GLOBALPLATFORM
ADVANCING STANDARDS FOR SMART CARD GROWTH

Open Platform

Card Specification

Version 2.0.1'

7 April 2000

Table of Contents

PART I INTRODUCTION

PART II ARCHITECTURE

PART III IMPLEMENTATION

PART IV APDU COMMAND REFERENCE

PART V SECURITY AND KEY MANAGEMENT

APPENDICES

Part I
Introduction

This page intentionally left blank.

1. Introduction

Chip card technologies hold great promise as the replacement for magnetic stripe card technology. However, the adoption of chip cards on a mass scale has been slow to develop. One significant reason for this slow adoption is the lack of standards among the many different vendor implementations of chip cards.

The world's leading financial institutions that make up the membership of the Visa International Service Association have identified multi-application chip cards as the strategic technology for future banking cards. In support of its Members, Visa has developed and piloted many chip card projects around the world. Visa has also created some significant standards in the chip card arena. However, these standards have been primarily targeted at either low levels of interoperability, such as the mechanical and electrical standards specified in the EMV specifications, or at the application layer in terms of developing standardized chip credit, debit, and purse applications for Visa Members. The main benefit of these standards has been realized in single-application chip cards, but has not significantly improved the situation for multi-application chip cards.

Beginning in the mid-1990s, a number of very significant breakthroughs occurred in the chip card industry with the introduction of open systems standards for application development. The three leading technologies in this area are Java Card™, Windows for Smart Cards®, and MULTOS™. These technology standards provide an important part of the solution towards the multi-application chip card vision—common programming standards allowing application portability between different manufacturers' card implementations.

Through the Open Platform initiative, Visa has worked with the chip card industry to deliver a missing and critically important chip card standard—a hardware-neutral, vendor-neutral, application-independent card management standard. This new standard provides a common security and card management architecture that protects the most important aspect of a chip card system investment—the infrastructure.

The Open Platform defines a flexible and powerful standard for Card Issuers to create multi-application chip card systems to meet their changing business needs. The standard 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 that portion of the Open Platform standard that must be implemented on chip cards (Open Platform Cards).

1.1 Audience

This specification is intended for card manufacturers developing Open Platform card implementations. Although this specification defines card components, command interfaces, transaction sequences, and interfaces which can be common across many different industries, it is not intended to provide precise implementation detail regarding low-level security which may vary according to industry.

1.2 Normative References

ISO 7816-1:1987	Identification cards –Integrated circuit(s) cards with contacts - Part 1: Physical characteristics
ISO 7816-2:1988	Identification cards – Integrated circuit(s) cards with contacts - Part 2: Dimensions and location of contacts
ISO/IEC 7816-3:1989	Identification cards – Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols
ISO/IEC 7816-3:1992	Identification cards – Integrated circuit(s) cards with contacts - Part 3: Amendment 1: Protocol type T=1, asynchronous half duplex block transmission protocol
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
Java Card™ 2.1	Go to the following web site for Java Card™ documentation: www.javasoft.com
Windows for Smart Cards®	Go to the following web site for more information on Windows for Smart Cards® : www.microsoft.com
Europay, MasterCard, and Visa (EMV): 3.1.1	Integrated Circuit Card Application Specification for Payment Systems; go to the following web site for this specification: www.visa.com

1.3 Terminology and Definitions

APDU (Application Protocol Data Units)	Standard communication messaging protocol between a card acceptance device and a smart card
Application Provider	Entity that owns an application and is responsible for the application's behavior
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
Cryptogram	Result of a cryptographic operation
Data Authentication Pattern (DAP)	Used to authenticate the origin and/or integrity of the data. Refer to section 4.3.1 The Data Authentication Pattern.
Decryption	The reversal of a corresponding encryption; decryption is performed using a symmetric secret key or an asymmetric private key to retrieve the original message
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
Encryption	The reversible transformation of data by a cryptographic algorithm to produce a cryptogram; encryption can be performed using a symmetric key or asymmetric key
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 and data downloading, and transactional processing.
Load File	A complete block of data transferred to an Open Platform card containing certification data and a Load File Data Block
Load File Data Block	A block of data containing one or more application(s) and libraries or support information for the application(s) as required by the specific platform.
Executable Load File	A portion of card memory that contains the executable code for one or more applications.
MAC (Message Authentication Code)	A symmetric cryptographic transformation of data that provides data origin authentication and data integrity
Private Key	The private component of an asymmetric key pair; the private key is always kept secret by its owner; the private key is used to decrypt cryptograms that are encrypted using the corresponding public key; it is also used to digitally sign messages for authentication
Public key	The public component of the asymmetric key pair; the public key is exposed and available to users but often is encapsulated within a certificate
Symmetric Cryptography	A cryptographic technique that uses the same secret key for both the originator's and the recipient's transformation;

1.4 Abbreviations and Notations

AID	Application Identifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
BER	Basic Encoding Rules
CLA	Class Byte of the Command Message
CPLC	Card Production Life Cycle
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
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
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, and Adleman asymmetric algorithm
SW	Status Word
SW1	Status Word One
SW2	Status Word Two
TLV	Tag Length Value

1.5 Revisions History

This section provides a brief overview of the revisions made to the Open Platform 2.0 card specifications.

Wording and formatting of the document has been modified for clarity.

The following is a list of the minor points and differences to note.

- Anything relating to a specific implementation of Open Platform has been removed from the main body of the document and is now detailed in Part V and the Appendices.
- Anything specific relating to the personalization of Open Platform or applications has been removed.
- Some changes relate specifically to the Java Card™ implementation of Open Platform and these are listed in the beginning of Appendix A.
- All the issues identified in the FAQ document dated April-June and the FAQ documents dated October-November and relating strictly to Open Platform, have been included in this version. The one caveat to this is point 3.1.20 of the FAQ document dated October-November 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 describes a specific use of security and key management that was not present in Open Platform 2.0.

This page intentionally left blank.

Part II
Architecture

This page intentionally left blank.

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.

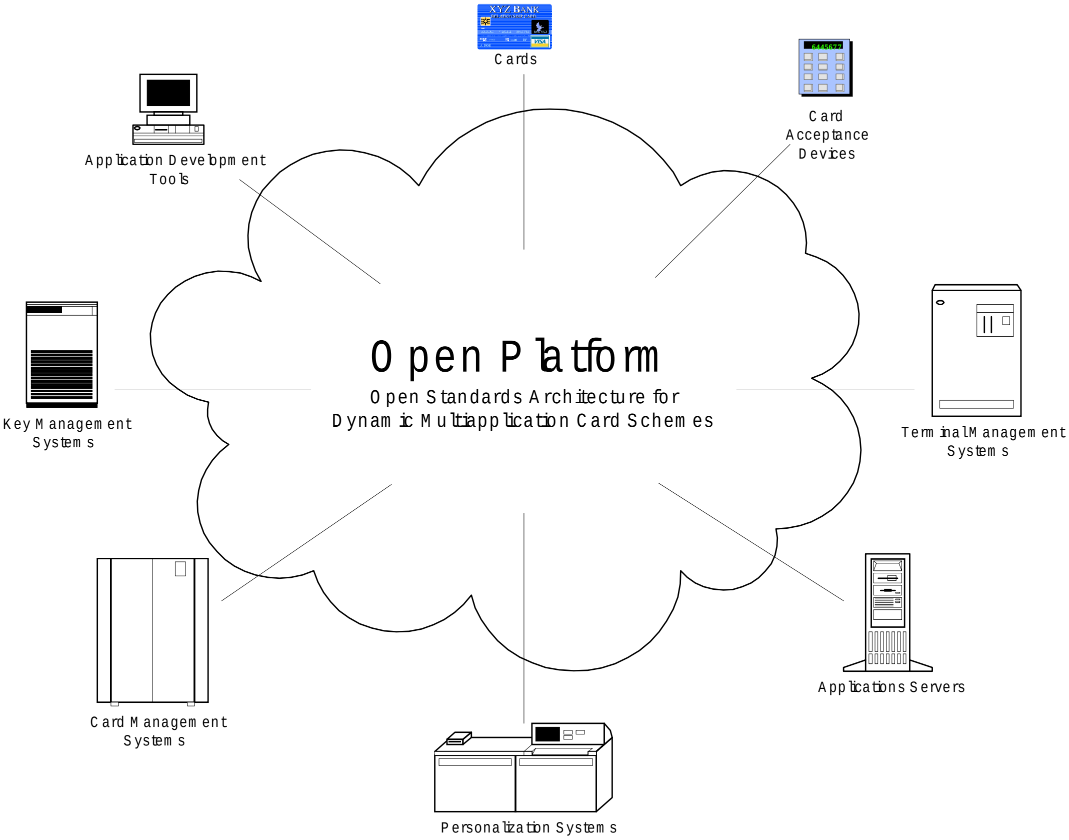


Figure 2-1: Open Platform architecture

The Open Platform architecture is designed to provide Card Issuers with the systems management architecture for managing these powerful cards. Although the Open Platform is based on the paradigm that there is one single Card Issuer for a card, it is designed with the idea that the Card Issuer will have an ever-changing array of business partners who may want to share application space on the Card Issuer's cards.

The Open Platform gives Card Issuers the power to manage and change the content of their cards while also offering the ultimate in flexibility by enabling them to share control of some of their card space 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 illustration shows the components in a sample card configuration which includes an application from the Card Issuer as well an application from one of the Card Issuer's partners who are referred to as Application Providers.

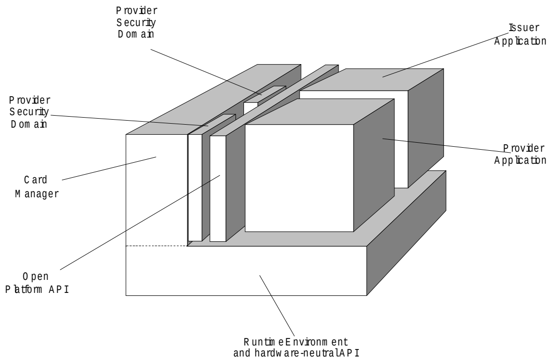


Figure 3-1: Open Platform card and component relationships

All applications are assumed to exist in a secure runtime environment that includes a hardware-neutral 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 designed 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 is considered the on-card representative of the Card Issuer. It manages the complete runtime environment for applications and is the overall system and security controller on an Open Platform card. The Card Manager provides an interface for applications to access its services and has a well-defined external APDU interface to ensure that all implementations behave consistently and can be managed by the same off-card management systems.

The Card Manager is responsible for overall card security and includes the Card Issuer's Security Domain which supports security services such as key handling, encryption, decryption, digital signature generation and verification for the Card Issuer's applications. Additionally, the Card Manager performs the application loading and related card content management on behalf of the Card Issuer while also managing the installation of applications that are loaded to the card by other Application Providers.

Another important function provided by the Card Manager is APDU command dispatching and application selection. When a SELECT command is received, the Card Manager sets the application referenced in the SELECT command to be the 'selected application.'

The Card Manager owns and uses an internal registry as an information resource for card management. The registry contains information for managing the card, load file executables, and application life cycle states, card blocking, PINs, application installation and deletion, and the authorization of memory allocation.

The Card Manager has the capability of behaving and functioning as an application. Therefore, the Card Manager has application characteristics such as application AID, application life cycle states, and it can select itself as the 'selected application'. An example of the Card Manager functioning as an application is when the Card Issuer selects the Card Manager to load a new application to the card.

See Chapter 6 'Card Manager,' for detailed descriptions of the functions and responsibilities of the Card Manager.

3.3 Security Domains

A Security Domain is the on-card representative of an Application Provider. It is a special key management application that may provide cryptographic services for all the applications owned by a particular Application Provider.

Security Domains are established on behalf of an Application Provider when the provider requires the use of keys on the card which are completely separate and isolated from the Card Issuer's keys. Because the Card Issuer owns applications on the card as well, the Card Manager includes the Card Issuer's Security Domain as a sub-component.

3.4 Open Platform API

The Open Platform API provides applications access to card management services administered by the Card Manager. In some cases this access provides a service to the application and in other cases, the API is used by applications to assist the Card Manager in the management of the card content.

3.5 Applications

Applications are software components that perform many functions on the card. Open Platform cards provide specifically for multiple application to co-exist. From an implementation perspective, applications can be classified into immutable persistent memory applications and mutable persistent memory applications.

- Immutable persistent memory applications are loaded into immutable persistent memory during the manufacturing stage and cannot be altered (they can be disabled, however).
- Mutable persistent memory applications can be loaded, installed or removed during initialization or post-issuance.

However, some applications may have components that reside in both immutable persistent memory and mutable persistent memory. This is especially true for application data for an immutable persistent memory application, which may have static data values in immutable persistent memory and data variables in mutable persistent memory.

The Open Platform is designed to support applications regardless of where they are stored in the card and throughout the full life cycle of a card.

This page intentionally left blank.

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 and then focuses on the specific roles and responsibilities of the Card Issuer and Application Providers as the owners of the card and applications, and the role and responsibilities of the on-card components.

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, Card Manager, Security Domains, applications, and associated data.

To ensure card security and integrity, the Open Platform is designed to support a range of well known, cryptographically provided security mechanisms for access control, data integrity, resource availability, confidentiality, and authentication. Choice of security policy and cryptography is assumed to be industry and product specific (see Part V).

Because the cards are just one 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 and may also vary depending on industry and product.

4.2 Security Roles and Responsibilities

4.2.1 Card Issuer

The Card Issuer is responsible for:

- generating and loading the Card Manager keys, if not otherwise performed by the Card Manufacturer;
- 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 load and initialize Security Domains;
- managing security relationships on the card;
- determining Card Manager policy with regards to card and application life cycle management, velocity check levels, and other security parameters;
- managing application content both on a pre-issuance and post-issuance basis; and,
- cryptographically authorizing Load Files that are to be loaded through Application Provider Security Domains with Delegated Management privilege (see Chapter 7, 'Security Domains,' for a description of Delegated Management).

