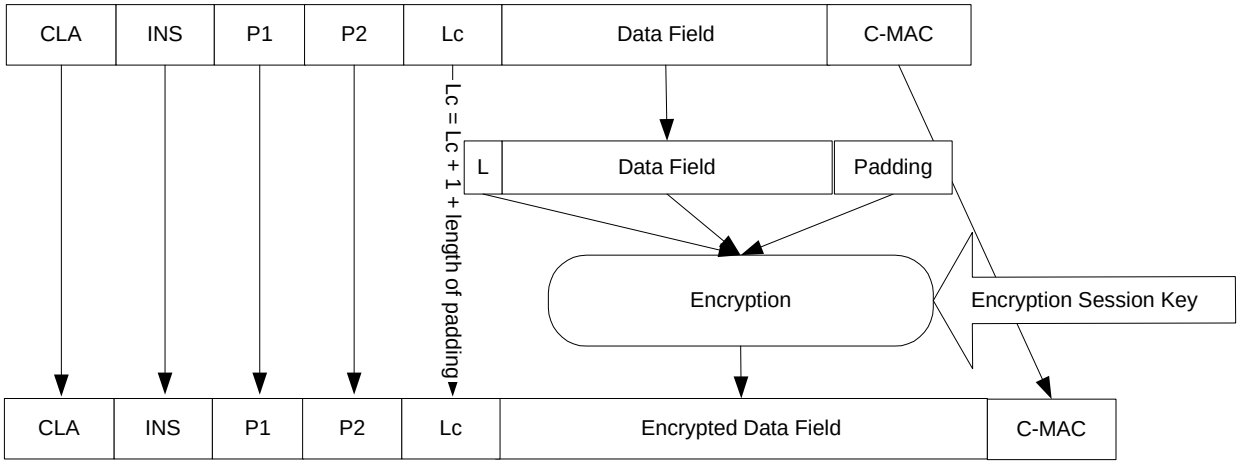


Figure D- 7: SCP01 - APDU Data Field Encryption

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- 2provides the list of Minimum Security Requirements for SCP 01 Commands

Command	Minimum Security Level
INITIALIZE UPDATE	None
EXTERNAL AUTHENTICATE	C-MAC

Table D- 2: Minimum Security Requirements for SCP 01 Commands

Table D-3 provides the list of SCP 01 Command support per Card Life Cycle State.

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED			TERMINATED		
	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD
INITIALIZE UPDATE	✓	✓		✓	✓		✓	✓		✓					
EXTERNAL AUTHENTICATE	✓	✓		✓	✓		✓	✓		✓					

Table D- 3: SCP 01 Command Support per Card Life Cycle State

Legend of Table D-3:

ISD: Issuer Security Domain.

DM SD: Application Provider Security Domain with Delegated Management Privilege.

SD: Application Provider Security Domain without Delegated Management Privilege.

✓ : Support required.

Blank cell: Support optional.

Striped cell: Support prohibited.

D.4.1 INITIALIZE UPDATE Command-Response APDU

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'	
INS	'50'	INITIALIZE UPDATE
P1	'xx'	Key Version Number
P2	'xx' '00'	Reference Control Parameter P2
Lc	'08'	Length of data
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 Index

This Key Index identifies a Key Index with the Key Version Number defined in Reference Control Parameter P1. If this value is zero the first key with the Key Version Number identified by P1 shall be used.

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 Secure Channel 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 data	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 data includes the Key Set 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

A successful execution of the command shall be coded by '90' '00'.

This command may either return a general error condition as listed in Section 9.1.3 - *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-Response APDU

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'	Open Platform command with Secure Messaging
INS	'82'	EXTERNAL AUTHENTICATE
P1	'xx'	Security level
P2	'00'	Reference Control Parameter P2
Lc	'10'	Length of data
Data	'xx xx…'	Host cryptogram and MAC
Le	'00'	

Table D- 7: EXTERNAL AUTHENTICATE Command Message

D.4.2.3 Reference Control Parameter P1 - Security Level

This parameter 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	Command encryption 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

Reference Control Parameter P2 is coded 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.