4.2.2 Application Provider

The Application Provider is responsible for:

- generating the keys for the Application Provider's Security Domain or obtaining Security Domain keys from a Trusted Third Party;
- working with the Card Issuer to load generated keys into the Security Domain;
- providing applications that meet the Card Issuer's security standards and policies;
- providing Load Files according to defined Application Provider's security standards and policies;
- managing keys for Security Domains;
- obtaining pre-authorization from the Card Issuer to load and install applications onto the Card Issuer's card during post-issuance; and,
- returning Load and Install Receipts to the Card Issuer following Delegated Management sessions.

4.2.3 Runtime Environment

The runtime environment is responsible for:

- providing a secure, consistent interface to all applications; and,
- performing secure memory management to ensure that each application's code and data is protected from unauthorized access from within the card.

4.2.4 Card Manager

The Card Manager shall be able to:

- initialize the card in a secure manner;
- communicate with off-card entities in accordance with the Card Issuer's security policies;
- manage the secure loading, updating, and deletion of applications on-card in a post-issuance state;
- manage global card data including card and application life cycle states;
- perform optional internal velocity checks on the Card Global PIN to prevent card and application access violations;
- provide cryptographic protection for the Card Issuer's applications during their personalization phase; and,
- perform the verification of Delegated Management functions.

4.2.5 Security Domain

Security Domains shall be able to:

- communicate with off-card entities in accordance with the Card Issuer's security policies; and
- provide cryptographic protection for the Application Provider's applications during their personalization (key loading) phase.

Security Domains with mandated DAP verification or general DAP verification privilege are also responsible for:

- Performing the verification of the Load File Data Block Data Authentication Pattern(s).

Security Domains with Delegated Management privilege are also responsible for:

- performing the Delegated Management processes to securely load, update and delete applications post-issuance; and,
- returning Load, Install and Delete Receipts during Delegated Management.

4.2.6 Application

Applications are responsible for:

- exposing only data and resources that are necessary for proper application function; and,

- performing internal velocity and other checking to determine if the application should block itself.

4.2.7 Back-end System

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, which communicate with the cards, perform the verifications, and manage the off-card key databases, also must be trusted). Responsible personnel, secure operating systems, and system security policies are all essential components that secure the back-end systems. These assumptions are beyond the scope of the Open Platform Card Architecture and therefore are not discussed within this documentation.

4.3 Cryptographic support

One of the major requirements for an Open Platform card is the ability to provide a minimal level of cryptographic functionality. This cryptography is, for example, used by the Open Platform for the generation of Data Authentication Patterns, and is available for use by the applications present on the card.

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/Adelman (RSA) algorithm.

Services to encrypt and decrypt any pattern of data using these algorithms must be available to applications.
Due to certain restrictions placed upon cryptography by governments, features must be available to allow some or all of these cryptographic services to be disabled completely or restricted to certain applications on the card depending on the governmental controls and the environment in which the card is operating.

4.3.1 The Data Authentication Pattern

The term DAP is used throughout this document to represent an additional value associated with a message or block of data. The purpose of a DAP is to provide a method of verifying the transmission, source and/or integrity of either a message or a particular block of data within a message or spread over a number of messages.

Various methods and standards, known and unknown to the writers, exist for generating a DAP. It is expected that each industry that utilizes the Open Platform would have preferred methods of achieving a similar result.

Within this document, the term DAP is used in various places but does not necessarily imply one method being used to calculate the pattern. The following is a list of references made to a DAP, the intended purpose of the DAP and an example of a method which could be used to generate the DAP.

4.3.1.1 Load File DAP (HASH)

The Load File DAP is intended to verify the transmission of a complete Load File to an Open Platform card. Its primary intention is to provide a method for the receiving entity to verify that the content of the Load File has not been inadvertently modified. It also has another function in that it is included in the Load Token (see 4.3.1.3) as a link back to the original Load File.

As this DAP is only intended to ensure the correct transmission of data, it is not required to use a cryptographic algorithm to generate this DAP. A hashing method such as SHA1 may be suitable.

4.3.1.2 Load File Data Block DAP

The Load File Data Block DAP is generated by an off-card entity (other then the Card issuer) and intended to verify the integrity of a Load File Data Block which contains the actual code of the application. Its intention is to provide a method for an entity with a presence on the card to verify that the Load File Data Block being transferred to the card is valid and has not been tampered with in any way.

A cryptographic algorithm for generating a digital signature or message authentication code (MAC) across the complete Load File Data Block is required.

With symmetric cryptography, DES in Cipher Block Chaining (CBC) mode may be used to generate a MAC of the Load File Data Block.

Asymmetric cryptography provides many solutions for generating a digital signature over a block of data an example being first hashing the Load File Data Block using the SHA-1 algorithm and then signing the resultant hash with an RSA private key.

4.3.1.3 Load and Install DAP (Load or Install Token)

The Load DAP and Install DAP are generated by the Card Issuer and used in Delegated Management to provide the Issuer of an Open Platform card control over what content is being loaded when not performing the process itself. This is done by providing the Card Manager the ability to verify that certain functions are being carried out only by authorized Application Providers with known content that has not been modified.

Cryptographic algorithms are required and apply to a set of information identifying what is being requested of the card.

The Load DAP is also intended to verify that only a certain Load File is being transferred to the card and it achieves this by incorporating the Load File DAP (see 4.3.1.1) as one of its pieces of information.

With symmetric cryptography, DES in Cipher Block Chaining (CBC) mode may be used to generate a MAC of the required information.

Asymmetric cryptography provides many solutions for generating a digital signature over a block of data an example being first hashing the required information using the SHA-1 algorithm and then signing the hash with an RSA private key.

4.3.1.4 Receipt DAP

The receipt DAP is generated by the Card manager during Delegated Management as proof to the Card Issuer from the Application Provider that the content of the card has been modified i.e. content added or deleted.

A cryptographic algorithm is required and applied to a set of information identifying what has being modified on the card.

With symmetric cryptography, DES in Cipher Block Chaining (CBC) mode may be used to generate a MAC of the required information.

Asymmetric cryptography provides many solutions for generating a digital signature over a block of data an example being first hashing the required information using the SHA-1 algorithm and then signing the hash with an RSA private key.

4.3.1.5 Message DAP (MAC)

The message DAP is generated by an off-card entity and used to ensure the integrity of a message being transmitted by the off-card entity to the card (see 4.3.2).

With symmetric cryptography, DES in Cipher Block Chaining (CBC) mode may be used to generate a MAC of the required information.

Asymmetric cryptography provides many solutions for generating a digital signature over a block of data an example being first hashing the required information using the SHA-1 algorithm and then signing the hash with an RSA private key.

4.3.2 Secure Messaging

An Open Platform card may provide security services related to information exchanged between the card and an off-card entity. Security requirements may not necessarily apply to each individual message being transmitted and only apply to the environment and/or context in which messages are transmitted. The concept of the life cycle of the card (see 5.1 ‘Card Manager Life Cycle’), may determine the level of security requirements. A higher level of security may be required if the card is in the possession of the end user. Whereas a lower level or limited security may be sufficient if the card is in a known secure environment such as a personalization bureau.

An Open Platform card offers the following security services associated with messages:

- Confidentiality: In some environments (e.g. an open network), hiding the content of a message from external entities requires the confidential data to be encrypted using a cryptographic algorithm such as DES or RSA.
- Integrity: insuring that the data transmitted by the off-card entity is received unaltered by the card requires a MAC or a digital signature (message DAP) of the data to be transmitted along with the data.

Mutual Authentication at the beginning of a communication session, establishing a Secure Channel, is also recommended prior to any relevant data being transferred to a card. Mutual Authentication provides a way for a card and an off-card entity to authenticate each other and exchange security related information regarding message confidentiality and integrity.

Mutual Authentication combined with the card life cycle state allows the card to assume its environment and/or context. Note that Mutual Authentication is only provided for a limited time (i.e. a session) and is valid only for the messages within that session. If the session ends for any reason, Mutual Authentication should be reiterated to initiate a new Secure Channel.

This page intentionally left blank.

Part III
Implementation

This page intentionally left blank.

5. Life Cycle Models

The Open Platform defines life cycle state models to control the functionality and security of the following Open Platform components: Card Manager, Executable Load Files, and Applications. These life cycle states are entries within the registry of the Card Manager and accessible to authenticated off-card entities or to applications requesting their own life cycle states. The life cycle models of each component are presented in this section.

5.1 Card Manager Life Cycle

The Card Manager is responsible for maintaining the overall security and administration of the card and its contents. Because the Card Manager plays this supervisory role over the entire card, its life cycle can be thought of as the life cycle of the card.

Although a card's life includes activities prior to the initial life cycle state of the Card Manager, these activities are considered manufacturer specific and therefore outside the scope of this specification. The card's life cycle from an Open Platform perspective only has meaning at the beginning of the Card Manager life cycle (OP_READY).

The Card Manager owns and maintains the card life cycle state information within the registry and manages the requested state transitions in response to APDU commands.

The end of the Card Manager life cycle is considered to be equivalent to the end of the card's life cycle (TERMINATED).

5.1.1 Card Manager Life Cycle States

The card life cycle states are:

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

Life cycle states OP_READY and INITIALIZED are intended for use during the pre-issuance phases of the card life cycle. Life cycle states SECURED, CM_LOCKED and TERMINATED are intended for use during the post-issuance portion of the card’s life cycle although it is possible to terminate the card at any point in its life cycle.

5.1.1.1 OP_READY

In the irreversible card life cycle state OP_READY, all the basic functionality of the run-time environment is available and the Card Manager, acting as the selected application, is ready to receive, execute and respond to APDU commands.

The card is assumed to have the following functionality in the state OP_READY:

1. Executable Load Files that were included in immutable persistent memory are available.
2. The run-time environment is ready for execution.
3. Card Manager acts as the selected application.
4. An Initialization key is available within the Card Manager.

During this life cycle state, Security Domains and their key sets may be loaded in order to maintain cryptographic key separation from the Card Issuer’s keys. Also, for applications expected to be available at card issuance, the loading of application not already present into memory and the installation of any applications can occur. Additionally, if any personalization information is available at this stage, it is possible to personalize applications, if required.

5.1.1.2 INITIALIZED

The irreversible card life cycle state INITIALIZED is an administrative card production state. Its exact definition is manufacturer, issuer and/or implementation dependent. This state is commonly used to indicate that some initial data has been populated (e.g. keys and Card Manager data) but that the card is not yet ready to be issued to the cardholder.

Only an authenticated Card Issuer can initiate the transition from OP_READY to INITIALIZED.

5.1.1.3 SECURED

The irreversible Card Manager life cycle state SECURED is the normal, intended operating life cycle state of the card during issuance. This state is the indicator for Card Manager to enforce the Card Issuer’s security policies related to post-issuance card behavior such as application loading, installation and activation.

The card is assumed to have the following functionality in the state SECURED:

1. The Card Manager contains all necessary key sets and security elements for full functionality.
2. Card Issuer initiated card content changes can be carried out through the Card Manager.
3. Post-issuance personalization of applications belonging to the Card Issuer can be carried out via the Card Manager.
4. Active Security Domains contain all necessary key sets and security elements for full functionality.
5. Application Provider initiated card content changes can be carried out via Security Domains that have Delegated Management privilege.
6. Post-issuance personalization of applications belonging to the Application Providers can be carried out via the Security Domains.

Only an authenticated Card Issuer can initiate the transition from INITIALIZED or CM_LOCKED to SECURED.

5.1.1.4 CM_LOCKED

The reversible state CM_LOCKED is used to tell the Card Manager to temporarily disable all applications on the card except for the Card Manager. This state is present to provide the Card Issuer with the ability to identify security threats either internal or external while application functionality is disabled on the card.

Setting the Card Manager to this state means that the card will no longer function except via the Card Manager which is controlled by the Card Issuer.

The Card Manager itself, an application on the card with the relevant privilege or an authenticated Card Issuer can initiate the transition from SECURED to CM_LOCKED.

By communicating with the card, the Card Issuer may determine that the threat is either no longer present or of limited severity. If this is the case, the Card Issuer can reset the card to its previous operating state.

5.1.1.5 TERMINATED

The Card Manager is set to the life cycle state TERMINATED to permanently disable all card functionality including the functionality of the Card Manager itself. This state is created as a mechanism for a privileged entity to logically ‘destroy’ the card for such reasons as the detection of a severe security threat or expiration of the card.

The Card Manager state TERMINATED is irreversible and signals the end of the card’s life cycle and the card. (The card shall no longer respond to any communication).
The Card Manager itself, an application on the card with the relevant

privilege or an authenticated Card Issuer can initiate the transition from any of the previous state to TERMINATED.

5.1.2 Card Manager Life Cycle Transitions

Figure 5-1 illustrates the state transition diagram for the Card Manager life cycle. This can typically be viewed as a sequential process with certain possibilities for reversing a state transition or skipping states.

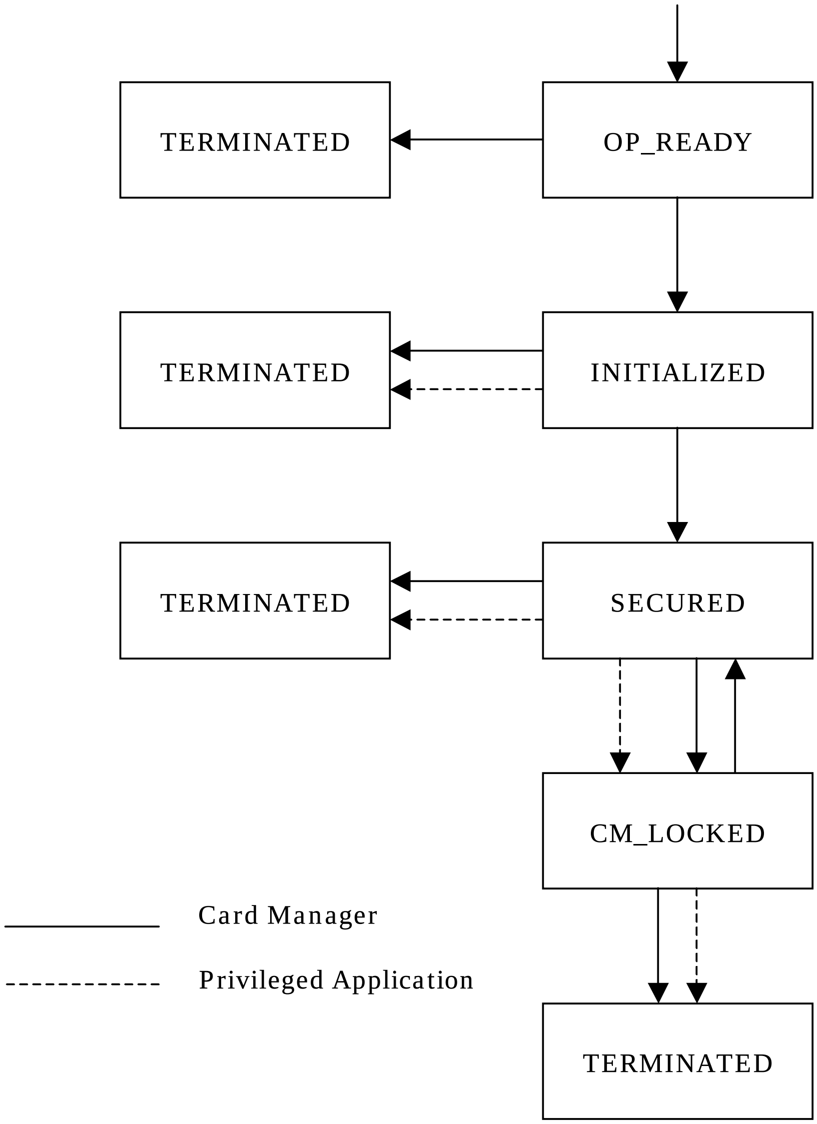


Figure 5-1: Card Manager life cycle state transitions

5.2 Executable Load File Life Cycle

The Executable Load File is the actual container of one or more application's executable code.

An Executable Load File can be initially present on a card if placed in memory by the Card Manufacturer in a vendor specific manner. This Executable Load File could reside in immutable persistent or mutable persistent memory.

An Executable Load File can also become present on the card through the process of transferring a Load File to the card. While the Load File and Load File Data Block may contain additional information required for security, linking and/or verification, the Executable Load File is the resulting usable image stored in mutable persistent memory.

The Card Manager owns and maintains the Executable Load File life cycle state information within the registry and manages the life cycle states directly or indirectly through a Security Domain with the Delegated Management privilege.

5.2.1 Executable Load File Life Cycle States

The Executable Load File life cycle states are:

1. LOADED
2. LOGICALLY_DELETED

5.2.1.1 LOADED

The Card Manager considers all Executable Load Files that are present in the card and available for use either from 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 becomes an entry in the registry following the successful completion of the load process. Executable Load Files placed on the card by the card manufacturer will automatically have entries within the registry.

5.2.1.2 LOGICALLY_DELETED

The Card Manager may receive a request either directly from the Card Issuer or indirectly from a Security Domain with the Delegated Management privilege 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 Executable Load File is logically deleted by setting its state to LOGICALLY_DELETED.

The space used to store a logically deleted Executable Load Files cannot be reclaimed, and its entry in the registry also remains.

The space previously used to store a physically deleted Executable Load File is reclaimed and can be reused by the memory manager. The entry within the registry is also removed, and the card retains no record of the Executable Load File's previous existence.

LOGICALLY_DELETED state cannot be reversed. The Card Manager considers the state LOGICALLY_DELETED to be equivalent to the physical deletion of the Executable Load File.

5.2.2 Executable Load File Life Cycle Transitions

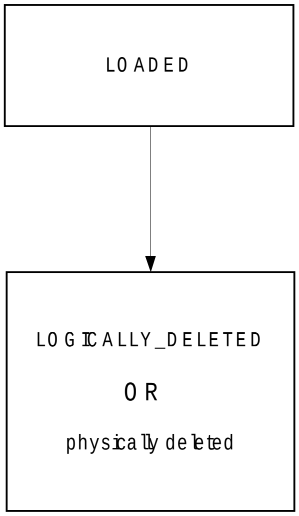


Figure 5-2: Executable Load File life cycle state transitions

5.3 Application Life Cycle

The Application life cycle begins when an application is installed in the card. This installation occurs from an Executable Load File that is present on the card. The Card Manager is responsible for managing, directly or indirectly through a Security Domain with the Delegated Management privilege, the initial life cycle state transitions of an application before it is fully functional.

Once an application is available for selection from the outside world, it takes control of managing its own life cycle. The life cycle states modifiable by the application are used by the Card Manager to inform off-card entities of the status of applications. The Card Manager changes the state in the registry on instruction from the application.

The definitions of these state transitions are application dependent and are only known to the application.

The Card Manager can take control of the Application life cycle at any stage if the Card Manager or the Card Issuer detects a security problem, or if the application is to be deleted either physically or logically.

The Card Manager owns and maintains the application life cycle state information within the registry.

5.3.1 Application Life Cycle States

The Application life cycle states are:

1. INSTALLED
2. SELECTABLE
3. PERSONALIZED
4. BLOCKED
5. LOCKED
6. LOGICALLY_DELETED

The application becomes an entry in the registry and Card Manager sets the life cycle of an application to its initial state of INSTALLED during the application installation process. The Card Manager is also responsible for making the application available for selection by setting its life cycle to SELECTABLE. While the Card Manager actually performs these transitions, they may be initiated from a Security Domain with the Delegated Management privilege.

The application manages its life cycle transitioning from SELECTABLE to PERSONALIZED and optionally to BLOCKED. From the BLOCKED state, the application can transition its life cycle state back to its previous state. The Card Manager CANNOT perform these transitions without instruction from the application (i.e. SELECTABLE to PERSONALIZED, SELECTABLE to BLOCKED, PERSONALIZED to BLOCKED, BLOCKED to PERSONALIZED or BLOCKED to SELECTABLE)

At any point in the application life cycle, the Card Manager can take control for security protection by setting the application life cycle state to LOCKED. Also, if the application is to be removed from the card, the Card Manager controls that process and sets the appropriate resultant life cycle state if any.

Table 5-1 summarizes the possible application life cycle transitions. The entry for "Executable Load File " is included for reasons of completeness, since the application is installed from an Executable Load File. The entry “deleted” is also included for completeness, although the application may disappear altogether or transition to the LOGICALLY_DELETED state. The table entry defines whether the application itself, the Card Manager, or a Security Domain with the Delegated Management privilege, initiates the

Application life cycle transition. Empty fields indicate that the transition is not allowed. Transitions that are reversible require appropriate checking to ensure correct reversibility (e.g. A transition from SELECTABLE to BLOCKED and then to PERSONALIZED is not allowed).

Table 5-1: Application life cycle transitions

From \ To	**INSTALLED**	**SELECTABLE**	**PERSONALIZED**	**BLOCKED**	**LOCKED**	*Deleted*
Executable Load File	Card Manager / Security Domain					
INSTALLED		Card Manager / Security Domain			Card Manager	Card Manager / Security Domain
SELECTABLE			Application	Application	Card Manager	Card Manager / Security Domain
PERSONALIZED				Application	Card Manager	Card Manager / Security Domain
BLOCKED		Application	Application		Card Manager	Card Manager / Security Domain
LOCKED	Card Manager	Card Manager	Card Manager	Card Manager		Card Manager / Security Domain

5.3.1.1 INSTALLED

The state INSTALLED in the context of Open Platform means that the application executable code has been properly linked and that any necessary memory allocation has taken place so that the application can execute internally to the card. The application becomes an entry in the registry and is visible to authenticated off-card entities. The install process specifically does not include establishing the application as a general externally visible application (i.e. not yet selectable). Moreover, the install process is not intended to incorporate personalization of the application, which may occur as a separate step.

5.3.1.2 SELECTABLE

The state SELECTABLE is used to make an application available to receive commands from outside the card. Applications must be properly installed and functional before they can be set to the life cycle state SELECTABLE. The transition to SELECTABLE can be combined with the application installation process so that the INSTALLED state is never visible to an off-card entity. (See the INSTALL [for install and make selectable] command).

The behavior of the application while in this state is determined by the application itself.

Once an application has been set to the state SELECTABLE, it can set its own state to BLOCKED and at some later stage reverse its own state back to SELECTABLE.

5.3.1.3 PERSONALIZED

The definition of what is required for an application to transition to the state PERSONALIZED is application dependent but is intended to indicate that the application has all the necessary personalization data and keys for full runtime functionality (i.e. usable in its intended environment).

The behavior of the application, while in this state, is determined by the application itself.

Once an application has been set to the state PERSONALIZED, it can set its own state to BLOCKED and at some later stage reverse its own state back to PERSONALIZED.

5.3.1.4 BLOCKED

The definition of what is required for an application to transition to the state BLOCKED is application dependent but is intended to indicate that an application specific security problem has been detected either from within the application or from outside the card.

The behavior of the application, while in this state, is determined by the application itself. Typically, it would limit its own functionality to only allow it to transition itself back to its previous state.

5.3.1.5 LOCKED

The Card Manager or Card Issuer uses the life cycle state LOCKED as a security management control to prevent the selection, and therefore the execution, of the application.

If the Card Manager detects a threat from within the card and determines that the threat is associated with a particular application, that application can be prevented from further selection by being set to a LOCKED state by the Card Manager. Alternatively, the Card Issuer may determine that a particular application on the card needs to be locked for a business or security reason and can initiate the application life cycle transition via the Card Manager.

Once an application is set to LOCKED, the application can only become functional again if the Card Manager sets the application's life cycle back to the state that it held just prior to being set to the state LOCKED.

5.3.1.6 LOGICALLY_DELETED

The Card Manager may receive a request either directly from the Card Issuer or indirectly from a Security Domain with the Delegated Management privilege to delete an application. If the application cannot be physically deleted (e.g., because it is stored in immutable persistent memory), the application is logically deleted by setting its state to LOGICALLY_DELETED.

Once an application has been set to the state LOGICALLY_DELETED, it cannot be reversed. The Card Manager considers the state LOGICALLY_DELETED to be equivalent to the physical deletion of the application.

The space used to store a logically deleted application cannot be reclaimed, and its entry in the registry also remains.

The space previously used to store a physically deleted application is reclaimed and can be reused by the memory manager. The entry within the registry is also removed, and the card retains no record of the deleted application's previous existence.

5.3.2 Application Life Cycle Transitions

Figure 5-4 illustrates the state transition diagram for the Application life cycle. This can typically be viewed as a sequential process with certain possibilities for reversing a state transition or skipping states.

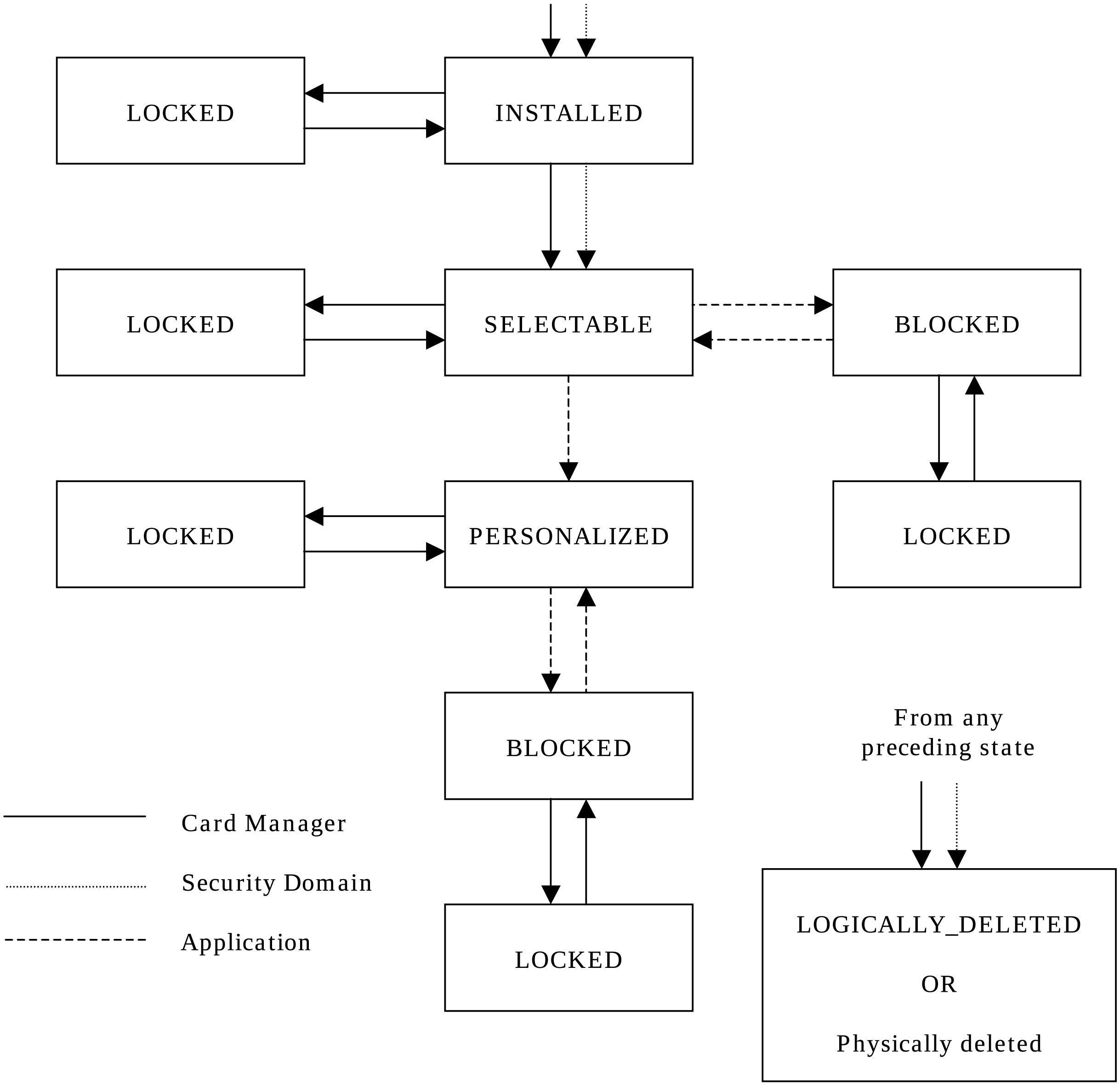


Figure 5-3: Application life cycle state transitions

5.4 Sample Life Cycle Illustration

This section provides a description of a sample Open Platform card and its life cycle state transitions from the card's creation to the time it is terminated. It also shows the status of several applications and their relationship with the card life cycle states. Figure 5-5 illustrates these sample life cycle states:

- Application A is loaded into immutable persistent memory when the chip is manufactured. It is then installed during initialization. It is used throughout the card’s life until the card is terminated and as long as the card is not in a LOCKED state.
- Application B is also loaded into immutable persistent memory during manufacturing and installed prior to the card being initialized. Unlike Application A, Application B is deleted some time during the card’s life before the card is terminated. Because Application B is stored in immutable persistent memory, Application B cannot be physically deleted from the card so its life cycle state is set to LOGICALLY_DELETED. (Note that applications can also be loaded into mutable persistent memory in the OP_READY state.)
- Application C is contained within an Executable Load File C that is loaded into mutable persistent memory during initialization. The application is then installed post-issuance while the card is in the card state SECURED. The application is used for some time and then deleted before the card is terminated. Because the application and its Load File are stored in mutable persistent memory, the application and associated Load File can be purged from mutable persistent memory and the memory space reclaimed for reuse.
- Application D is contained within an Executable Load File D that is loaded and installed during initialization. It is used during the full lifetime of the card until the card is terminated and as long as the card is not in a LOCKED state.
- Application E is contained within an Executable Load File E that is loaded and installed post-issuance while in the card state SECURED. Application E is used until the card is terminated and as long as the card is not in a LOCKED state.
- Application F is contained within Load File F which is also loaded and installed post-issuance. Application F and its Executable Load File are deleted some time during the card’s lifetime.

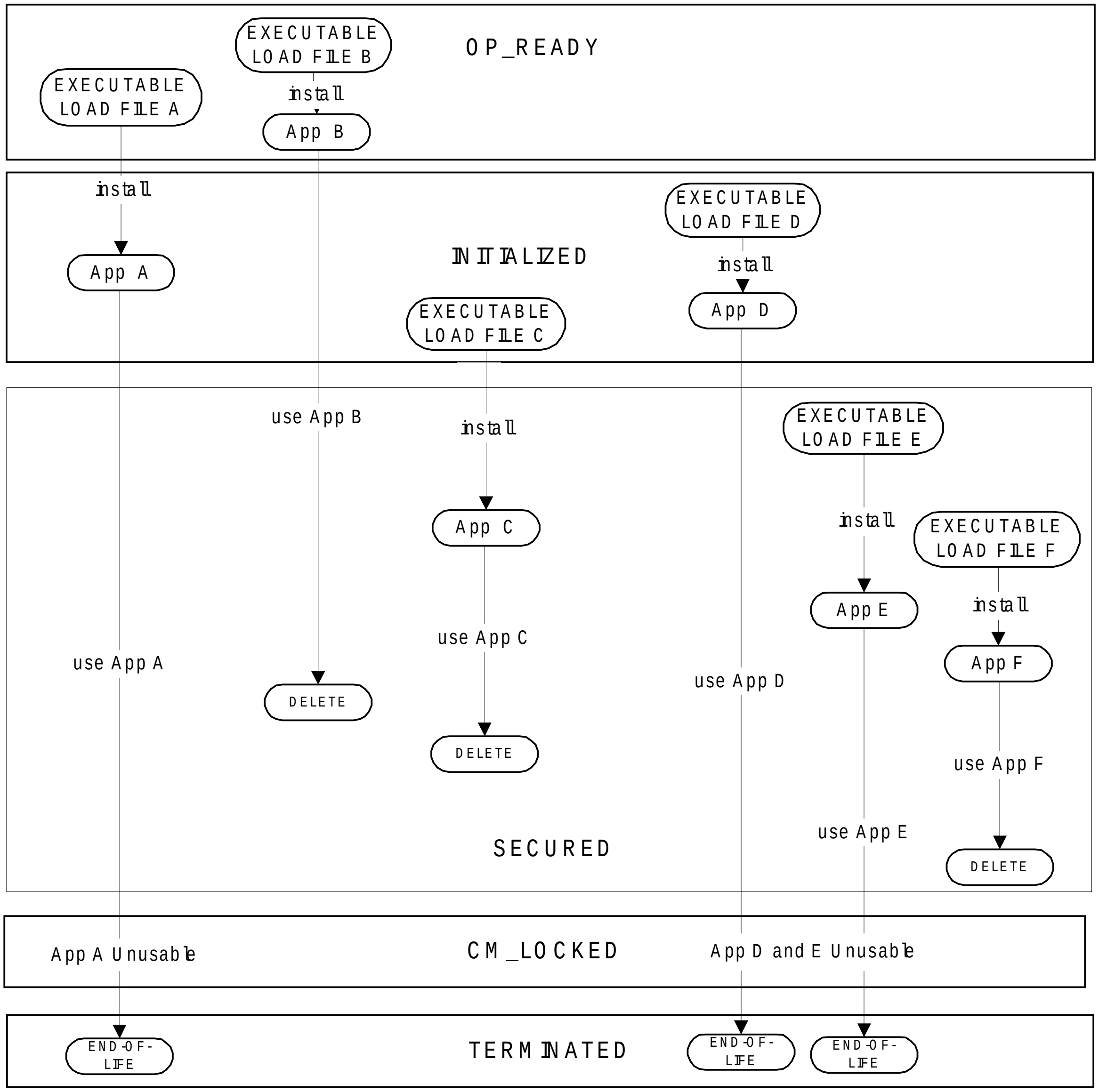


Figure 5-4: Sample card life cycle and application life cycles

This page intentionally left blank.

6. Card Manager

6.1 Overview

The Card Manager is the card component responsible for all card administration and card system service functions. As some of the Card Manager's functionality may be similar to that performed by the Runtime Environment, the Card Manager could be an integral part of the Runtime Environment. Its exact functionality is defined by the industry in which it will be utilized, and it is owned and controlled by the Card Issuer. The Card Manager's role is similar to that of a system 'manager' or 'administrator' with the specific purpose of allowing a Card Issuer the ability to access, manage and control a card's contents, the card's life cycle, and that of the applications.

The Card Manager supports the following functions and data:

- Command Dispatch

  Application selection
  Command dispatching

- Content Management

  Content Verification
  Content Loading
  Content Installation
  Content Removal

- Security Management

  Security Domain Locking
  Application Locking
  Card Manager Locking
  Card Termination
  Application privilege usage
  Security Domain privilege usage
  Global PIN with Velocity Checking
  Tracing and Event Logging

- Issuer Security Domain

  Secure communication support for applications during:
  Application personalization

Application runtime

- Card Manager Data

  Card Manager AID
  Card Production Life Cycle Data
  Card Issuer Identifier

The Card Manager architecture can be illustrated as a collection of system functions built upon a system data manager referred to as the registry. The registry is a data store required to support the various system functions of the Card Manager:

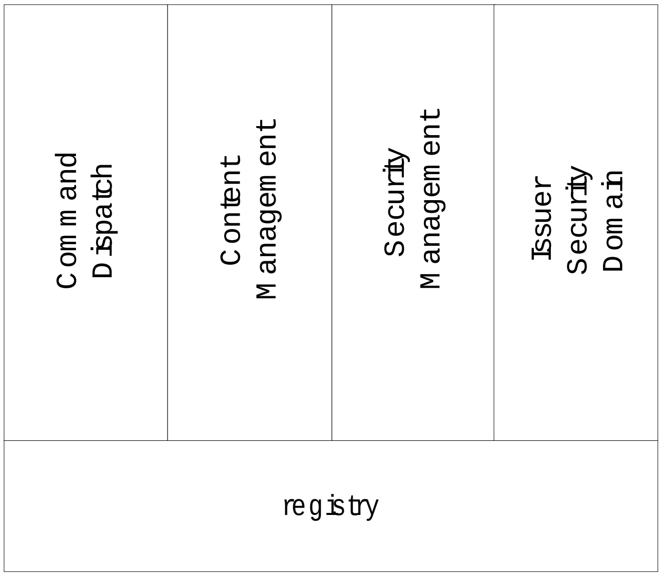


Figure 6-1: Card Manager architecture

The Card Manager uses the registry for many purposes, including the following:

1. Command Dispatch:
   - determine if an application is present and available to respond to SELECT commands; and,
   - dispatch commands to the selected application for command processing.
2. Card Content Management:
   - store state and management information about newly loaded Executable Load Files, installed applications and deleted Executable Load Files or 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; and,
- identify an application's associated Security Domain to provide access from that application to its Security Domain services.

3. 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 of 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; and,
   - verify an application's accessibility to functionality that require privileges.
4. Issuer Security Domain:
   - identify the Card Issuer's applications to provide access from those applications to Issuer Security Domain services of the Card Manager.

To support the required Card Manager functions, the Card Manager exposes an APDU command interface to off-card entities and an API to on-card applications.

The APDU command interface exists primarily to manage card administration functions, and in that capacity only the authenticated Card Issuer is allowed to access this interface. The API is the interface that applications use to request services from the Card Manager. Any application can use the API to communicate with the Card Manager from within the card; however, the Card Manager retains control of the services and determines on an individual request basis whether or not to grant each request.

The following sections detail the implementation and behavior of each of the Card Manager functions.

6.2 Card Manager Life Cycle

Since the Card Manager is responsible for maintaining the overall security and administration of a card and its contents, its life cycle can be thought of as the life cycle of the card. In this context, the valid life cycle states for the

Card Manager are OP_READY, INITIALIZED, SECURED, CM_LOCKED and TERMINATED as described in Chapter 5, ‘Life Cycle Models’.

6.3 Secure Communications

The Open Platform architecture is designed to enable secure communication between the Card Issuer and the Card Manager, between Application Providers and their Security Domains, and between Application Providers and their applications. The commands were designed to support both integrity and confidentiality at the command level but they do not require a specific form of cryptography.

It is assumed that one-way or mutual authentication may be required to establish a secure communication session between the various on-card and off-card entities; however, the Open Platform does not require that this process take place within the context of each transaction. The requirement and format of this process is considered industry and product specific.

6.4 Application Access to Card Manager Services

Applications have the ability to access and/or modify some content known or managed by the Card Manager. The manner in which this access is provided to the application is platform specific but must encompass the possibility of:

- retrieving the application’s own current life cycle state stored by the Card Manager in the registry;
- retrieving the current life cycle state of the Card Manager;
- retrieving card production life cycle data;
- retrieving the services of a Security Domain associated to the application;
- retrieving the number of times the Global PIN can be incorrectly presented prior to the Global PIN being blocked;
- locking the Card Manager (i.e. the application’s ability, depending on the relevant privilege, to transition the Card Manager’s life cycle state to CM_LOCKED);
- setting the content of the historical bytes in the Answer to Reset (ATR) string. This depends on the application having the relevant privilege i.e. only available to the default selectable application;
- transitioning the application’s own life cycle state stored by the Card Manager in the registry (i.e. transitioning to life cycle states PERSONALIZED, BLOCKED and SELECTABLE);
- setting a new value for the Global PIN. This depends on the application

having the relevant privilege;

- terminating the Card Manager (i.e. the applications ability, depending on the relevant privilege, to transition the Card Manager's life cycle state to TERMINATED);
- requesting that the Card Manager verify the content of a PIN block by comparing the PIN block to the Global PIN value; and

Refer to Appendix A 'Open Platform API' for an example of a Java Card™ implementation of this interface.

6.5 Command Dispatch

Commands that are received by an Open Platform card shall either be processed by the Card Manager or dispatched to the selected application for processing.

If a SELECT [by name] command is received, the Card Manager shall determine if the application is present in the registry and available for selection. If the application is available for selection, the Card Manager shall make this application the selected application and dispatch the SELECT command to the application.

Any other type of command or SELECT commands that cannot result in an application becoming the selected application will be distributed to the selected application. It is the responsibility of the application to correctly reject commands that it does not recognize, expect or cannot process. This relates specifically to SELECT commands.

The application selection process is defined in the following section.

6.5.1 Application Selection

The Card Manager shall support application selection via two processes:

1. Implicit/Default Selection; and,
2. Explicit Selection.

The Card Manager may also support additional selection processes such as Partial Selection as defined in EMV. The following sections assume Partial Selection – Partial Selection implies that when selecting an application the off-card entity does not require knowledge of the full AID. As multiple applications on the card may have the same partial AID, it is required that a method exists to select all application matching the partial AID.