A successful execution of the command shall be coded by ‘90’ ‘00’.

This command may either return a general error condition as listed in Section 9.1.3 - *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’

The following section defines the usage of Secure Channel Protocol 02 (SCP02).

E.1 Secure Communication

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; and,

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 card shall support at least one of the following implementation options as defined by "i" (see Appendix F - *GlobalPlatform Data Values and Card Recognition Data*):

- "i" = '40': Initiation Mode explicit, C-MAC on modified APDU, ICV set to zero, 1 Secure Channel Base Key,
- "i" = '50': Initiation Mode explicit, C-MAC on modified APDU, ICV set to zero, 3 Secure Channel Keys,
- "i" = 'A0': Initiation Mode implicit, C-MAC on unmodified APDU, ICV set to MAC over AID, 1 Secure Channel Base Key,
- "i" = 'B0': Initiation Mode implicit, C-MAC on unmodified APDU, ICV set to MAC over AID, 3 Secure Channel Keys.

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)

Note: the Sequence Counter is not updated for C-MACs that are 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.

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 an Open Platform Java Card.

The explicit Secure Channel initiation allows the off-card entity to instruct the card (Appendix E.5.2-*EXTERNAL AUTHENTICATE Command-Response APDU*) 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 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 E.5.1 - *INITIALIZE UPDATE Command-Response APDU*) 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 (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-Response APDU).*

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-MACs and/or R-MACs.

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 7.4 - *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.

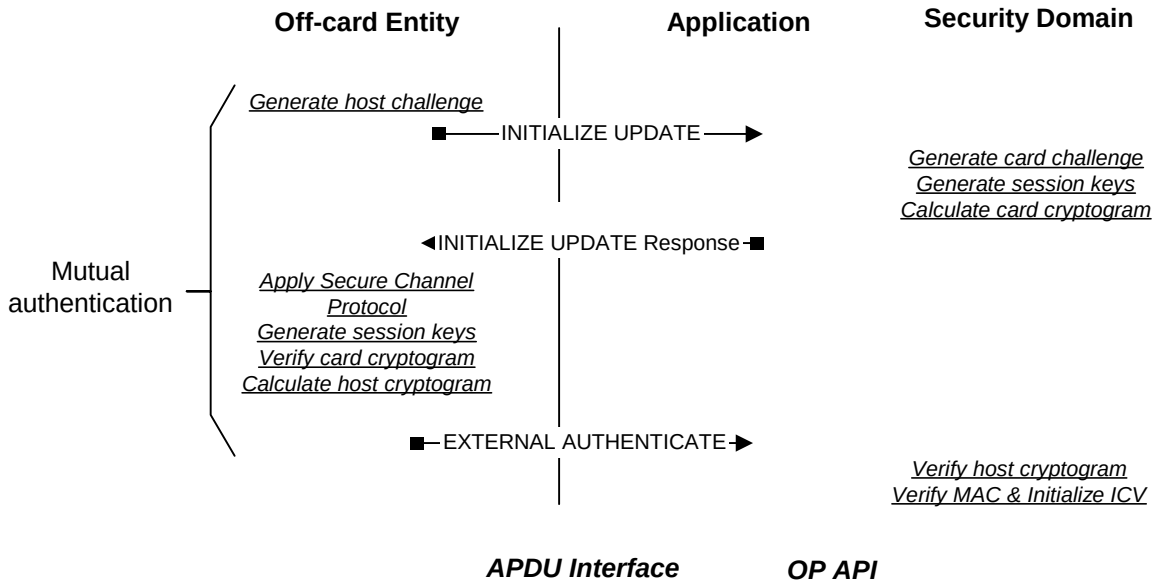


Figure E- 1: SCP02 - Explicit Secure Channel Initiation Flow

E.1.2.2 Implicit Secure Channel Initiation

The Secure Channel is implicitly initiated when receiving the first command-APDU 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 Session Keys 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.

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 and END R-MAC SESSION commands

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 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 original ICV is detailed in Appendix B.1.2.2 - Single DES Plus Final Triple DES.

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.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- 1: 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
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- 2: 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 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.4 Cryptographic and Hashing 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 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 Key using the Secure Channel Base Key or MAC Key (S-MAC) and the Session Key derivation data with a constant of '0101',
- Generating the Secure Channel R-MAC Session Key using the Secure Channel Base Key or MAC Key (S-MAC) and the Session Key derivation data with a constant of '0102',
- Generating the Secure Channel Encryption Session Key using the Secure Channel Base Key or Encryption Key (S-ENC) and the Session Key derivation data with a constant of '0182',
- Generating the Secure Channel Data Encryption Session Key using the Secure Channel Base Key or Data Encryption Key (DEK) and the Session Key derivation data with a constant of '0181'.

The DES operation used to generate these keys is always triple DES in CBC mode.

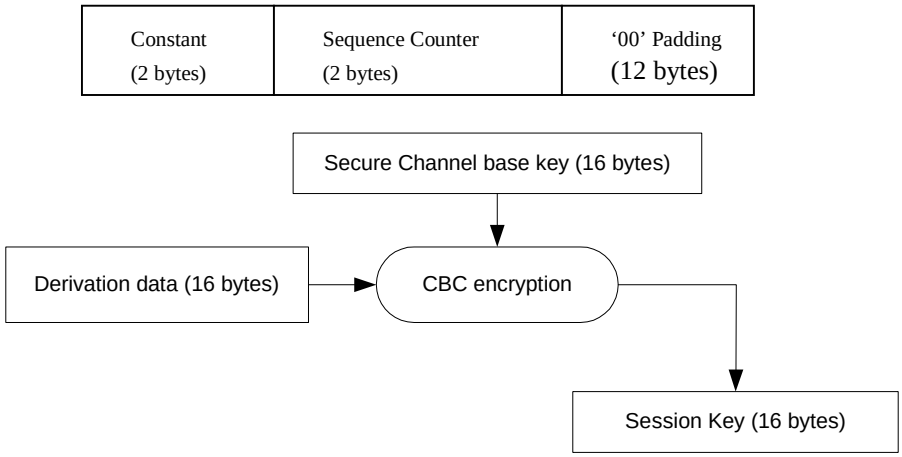


Figure E- 2: SCP02 - Create Secure Channel Session Key from the Base Key

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 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.

E.4.2.2 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.

E.4.3 Authentication Cryptogram in Implicit Secure Channel Initiation

When using the Implicit Secure Channel 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*.

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 includes Secure Messaging. This is achieved by setting bit 3 of the Class byte. For all the commands defined in this specification, the class byte of commands that contain Secure Messaging will be ‘84’.

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.