6.5.1.1 Implicit/Default Selection

Following the power on to the card and before the first command is issued to the card by an off-card entity, the Card Manager must determine if an application in the registry exists with the “default selected” privilege.
If an application with the “default selected” privilege does exist, this application becomes the selected application.
If no application with the default selectable privilege is present, the Card Manager is the selected application.

Runtime Behavior

The following is a list of behavioral requirements for the Card Manager in the implicit/default application selection process:

- If the Card Manager is in the state CM_LOCKED, the Card Manager will not attempt to identify the default selected application.
- The Card Manager shall only search for the application that is marked with the “default selected” privilege in the registry.
- The application with this privilege shall be in either the SELECTABLE, PERSONALIZED, or BLOCKED application life cycle state.
- If no application meets these criteria, then the Card Manager shall act as the default selected application.

6.5.1.2 Explicit Selection

1. If the Card Manager receives a command other than a SELECT by name command, the command is dispatched to the selected application.
2. If the Card Manager receives a SELECT [by name] command, the Card Manager searches the registry for a matching or partially matching AID entry.
3. If no matching entry is found or one or more matching entries are found but are not capable of being selected, the command is dispatched to the selected application.
4. If a matching entry is found and the application is capable of being selected, the Card Manager makes this application the selected application and the SELECT command is dispatched to this application.
5. If a subsequent SELECT [by name] command is received and select next is indicated, the Card Manager continues searching through the registry for the next partially matching AID.

Runtime Behavior

The following is a list of behavioral requirements for the Card Manager in the explicit application selection process:

- the Card Manager shall not attempt to search the registry if the Card Manager life cycle state is CM_LOCKED. (In the CM_LOCKED state, the Card manager is always the selected application and the Card Manager is responsible for responding with the correct error if an application other than itself is being selected in the CM_LOCKED state);
- the Card Manager shall only select the application that partially or fully matches the AID contained in the SELECT command;
- only applications are included in the search process. Executable Load Files are not included, as they can never transition to a SELECTABLE state;
- if a match is found, and the application's life cycle state is SELECTABLE, PERSONALIZED, or BLOCKED, this application becomes the currently selected application; and,
- matching entries in the life cycle states, INSTALLED, LOCKED or LOGICALLY_DELETED are ignored and the Card Manager skips to the next entry in the registry.

6.5.2 Application Command Dispatch

Once an application becomes the selected application, the responsibility for subsequent command dispatching still rests with the Card Manager (i.e. all commands except SELECT [by name] commands are immediately dispatched to the application). The processing of the command is the responsibility of the selected application.

Runtime Behavior

If a subsequent SELECT [by name] command is received and the Card Manager has transitioned to a CM_LOCKED state, the Card Manager shall not attempt to search the registry and the Card Manager becomes the selected application. The Card Manager is responsible for responding with the correct error if an application other than itself is being selected in the CM_LOCKED state.

If a subsequent SELECT [by name] command is received and select first/only is indicated, the normal explicit selection process is followed even if the specified AID is identical to that of the selected application. If no matching selectable AID is found, the command is dispatched to the selected application.

If a subsequent SELECT [by name] command is received and select next is indicated, the Card Manager continues searching through the registry from the current entry for the next matching or partially matching AID. If no further selectable match is found, the selected application remains selected and the Card Manager returns the appropriate error to the off-card entity. If a selectable match is found, this new application becomes the selected application.

6.6 Card Content Management

Content management on an Open Platform card includes the ability for the loading, installation, and removal of card content. This content management and control is the responsibility of the Card Manager. The following sections describe the implementation and runtime behavior requirements of the Card Manager to support these functions.

6.6.1 Content Loading and Installation

The Open Platform content loading process is designed to allow the Card Issuer or entities authorized by the Card Issuer (i.e. Application Provider) to change the content of the card's mutable persistent memory at certain points in the Open Platform Card Manager life cycle. This includes the OP_READY state and INITIALIZED state when the card is in control of the issuer and in the post-issuance phase when the card is in the SECURED state. Content loading should be prohibited in the CM_LOCKED state. All loading of new content to an Open Platform card is performed within a load process and contained in Load Files, which are used to transport application and non-application content (e.g., support libraries) to the card.

The Card Manager, and possibly Security Domains with the relevant privileges, must verify the Load File before the Load File Data Block is processed by Runtime Environment installer. The internal organization of the Load File Data Block is transparent to Open Platform. The Java Card™ CAP file definition or Windows for Smart Cards® RTE format are examples of an expected Load File Data Block. The Load File Data Block contains the information required by the installer in order to create an Executable Load File. While specific to the implementation, this process may include linking and actual verification of the executable code.
(Additional authentication data may also be present in the Load File. See Section 9.6 'LOAD command' for more detail.)

The successful completion of the above load is the presence of an Executable Load File, with an entry in the registry. However, any applications present within the Executable Load File are not yet ready for execution. Applications must be explicitly extracted and installed, resulting in another entry in the registry.

The loading and installation process can be diagrammed in two phases:

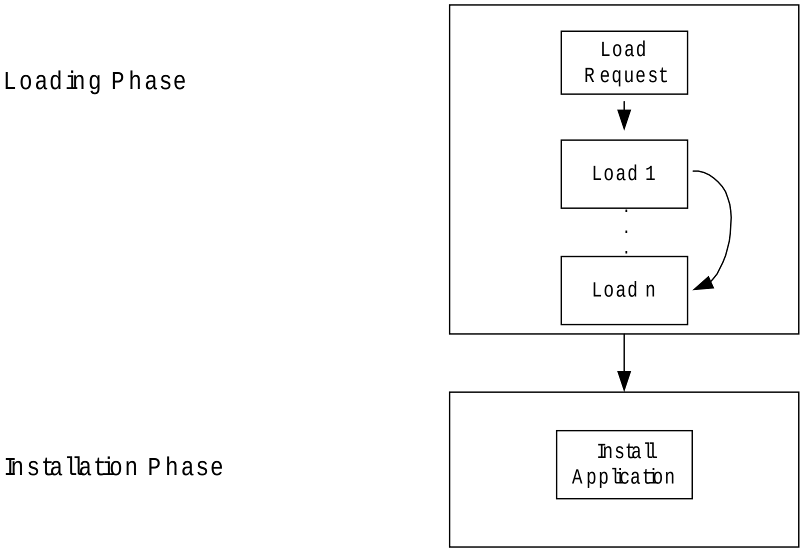


Figure 6-2: Loading and Installation process

These phases utilize a combination of two different APDU commands to accomplish the loading and installation. The INSTALL [for load] command serves as the load request to the Card Manager 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 [for install] command is then used once more following the load to instruct the Card Manager to install an application from the Load File.

The Open Platform allows for two scenarios regarding an executable application becoming present on a card:

1. 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.

2. 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.

The internal processing of installations may vary depending on the card runtime environment but is assumed to involve the creation of instances and allocation of application data memory. Following installation, the Card Manager registers information in the registry regarding application's life cycle status, associations and privileges.

Runtime Behavior (loading)

The following is a list of runtime behavior requirements for the Card Manager during the content loading process:

- On receipt of a load request (INSTALL [for load] command), the following behavior is expected.
  - check that the AID of the Load File is valid and not already present in the registry; and,
  - If an AID is present for the associated Security Domain, check that this AID exists within the registry, has the Security Domain privilege and is in a valid state.
- On receipt of the Load File (multiple LOAD commands) the following behavior is expected.
  - verify the resource requirements of Load Files;
  - check in the registry if any Security Domain requires mandatory authentication of Load File Data Blocks and if so ensure that the required authentication data is present in the Load File;
  - separate the authentication data from the Load File Data Block.; and,
  - check that the authentication data relates to Security Domains present in the registry.
- On final receipt of the Load File (last LOAD command) the following behavior is expected.
  - verify the Load File DAP received in the INSTALL [for load] command;
  - if any authentication data was present in the Load File, request the Security Domain associated with each Data Authentication Pattern (DAP) for verification;
  - request that the runtime installer create an Executable Load File using the Load File Data Block; and,
  - create an entry of the Executable Load File and its associated Security Domain in the register.

If, at any stage, the Card Manager determines that card resources are insufficient for the loading process, the Card Manager shall terminate the loading process, return the appropriate error and reclaim any memory consumed by the load process.

Runtime Behavior (Installation)

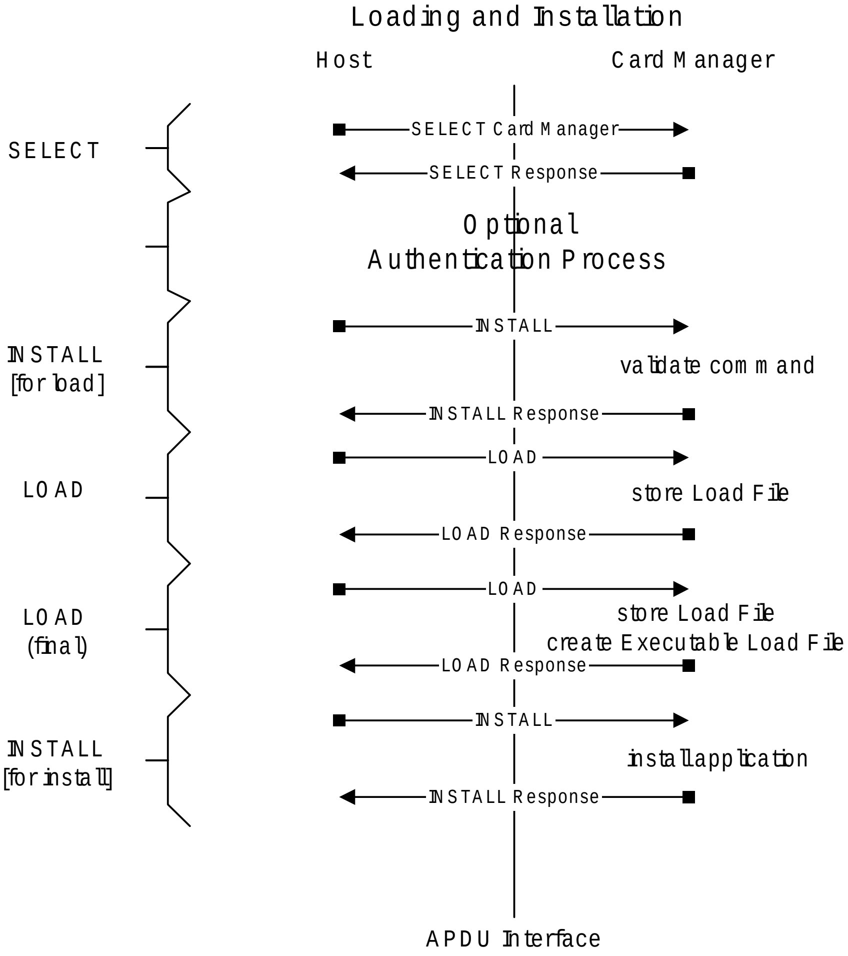
The following is a list of runtime behavior requirements for the Card Manager during the content installation process:

- On receipt of the install request (INSTALL [for install] command), the following behavior is expected.
  - check that the Executable Load File AID is present in the registry;
  - check that the application to be installed exists within the Executable Load File;
  - check that the AID with which this application will be selected is not already present in the registry;
  - verify the resource requirements of the application;
  - request that the runtime installer create an application from the Executable Load File and ensure that the application will be selectable by the specified instance AID, has knowledge its privileges and can access its application specific install parameters; and,
  - create an entry of the application and its associated Security Domain and privileges in the register.

If the Card Manager determines that card resources are insufficient for the application, the Card Manager shall terminate the installation process, return the appropriate error and reclaim any memory consumed by the install process.

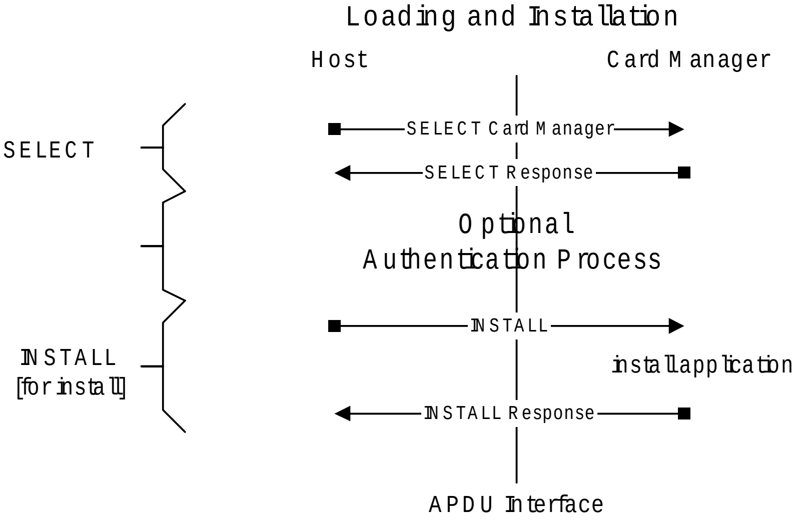
Loading and Installation Flow

The following flow is an example of the loading and installing of an application to an Open Platform card in one transaction:



Installation Flow

The following flow is an example of installing of an application to an Open Platform from an Executable Load File already present on the card:



6.6.2 Content Removal

This section defines a content removal process that enables a Card Issuer to flexibly manage the mutable persistent memory space of the cards through the removal of unwanted, expired, unused or potentially compromised applications and/or Executable Load Files. As a general rule, only data not referenced may be deleted.