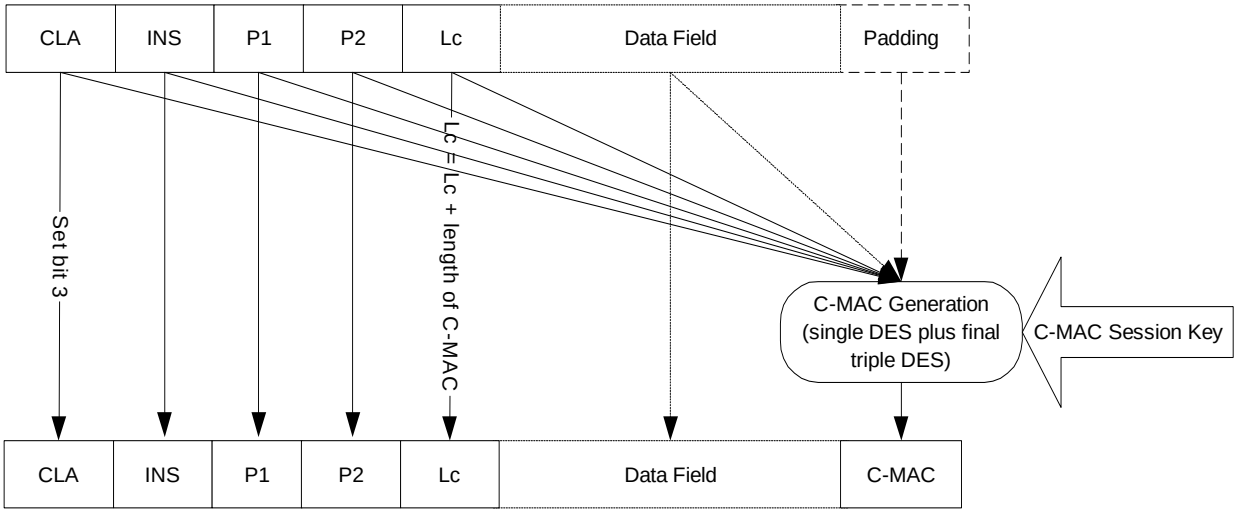


Figure E- 3: SCP02 – C-MAC Generation on Unmodified APDU

- 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.

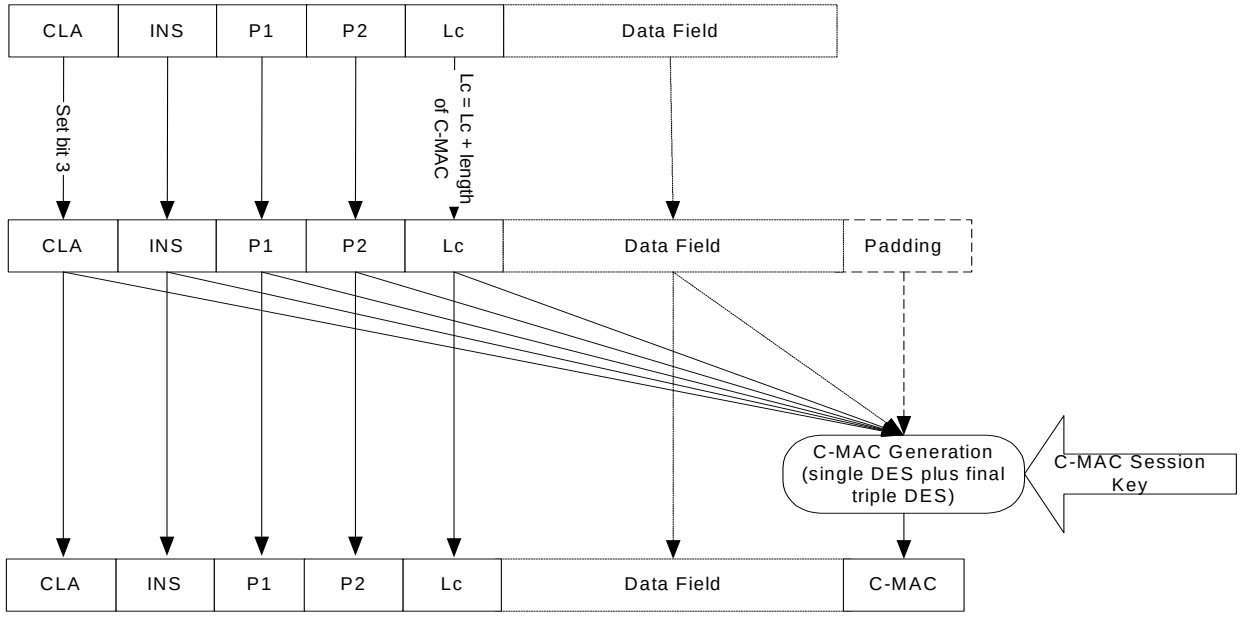


Figure E- 4: SCP02 – C-MAC Generation on Modified APDU

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.