The DELETE command allows for the actual removal of content from mutable persistent memory and the logical removal of content from immutable persistent memory.

Depending on the memory location of the object being removed (in immutable persistent or mutable persistent memory) the Card Manager performs different actions as described in the following sections.

6.6.2.1 Executable Load File Removal

An Executable Load File may contain applications, support libraries or both.

Runtime Behavior

The following is a list of runtime behavior requirements for the Card Manager during the Executable Load File removal process:

- determine if the Executable Load File being deleted has an entry within the registry;

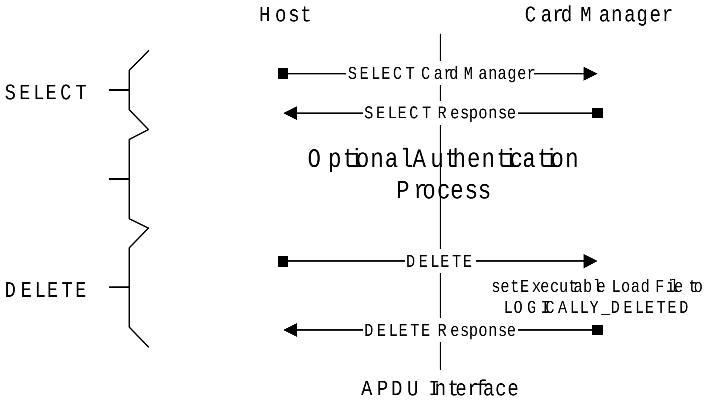
- determine if any applications or other Executable Load Files present in the card make reference to this Executable Load File;
- release and mark as available any mutable persistent memory; and,
- remove the entry within the registry.

If the Card Manager determines that the context cannot be deleted or that resources within the context being deleted are still referenced, the Card Manager shall terminate the delete process and return the appropriate response.

Deletion Flow

The following flow is an example of deleting an Executable Load File present in immutable persistent memory:

Delete Executable Load File



6.6.2.2 Application Removal

Application removal may involve the removal of application instances as well as any application data associated with the application. Physical removal may occur in mutable persistent memory while only logical removal is possible in immutable, persistent memory.

Runtime Behavior

The following is a list of runtime behavior requirements for the Card Manager during the application removal process:

- determine if the application being deleted has an entry within the registry;
- determine if any other applications present in the card make reference to this application;
- determine if any other present in the card make reference to or any data within this application;
- release and mark as available any mutable persistent memory; and,

- remove the entry within the registry.

If the Card Manager determines that the context cannot be deleted or that resources within the context being deleted are still referenced, the Card Manager shall terminate the delete process and 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 within the registry is set to the irreversible LOGICALLY_DELETED state.

6.7 Delegated Management

The Open Platform is designed with the principle of providing maximum flexibility to Card Issuers and their business partners regarding content management while ensuring that the Card Issuers retain control of the content present on cards. Delegated Management provides this ability for a Card Issuer to provide an Application Provider business partner the ability to perform specific content management.

Delegated Management is not a mandated feature of the Open Platform and is only necessary for issuers that choose to offer this flexibility. A level of cryptographic security is required for Delegated Management and a Card Manager supporting this feature must therefore have knowledge of the key(s) and algorithms used to verify a Load and Install Token and the key(s) and algorithms used to generate a Receipt DAP. In addition to the cryptographic information the Card Manager must also keep track of a confirmation counter that is incremented when generating and returning each Delegated Management Receipt.

This ability for Content Management occurs through the use of the Application Provider's Security Domain. Delegated Management is a privilege that a Security Domain must be granted during Installation. Only Security Domains with this privilege have the ability to perform:

- Delegated Loading;
- Delegated Installation; and,
- Delegated Deletion.

The Application Provider, instead of the Card Issuer, would manage each of the above processes. However, it is important to note that although the Application Provider's Security Domain would be receiving Content Management related commands, these commands are forwarded to the Card Manager who performs the physical loading and installation of the application.

For a further description of Delegated Management, see Chapter 7, 'Security Domains'.

6.8 Security Management

As described in Chapter 4, 'Security Architecture,' the Card Manager is at the very center of the security scheme, controlling access to and executing all of the most trusted services on the card. In its role as administrator of the complete card, the Card Manager must also implement a number of 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;
- Optional Global PIN management and Operational Velocity Checking for Global PIN retry counter; and,
- Optional Tracing and Event Logging.

As implied in precious sections, the Card Manager is also responsible for the security and controls regarding the loading, installing and deletion of content on the card. Even though it may not perform all these checks itself, it is responsible for insuring that the correct entity performs the check and for checking for the correct response. These functions include the following:

- checking load File DAP verification;
- checking delegated Management token verification;
- checking load File Data Block DAP verification; and,
- generating Delegated Management Receipt.

6.8.1 Global PIN Management

The Card Manager may optionally provide a mechanism for Cardholder verification that can be used by all applications on the card. Although there are a number of Cardholder verification mechanisms available today, the most widely recognized is the PIN. The Open Platform provides for the implementation of a Card Global PIN service in the Card Manager to support Cardholder verification requirements.

If Global PIN services are present in an implementation, the Card Manager must:

- reserve space for the Global PIN management;
- provide the Global PIN services to applications; and,
- manage the PIN retry limit and PIN retry counter.

6.8.2 Application Locking

With the Open Platform, a Card Issuer has a mechanism to disable the continued execution status of an on-card application. This mechanism can be invoked from within the card's runtime environment based on various exceptions handled by the Card Manager or from the use of externally invoked commands. In general, this mechanism is referred to as Application Locking and may only be performed by the Card Issuer.

Runtime Behavior

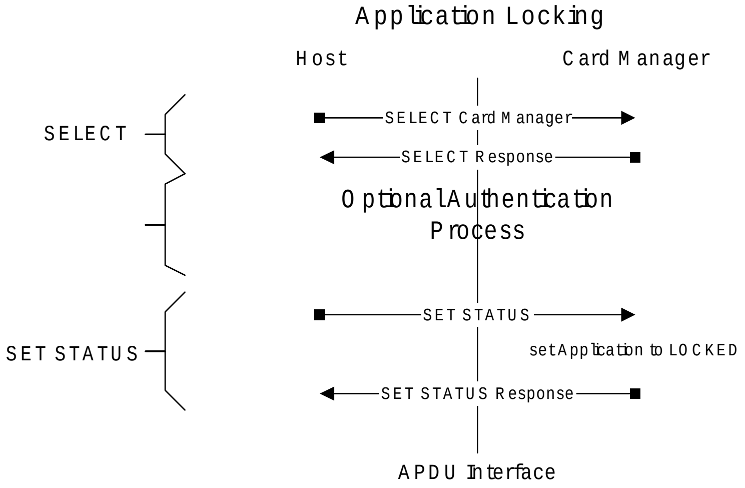
On receipt of a SET STATUS command, the following is a list of runtime behavior requirements for the Card Manager during the application locking process:

- check that an entry for the application being locked is present in the registry;
- check that the application's state is not LOGICALLY_DELETED;
- set the application's life cycle state from its current state to LOCKED; and,
- keep a record of the previous state to ensure that the application can only be transitioned back to this previous state by the Card Manager.

Once an application has been set to the LOCKED state, the application shall not be selectable. If it was the default selected application, the application is relieved of this privilege. If at a later stage the application is unlocked, the default selectable privilege does not get reinstated.

Application Locking Flow

The following flow is an example of locking an application:



6.8.3 Card Locking

Once in the CM_LOCKED state, all applications except the Card Manager are disabled. Because of the severity of this action, Card Locking must only be allowed under the most stringent conditions and shall only be performed with the proper security mechanisms and authorizations.

Card Locking requests can originate from two sources:

- Internal source— either the Card Manager or an authorized application can initiate Card Locking requests from within the card. Internal requests are likely to occur in response to certain card runtime situations. For example, the Card Manager 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.
- Off-card source—explicit Card Locking requests can be initiated by commands that are sent to the card by the Card Issuer to the Card Manager or by an authorized representative to an application with the Card Manager lock privilege.

Runtime Behavior

On receipt of a SET STATUS command or instruction from a privileged application instructing the Card Manager to lock the card, the following is a list of runtime behavior requirements for the Card Manager during the card locking process:

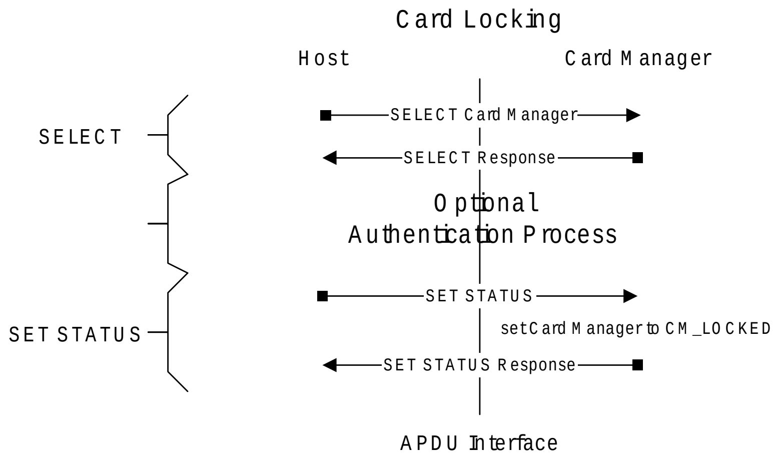
- If being requested by an application to lock the card, check within the registry that this application has the required Card Manager lock privilege;
- Set the card state to locked from its current state; and,
- keep a record of the Card Manager's previous state to ensure that the Card Manager can only be transitioned back to this previous state.

The CM_LOCKED state only applies at the completion of the current session i.e. when an application is selected or the card is reset.

As soon as the CM_LOCKED state applies, the Card Manager is the only functioning application.

Card Manager Locking Flow

The following flow is an example of locking the Card Manager:



6.8.4 Card Termination

From time to time, a Card Issuer must terminate the further use of an issued card. 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.

On an Open Platform card, Card Termination requests can originate from two sources:

- Internal source— either the Card Manager or an authorized application with the necessary privilege can initiate Card Termination requests from within the card. Internal requests are likely to occur in response to certain card runtime situations.
- Off-card source—explicit Card Termination requests can be initiated by APDU commands that are sent to the card by the Card Issuer to the Card Manager or by an authorized representative to an application with the card terminate privilege.

Runtime Behavior

On receipt of a SET STATUS command or instruction from a privileged application instructing the Card Manager to terminate the card, the following is a list of runtime behavior requirements for the Card Manager during the card termination process:

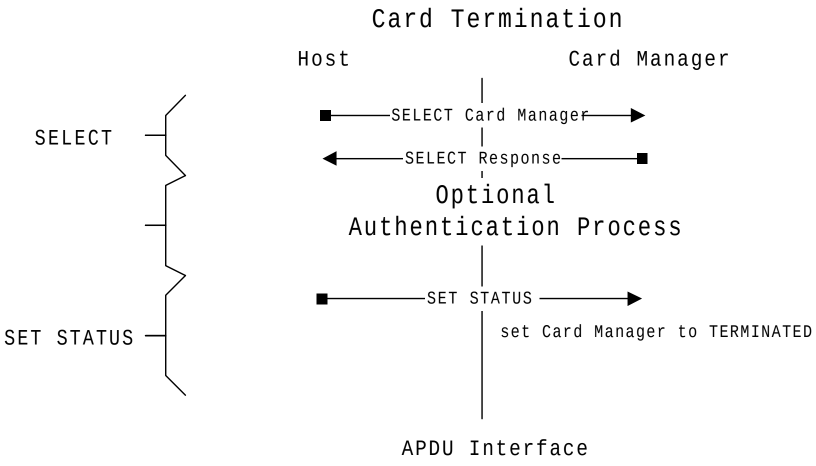
- If being requested by an application to terminate the card, check within the registry that this application has the required card terminate privilege; and

- Terminate the card.

Immediately following the Card Manager being set to the TERMINATED state, it shall no longer respond to any communication whatsoever and all card content inclusive of applications and data shall be inaccessible.

Terminate Card Flow

The following flow is an example of terminating the card:



6.8.5 Operational Velocity Checking

The Card Manager shall be implemented with a velocity checking security mechanism. Because the card does not have a Real Time Clock, the term 'velocity checking' in this specification has a different meaning than traditional velocity checking. 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 can include, but are not limited to the following:

- Content Installation; and
- Card and Application exceptions.

Typically, velocity checking is used to thwart 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.

6.8.5.1 Content Loading and Installation

The Card Manager may keep track of the number of unsuccessful consecutive attempts of the 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.

6.8.5.2 Exceptions

Velocity checking may be implemented in cases in which the Card Manager 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 Card Manager may enable velocity checking to enforce strict APDU command sequencing for card and application management operations (e.g., Content Loading and Installation). The Card Manager 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 registry. The Card Manager may choose to lock an application that is exhibiting such behavior.

6.8.6 Tracing and Event logging

Tracing and event logging functions can be enabled on an Open Platform card. These functions shall be implemented according to a Card Issuer's policy requirements, and are therefore considered extensions to the Card Manager.

6.8.7 Securing Content Loading and Installation

The following section details the security features that are available during content loading and installation and the responsibility of the Card Manager in verifying these controls are present and correct.

6.8.7.1 Load File DAP

This Data Authentication Pattern is a redundancy check across the whole Load File to be transferred to the card and is present as a field in the INSTALL [for load] command. Typically it would be a hash of the complete Load File including DAP Blocks, if any, and the Load File Data Block.

The Card Manager, on receiving the complete Load File, would perform the same function as the off-card entity and compare its own result with that received in the INSTALL [for load] command.

6.8.7.2 Tokens

Tokens relate specifically to Delegated Management and more detail is provided in section 7.8. The Card Issuer provides a token to the entity performing a Delegated Management function (i.e. the Application Provider).