E.4.5 APDU Response R-MAC Generation and Verification

When using Explicit Secure Channel 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 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-Response APDU*). 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-Response APDU*).

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 response data preceded with a byte that codes its length
- the status words

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 C-MAC. The rules for R-MAC padding are as defined in Appendix B.4 - *DES Padding.*

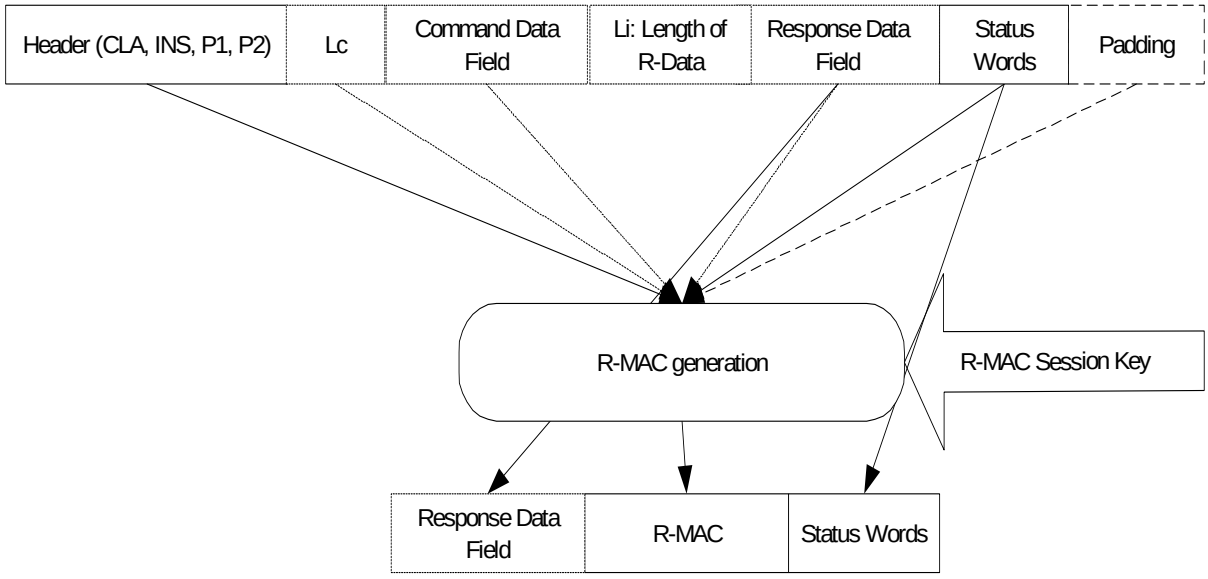


Figure E- 5: SCP02 – R-MAC Generation

The R-MAC can be retrieved from the card by sending an END R-MAC SESSION command. This ends the R-MAC session.

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 completed successfully or not.)

E.4.6 APDU Command Data Field Encryption and Decryption

Depending on the 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.1.2 *MACing*.

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.