During Delegated Management, commands are transferred to the Card Manager from a Security Domain that has the Delegated Management privilege. The Card Manager is the only interface to the installer and as such the sole entity responsible for the physical memory management on the card.

The Card Manager, on receiving a request from a Security Domain with Delegated Management privileges, must verify the validity of the token.

6.8.7.3 Load File Data Block DAP

A Load File may also contain one or more optional or mandatory DAP Blocks. These DAP Blocks are used for the integrity verification of the Load File Data Block. A DAP would be created by an entity other that performing the loading that wishes to ensure that only verified content is being loaded to an Open Platform card.

The verification would be performed by the Security Domain associated to each DAP. It is the responsibility of the Card Manager to request a Security Domain to verify a DAP and to act appropriately if the Security Domain cannot verify the DAP.

An optional DAP must be associated with a Security Domain that is present in the registry and verification is required in order to commit an Executable Load File to memory.

On receipt of a Load File, the Card Manager expects one or more mandatory DAP block(s) if the registry contains one or more Security Domain(s) with the mandated DAP verification privilege

6.9 Issuer Security Domain

The Card Manager is regarded as the on-card representative of the Card Issuer. And as such, the Card Manager shall contain keys that the Card Issuer will use in support of cryptographic operations for card management functions, applications, and the applications of the Card Issuer's partners if desired. As an example, these keys could be used for mutual authentication and for cryptographic operations during the card management content loading, installation and removal processes, as well as personalization. For a further description of a Security Domain and its services, please see Chapter 7, 'Security Domains'.

An Issuer Security Domain does not provide services for Delegated Management or DAP verification.

6.10 Registry

The Card Manager owns and manages information deemed necessary to perform the functionality defined by Open Platform. Exactly how this information is managed is implementation and vendor specific but for the purpose of explanation, it is assumed to be some sort of container i.e. the registry.

The registry is used to:

1. Store card management information;
2. Store relevant application management information (e.g., AID, associated Security Domain and privileges);
3. Support card resource management (e.g., non-volatile memory allocation);
4. Store application and Card Manager life cycle information;
5. Manage the Card Global PIN;
6. Manage the Card Manager (Card Issuer's) keys;
7. Track any counters associated with velocity checking; and
8. Track any counters associated with Delegated Management Receipt generation.

The contents of the registry may be updated in response to:

1. A Card Issuer invoked action;
2. An internal Card Manager invoked action; and
3. An authorized application invoked action.

There are two sets of data elements, one set for Card Manager, and the other set for all applications, including Security Domains. 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.

6.10.1 Card Manager Data Element Description

As the Card Manager has its own AID and Life Cycle State, it may be decided by the implementers of an Open Platform card to store Card Manager information in the same registry as application information. This specification neither suggests nor prohibits this one way or another.

If the Card Manager is stored in the registry, it must be noted that its status cannot be visible to an off-card entity as an application. As noted in the section 9.4 'GET STATUS command' a separate indicator is provided for an off-card entity to retrieve the Card Manager status.

The following sections describe the possible registry data elements for the Card Manager.

6.10.1.1 Card Manager AID

The Card Manager is associated with an AID that must be unique among the applications or Executable Load Files on the card. This AID may be specified in a SELECT command in order for the Card Manager to be the selected application.

The Card Issuer should be able to define and set its own value for the Card Manager AID.

While in the early stages of a card's life, the Card Manager is typically the default selected application, at later stages in the cards life, another application may become the default selected application. In this case, to make the Card Manager easily accessible to off-card entities (e.g. back-end card management system), a global AID is currently defined by Visa International:

- A0 00 00 00 03 00 00.

6.10.1.2 Card Manager Life Cycle State

The Card Manager life cycle state data element contains the current life cycle state of the Card Manager.

The Card Manager life cycle states visible to off-card entities are:

1. OP_READY
2. INITIALIZED
3. SECURED
4. CM_LOCKED

6.10.2 Application/Executable Load File Data Elements

There are a mandatory number of data elements associated with each Application and Executable Load File that the Card Manager must store in the registry.

The following sections describe the registry data elements for Applications/Executable Load Files.

6.10.2.1 Application/Executable Load File AID

Each Executable Load File or application is associated with an AID that must be unique among the Applications or Executable Load Files on the card.

The Application AID may be specified in a SELECT command in order for an Application to become the selected application.

The Executable Load File AID is specified to identify the file from which an application is to be installed.

6.10.2.2 Application/Executable Load File Life Cycle State

The Application/Executable Load File life cycle state data element contains the current life cycle state of the Application/Executable Load File.

The Executable Load File life cycle states are:

1. LOGICALLY_DELETED
2. LOADED

The Application life cycle states are:

1. LOGICALLY_DELETED
2. INSTALLED
3. SELECTABLE
4. PERSONALIZED
5. BLOCKED
6. LOCKED

6.10.2.3 Resource Allocation

The Resource Allocation data element contains a value for the total amount of resources that are available to an application. It is a system-specific value and is used as a control mechanism by the Card Manager to limit the amount of resources that an application may claim during runtime.

When additional resources are requested by an application, the Card Manager must validate against the value of this data element in the registry.

Runtime Behavior

The following is a list of behavioral requirements for the Card Manager in the card resource allocation process:

- If the additional resource requested by an application exceeds its allocation limit, the Card Manager shall terminate processing of the offending application and return an appropriate response code.
- The Card Manager may choose to lock an application that makes repeated attempts to allocate additional resources beyond its allocation limit.

6.10.2.4 Application Privileges

The Application Privileges data element indicates the privileges for each application. The Application Privileges are as follows:

1. Application is a Security Domain without Delegated Management privilege.
2. Application is a Security Domain with DAP verification privileges.
3. Application is a Security Domain with Delegated Management privilege.
4. Application has Card Manager lock privilege.
5. Application has card terminate privilege.
6. Application is the default selected application.
7. Application has privilege to change the Card Global PIN.
8. Application is a Security Domain and mandates the presence of a DAP in all Load Files.

The following rules apply to the assignment of Application Privileges:

- each Security Domain may only be set to a single privilege level regarding Delegated Management privilege (i.e., with or without);
- only one application may be set with the “default selected” privilege at a time; and,
- Mandatory DAP verification overrides DAP verification privilege. (Only one level of DAP verification can be set.)

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

Runtime Behavior

The Card Manager uses the Application Privileges data element for the controlling the following runtime behavioral requirements:

- mandating a DAP 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 Delegated Management privilege;
- checking for the validity of Card Manager lock requests;
- checking for the validity of card termination requests;
- supporting implicit application selection (At any given time, the Card Manager ensures that only one application, including itself, is marked as the default selected application.);

- checking for the validity of change ATR historical byte requests (only accessible to the default selected application); and,
- checking for the validity of requests to change the Global PIN.

6.10.2.5 Associated Security Domain

The associated Security Domain data element contains the AID of an application's associated Security Domain. Applications have the choice to use certain services of their associated Security Domains.

This association is initiated when the Load File from which the application is installed is loaded to the card. The INSTALL [for load] command specifies the associated Security Domain that is linked to the Executable Load File and this link is retained when the application is installed. If no Security Domain is specified in the INSTALL [for load] command, the Card Manager is assumed to be the associated Security Domain.

Runtime Behavior

When the Card Manager receives a request from an application to use a service of the application's associated Security Domain the Card Manager must exhibit the following runtime behavioral requirements:

- locate the application's entry in the registry and retrieve the associated Security Domain;
- if no associated Security Domain is present the services of the Issuer Security Domain must be used; or,
- if an associated Security Domain is present, the entry for the associated Security Domain must be located and the applications request for service forwarded to this Security Domain.

The associated Security Domain must be in an accessible life cycle state (i.e. PERSONALIZED, BLOCKED or LOCKED) in order for its services to be usable.

6.11 Card Manager APDU Command Summary

The following table lists all the APDU commands that are required to support a Card Manager implementation

Table 6-1: Card Manager APDU Interface

Command	**Description**
SELECT	Selects an application
SET STATUS	Sets card life cycle or application life cycle states
GET STATUS	Retrieves card or application life cycle status
GET DATA	Retrieves Card Manager data from the card
INSTALL	Prepares application installation
LOAD	Loads File into the card

PUT KEY	Loads a key or keys to the Card Manager
DELETE	Deletes an application or Executable Load File

See Chapter 9, 'APDU Command Reference' for a detailed description of the APDU commands.

6.12 Card Manager Data

Besides the registry and any other implementation specific information required by the Card Manager to perform its required functionality, the Card Manager should be able to store and access the Card Issuer Identifier and the Card Production Life Cycle Data (CPLC). This information and any other data retrievable with the Open Platform defined GET DATA command should be available off-card without the general requirement for mutual authentication between the card and the off-card entity requesting the information.

6.12.1 CPLC Data

The Card Production Life Cycle may be used by an off-card entity to track the life of the card and also to determine the versions and features of each card.

The CPLC data typically contains the whom, when and what identification for the following:

- The integrated circuit (IC) fabrication;
- The Operating System;
- The integrated circuit card (ICC) manufacturer.;
- Module fabrication;
- Embedding;
- Pre-personalization; and;
- Personalization.

6.12.2 Card Issuer Identifier

The Card Issuer ID may be used by an off-card entity to associate the card to a specific card management system. The Card Issuer ID typically contains the ISO 7812 defined identification of the Card Issuer.

7. Security Domains

7.1 Overview

As explained in Chapter 6 ‘Card Manager’ the Card Manager (in the context of an Issuer Security Domain) contains keys that the Card Issuer uses in support of cryptographic operations for the Card Issuer's applications. While it is possible to utilize the Card Issuer keys in conjunction with applications that are not owned by the Card Issuer, it is not necessarily practical, nor recommended.

A Security Domain is a special type of application which is responsible for managing and providing cryptographic services for keys that are completely separate from, and not under the control of, the Card Issuer. Security Domains are privileged applications established on an Open Platform card to represent Application Providers who require some level of key separation from the Card Issuer.

All Security Domains contain some keys capable of supporting applications during personalization. Security Domain keys may also be used in support of runtime cryptography for applications if desired.

A Security Domain may support three additional privileges that are referred to as DAP verification, mandated DAP verification and Delegated Management. A Security Domain may support one or all of these additional privileges although mandated DAP verification would obviously supersede DAP verification and as such only one is required.

Mandated DAP verification allows a controlling body to own a Security Domain that is always used to verify integrity during loading. This ensures that only applications authorized by the controlling body may be loaded on cards that contain this Security Domain.

DAP verification allows an Application Provider to own a Security Domain which can be used to verify the integrity its applications during loading especially when the loading is carried out by an entity other than the Application Provider itself.

Delegated Management allows Application Providers to load and install their own applications with pre-authorization from the Card Issuer and to delete these applications and Executable Load Files at a later stage.

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

- Personalization Support: Secure communication support during personalization of an Application Provider's applications; and,
- Runtime Messaging Support. Provides secure communication support during runtime for an application that does not contain messaging keys.

A similar set of keys and algorithms must also be available to the Card Manager in its role as the Issuer Security Domain.

Security Domains with DAP verification privilege are used for:

- Verifying a Load File DAP (a specific key within the Security Domain must be identified as a DAP verification key);

Security Domains with Delegated Management privilege can be used for:

- Delegated Loading;
- Delegated Installation; and,
- Delegated Deletion.

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

An Issuer Security Domain is required on all Open Platform cards to manage the Card Issuer keys and to provide optional cryptographic services to applications.

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

A Security Domain with DAP verification privilege is required if an Application Provider wishes to guarantee the integrity of its application being loaded by another entity.

A Security Domain with mandatory DAP verification privilege is required if a controlling body wishes to guarantee the integrity of any application being loaded to the card.

If an Application Provider's application is to be loaded, installed, or deleted from the card by the Application Provider once the Card Manager state is set to SECURED, a Security Domain with Delegated Management privilege is required.

7.1.1 Security Domain Type

During application installation, bits indicating that the application is a Security Domain are included in the INSTALL command. This information is recorded in the Application Privileges data element of the registry. See section 6.10 'Registry' for a description of the data elements.

7.2 Security Domain Life Cycle

Security Domains are expected to follow the application life cycle model as described in Chapter 5, 'Life Cycle Models'. As an application, a Security Domain has the same behavior as an application throughout its application life cycles. As a provider of security services to application associated with them, the runtime behavior of Security Domains during their life cycle states is described in this section.

INSTALLED

Security Domains are installed on the card via the Card Manager, which sets their initial life cycle state to INSTALLED. Security Domains, in this state, are not available for selection, they cannot be associated with applications yet and therefore their Security Domain services are not available to applications.

SELECTABLE

The Card Manager sets Security Domains to the state SELECTABLE in order to enable personalization. As they do not yet have keys, the Security Domains still cannot be associated with applications and therefore their services are not available to applications when they are in this state.

PERSONALIZED

Once a Security Domain has been personalized with its keys and other necessary security elements, it sets its own state to PERSONALIZED through the Card Manager. In this state the Security Domain can be associated with applications and its services become available to these associated applications.

BLOCKED

A Security Domain has the option to block itself as a protection mechanism against any threat. Blocking a Security Domain does not have any required effect on the access to that Security Domain's services by applications through the Card Manager. However, Security Domains do have the option to modify their own behavior to prohibit or restrict their services while blocked.

LOCKED

The Card Manager always has the option to prevent the selection of any application on the card including Security Domains by locking the application. (This would prevent the Security Domain being used for Delegated Management if applicable). Locking a Security Domain prevents this Security Domain from being associated with new Load Files but does not have any required effect on the access to that Security Domain's services by applications through the Card Manager. However, Security Domains do have the option to modify their own behavior to prohibit or restrict their services while locked.