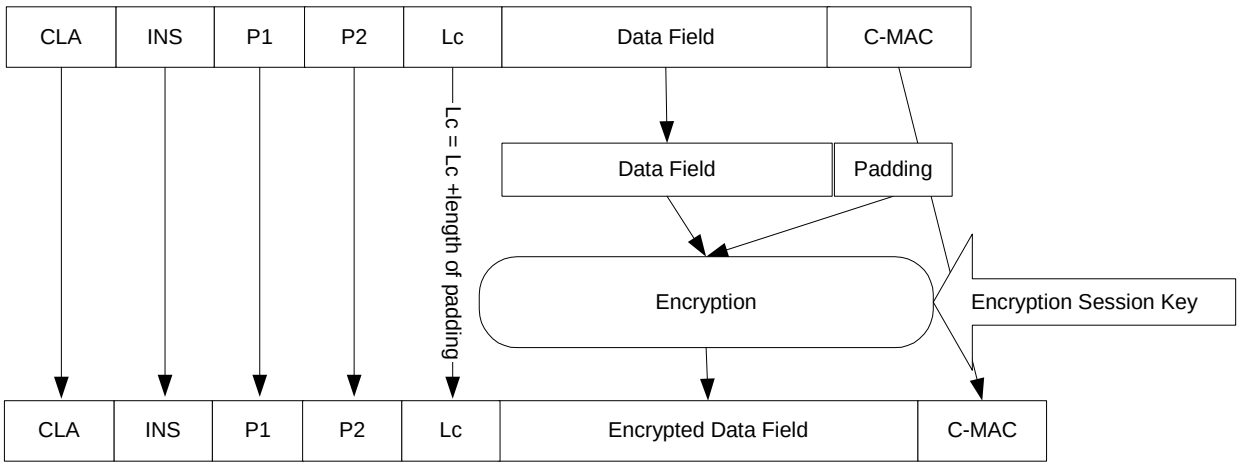


Figure E- 6: SCP02 - APDU Command Data Field Encryption

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 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 static Data Encryption Session Key and the encryption method described in Appendix -B.1.2 *MACing*.

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	✓	
HANDLE R-MAC SESSION		
END R-MAC SESSION		

Table E- 3: SCP 02 Command Support

Legend of Table E-3:

✓ : Support required.

Blank cell: Support optional.

Command	Minimum Security Level	
	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- 4: Minimum Security Requirements for SCP 02 Commands

Note that Table E-3 shall be used in conjunction with Table E-5 to determine SCP 02 command support requirements.

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED			TERMINATED		
	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD	ISD	DM SD	SD
INITIALIZE UPDATE	✓	✓		✓	✓		✓	✓		✓					
EXTERNAL AUTHENTICATE	✓	✓		✓	✓		✓	✓		✓					
BEGIN R-MAC SESSION															
END R-MAC SESSION															

Table E- 5: SCP 02 Command Support per Card Life Cycle State

Legend of Table E-5*:*

ISD: Issuer Security Domain.

DM SD: Application Provider Security Domain with Delegated Management Privilege.

SD: Application Provider Security Domain without Delegated Management Privilege.

✓ : Support required.

Blank cell: Support optional.

Striped cell: Support prohibited.

E.5.1 INITIALIZE UPDATE Command-Response APDU

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'	
INS	'50'	INITIALIZE UPDATE
P1	'xx'	Key Version Number
P2	'00'	Reference Control Parameter P2
Lc	'08'	Length of data
Data	'xx xx…'	Host challenge
Le	'00'	

Table E- 6: SCP02 - INITIALIZE UPDATE Command Message

E.5.1.3 Reference Control Parameter P1 - Key Set Version

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 shall 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 data	2 bytes
Sequence counter	2 bytes
Card Challenge	6 bytes
Card cryptogram	8 bytes

Table E- 7: SCP02 – 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 data includes the Key Set Version and the Secure Channel Protocol Identifier, here '02', used in initiating the Secure Channel session.

The card 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

A successful execution of the command shall be coded '9000'.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or one of the following error conditions:

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table E- 8: SCP02 – INITIALIZE UPDATE Error Condition

E.5.2 EXTERNAL AUTHENTICATE Command-Response APDU

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'	
INS	'82'	EXTERNAL AUTHENTICATE
P1	'xx'	Security level
P2	'00'	Reference Control Parameter P2
Lc	'10'	Length of data
Data	'xx xx…'	Host cryptogram and MAC
Le	'00'	

Table E- 9: SCP02 – EXTERNAL AUTHENTICATE Command Message

E.5.2.3 Reference Control Parameter P1 - Security Level

This parameter 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	Command Encryption, 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	Command Encryption 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- 10: EXTERNAL AUTHENTICATE Parameter P1

E.5.2.4 Reference Control Parameter P2

Reference Control Parameter P2 shall be coded '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 APDU command 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

A successful execution of the command shall be coded ‘9000’.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or the following WARNING condition:

SW1	SW2	Meaning
‘63’	‘00’	Authentication of Host Cryptogram failed

Table E- 11: SCP02 – EXTERNAL AUTHENTICATE warning Code

E.5.3 BEGIN R-MAC SESSION Command-Response APDU

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’ or ‘84’	Open Platform command
INS	‘7A’	BEGIN R-MAC SESSION
P1	‘xx’	Reference Control Parameter P1
P2	‘xx’	Reference Control Parameter P2
Lc	‘xx’	Length of data, if any
Data	‘xx xx…’	BEGIN R-MAC SESSION data and C-MAC, if needed
Le	‘00’	

Table E- 12: BEGIN R-MAC SESSION Command Message

E.5.3.3 Reference Control Parameter P1

This parameter 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- 13: BEGIN R-MAC SESSION 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 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:

Presence	Length in Bytes	Name
Mandatory	1	Length of data
Conditional	0-24	data

Table E- 14: BEGIN R-MAC SESSION Command Data Field

E.5.3.5 Return Processing State

A successful execution of the command is coded by ‘90’ ‘00’.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or the following error condition:

SW1	SW2	Meaning
‘6A’	‘88’	Referenced data not found

Table E- 15: BEGIN R-MAC SESSION Error Conditions

E.5.4 END R-MAC SESSION Command-Response APDU

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. At any time during an R-MAC session, the END R-MAC SESSION command may be issued to the card in order to terminate the 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' or '84'	Open Platform command
INS	'78'	END R-MAC SESSION
P1	'xx'	Reference Control Parameter P1
P2	'xx'	Reference Control Parameter P2
Lc	'xx'	Length of data, if any
Data	'xx xx…'	C-MAC, if needed
Le	'08'	Always returns the R-MAC

Table E- 16: END R-MAC SESSION Command Message

E.5.4.3 Response Message

The data field of response message contains the final R-MAC of the current R-MAC session.

E.5.4.4 Return Processing State

A successful execution of the command is coded by '90' '00'.

This command may either return a general error condition as listed in Section 9.1.3 - *General Error Conditions* or the following error condition:

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table E- 17: END R-MAC SESSION Error Conditions

F. GlobalPlatform Data Values and Card Recognition Data

F.1 Data Values

This is a placeholder where the ISO assigned GlobalPlatform RID, the GlobalPlatform OID and any other required values will be published, possibly by reference into the GlobalPlatform web site.

F.2 Structure of Card Recognition Data

All data is TLV encoded. Application tags not defined in this specification are RFU. Global Platform 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. The actual coding of OIDs is explained in ISO/IEC 7816-6.

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

Table F- 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 Open Platform Card Specification version “**v**”. Thus a card conforming to Open Platform Card Specification 2.1 would use OID **{globalPlatform 2 2 1}**

Note 3: Tag ’63’: The OID **{globalPlatform 3}** indicates an Open Platform card that is uniquely identified by the Issuer Identification Number (IIN) and Card Image Number (CIN), as defined in Sections 6.8.1 - Issuer Identification Number and 6.8.2 - 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 8.6 - Secure Channel Protocol Identifier."**i**" identifies the eventual implementation options such as Implicit or Explicit Initiation as defined in Appendix D.1.1 - *SCP01 Secure* Channel for SCP01 and Appendix E.1.1 - *SCP02 Secure* Channel for SCP02.

Note 5: The data object with tag '65' may contain information about the Open Platform 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: The value of the data object with tag ‘66’ is not defined by Global Platform. It may contain information about the card and chip implementation, such as the OS/RTE or a security kernel. Such information shall be TLV encoded.

F.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 9.8.3.1 - *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

Table F- 2: Security Domain Management Data