Issuer Security Domain

The Issuer Security Domain, incorporated in the Card Manager, has different behavior related to the Security Domain services available to applications. The Issuer Security Domain can be the associated Security Domain to applications in any stage of the Card Manager's life cycle.

7.3 Application Access to Security Domain Services

Applications have the ability to access Security Domain services through the Card Manager/Runtime Environment. By using these services, the application can rely on cryptographic support from the Security Domain to ensure confidentiality and integrity during personalization, and optionally, during runtime as well.

The Security Domain required services are generic and do not vary regardless of the Security Domain owner, or the cryptographic capabilities of the card.

The manner in which this access is provided is platform specific but must encompass the possibility of:

- initiating a Secure Channel or Mutual Authentication;
- verifying an off-card entity or completing Mutual Authentication. Successful verification of an off-card entity implies the creation of a Secure Channel;
- unwrapping a command received within a Secure Channel (verifying the integrity and/or obtaining the original data in the case of confidentiality);
- decrypting and verifying a key block within a Secure Channel; and,
- destroying any secrets created by the act of Mutual Authentication or setting up a Secure Channel.

Refer to Appendix A 'Open Platform API' for an example of a Java Card™ implementation of this interface.

7.4 Secure Communication

The Open Platform architecture is designed to enable secure communication between the Card Issuer and Card Manager, between Application Providers and their Security Domains, and between Application Providers and their applications. The commands were designed to support both integrity and confidentiality at the command level but they do not require a specific form of cryptography.

It is assumed that one-way or mutual authentication may be required to establish a secure communication session between the various on-card and off-card entities; however, the Open Platform does not require that this process take place within the context of each transaction. The requirement and format of this authentication process is considered industry and product specific.

7.5 Personalization Support

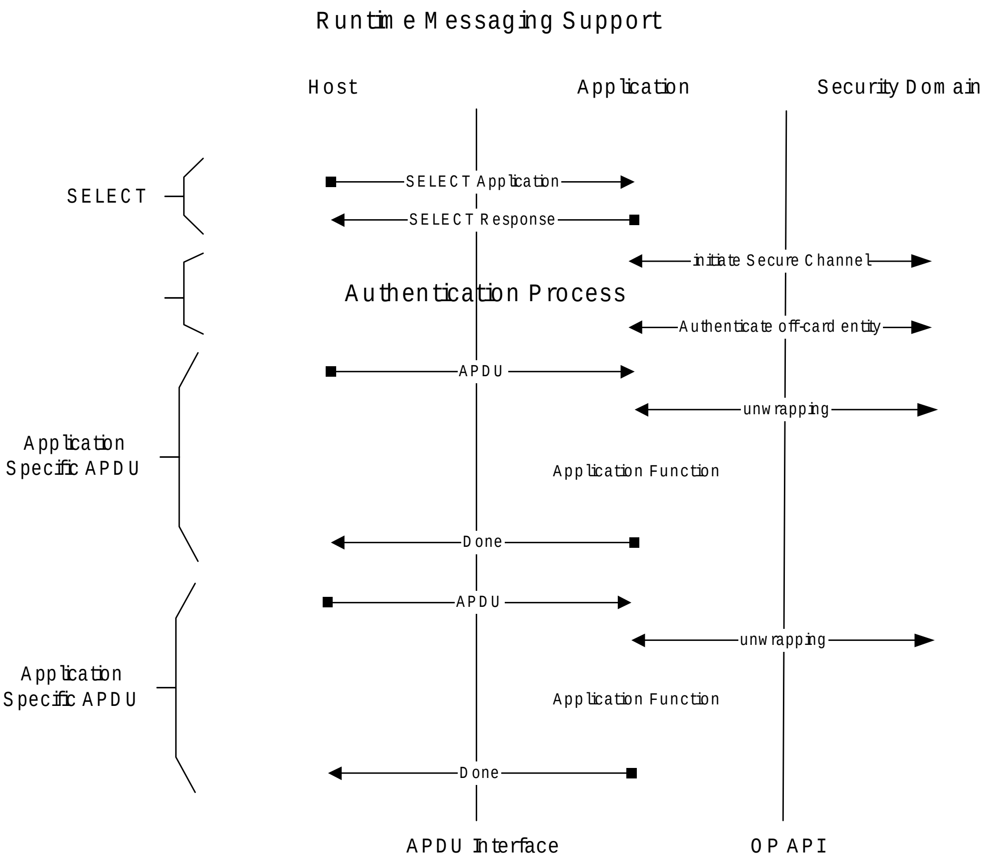
After an application is installed, the application may need to obtain its personalization data, including its own keys and Cardholder-specific data. The application can utilize the secure communication and key decryption services of its associated Security Domain to manage the secure loading of this personalization data.

7.6 Runtime Messaging Support

Security Domains may provide runtime support for secure communication to their applications. In this scenario, instead of loading additional communication keys into an application, Application Providers may choose to utilize their associated Security Domain's services for all application communication throughout the life of their application(s).

Runtime messaging Flow

The following flow is an example of an application using the services of it's associated Security Domain:



7.7 DAP Verification

The Open Platform allows for both the Card Issuer and Application Providers to load their own applications. The process for Application Providers loading their own applications is referred to as Delegated Management and is described in Section 7.8, 'Delegated Management'. In some instances it may not be possible for an Application Provider to load its own application and therefore the Application Provider would prefer to rely on the loading services of the Card Issuer or a third party who has loading privilege on the Card Issuer's cards.

In other instances, a controlling body may require that all applications to be loaded to cards under the controlling body's influence, must be certified by the controlling body.

In these situations the Application Provider may require a check or the controlling body may mandate a check of the application's integrity and authenticity before the application is loaded, installed and available to the cardholder. The DAP verification privilege for a Security Domain was created to provide this service on behalf of an Application Provider and the mandated DAP verification privilege was created to provide this service on behalf of a controlling body.

The requirement for DAP verification of a Load File Data Block is determined by the optional or mandated presence of one or more DAP blocks in the Load File. The Card Manager is required to check for the presence of these blocks and sequentially verify each DAP object prior to completing the loading process. While the Card Manager is responsible for checking each DAP present in the Load File, it does not have the cryptographic keys or knowledge of the algorithm required to perform the actual verification. The Card Manager therefore sends the required information to the identified Security Domain and it is this Security Domain's responsibility to inform the Card Manager whether the DAP is valid or not.

Security Domains with DAP privileges must have knowledge of the key(s) required to verify a Load File Data Block DAP.

7.8 Delegated Management

The philosophy behind the Open Platform architecture is to ensure the Card Issuer has complete control over the cards at all times. However, the design of the Open Platform also takes into account the possibility that the Card Issuer may not necessarily want to transactionally manage all card content changes to the cards, especially when the content does not belong to the Card Issuer.

The concept of Delegated Management is incorporated into Open Platform and allowing the Card Issuer the possibility of empowering partnered Application Providers the ability to initiate pre-approved changes to the card.

This pre-approval, which is central to the concept of Delegated Management, ensures that only card content changes that the Card Issuer has approved will be accepted and processed by the Card Manager. This delegation of some control in the card update process can allow Application Providers more flexibility in managing their own applications on the Card Issuer's cards.

Delegated Management supports all of the following functions:

- Delegated Loading
- Delegated Installation
- Delegated Deletion

7.8.1 Delegated Loading and Delegated Installation

Delegated Loading allows Application Providers to establish a secure session, for transferring their Load Files directly to their Security Domains.

Once each APDU has been securely transferred to the card, the Security Domain passes it to the Card Manager for any additional verification and processing. This process is comprised of one INSTALL [for load] command and multiple LOAD commands.

Runtime Behavior

The following is a list of additional runtime behavior requirements for the Card Manager during the Delegated Loading process:

- check that the Security Domain performing the load has the Delegated Management privilege;
- check that the Security Domain performing the load is the same Security Domain that will be associated with the Executable Load File;
- verify the Load DAP (Load Token present in the INSTALL [for load] command) generated by the Card Issuer;
- on receipt of the complete load file verify the Load File DAP; and,
- on completion of the load procedure a Load Receipt must be generated and returned to the Security Domain performing the Delegated Management function.

Providing a Load Token implies that the Card Issuer must have pre-authorized the data required for the delegated load process. The data to pre-authorize comprises most of the INSTALL [for load] command and includes the Load File DAP. The inclusion of the Load File DAP in the Load Token is what links the Token to the actual Load File.
The Load Token therefore allows the Card Manager to ensure that the Card Issuer authorized the load process and Load File.

Delegated Installation allows Application Providers to establish a secure installation session in order to request that an application already present in an Executable Load File within the card be installed. Once the INSTALL [for install] command has been securely transferred to the card, the Security Domain passes it to the Card Manager for any additional verification and processing.

Runtime Behavior

The following is a list of additional runtime behavior requirements for the Card Manager during the Delegated Installation process:

- check that the Security Domain performing the install has the Delegated Management privilege;
- check that the Security Domain performing the install is the same Security Domain associated with the Executable Load File from which the application is being installed;
- verify the Install DAP (Install Token present in the INSTALL [for install] command) generated by the Card Issuer; and,

- on completion of the install procedure an Install Receipt must be generated and returned to the Security Domain performing the Delegated Management function.

Providing an Install Token implies that the Card Issuer must have pre-authorized the data required for the delegated installation process. The data to pre-authorize comprises most of the INSTALL [for install] command.

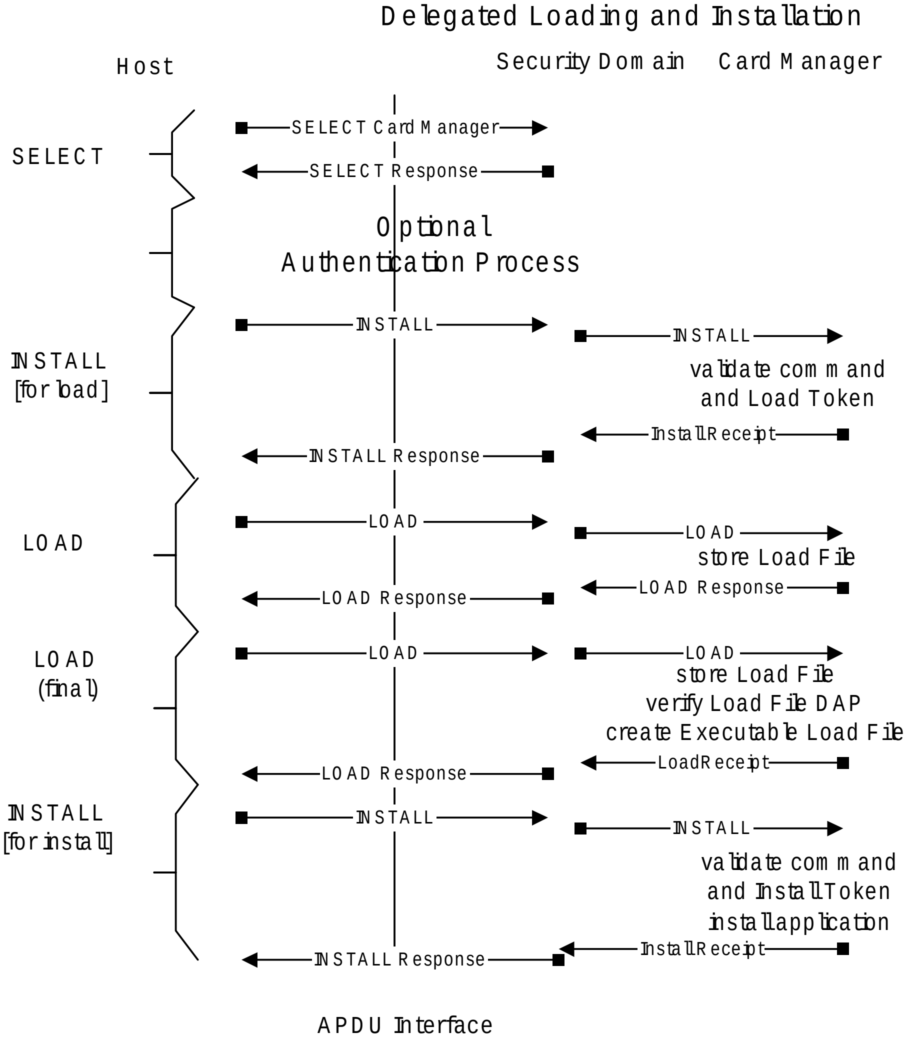
The Install Token therefore allows the Card Manager to ensure that the Card Issuer authorized the insall process.

Each Load or Install Receipt, including the DAP, is returned to the Security Domain performing the Delegated Management operation and must be transmitted to the off-card entity.
(While the Install Receipt is transmitted off-card as the response to the INSTALL [for install] command, the Load Receipt is only transmitted off-card following the completion of the load process i.e. as a response to the last LOAD command.)
The Application Provider, as a way to provide proof that the operation was successfully performed, is expected to forward these Receipts to the Card Issuer. The purpose of these Receipts is to assist the Card Issuer in keeping their card management systems synchronized with their card base.

The processes of Delegated Loading and Delegated Installation may occur in a single transaction where an application is loaded and then installed immediately. Alternatively, the Delegated Loading and Delegated Installation processes may occur independently of one another.

Delegated Loading and Installation Flow

The following flow is an example of loading and installing an application on the card through a Security Domain with the Delegated Management privilege:



7.8.2 Delegated Deletion

Application Providers have the ability to instruct the Card Manager to delete applications that they own. The Card Manager will honor these requests without authorization from the Card Issuer.

Once the DELETE command has been securely transferred to the card, the Security Domain passes it to the Card Manager for any additional verification and processing.

Runtime Behavior

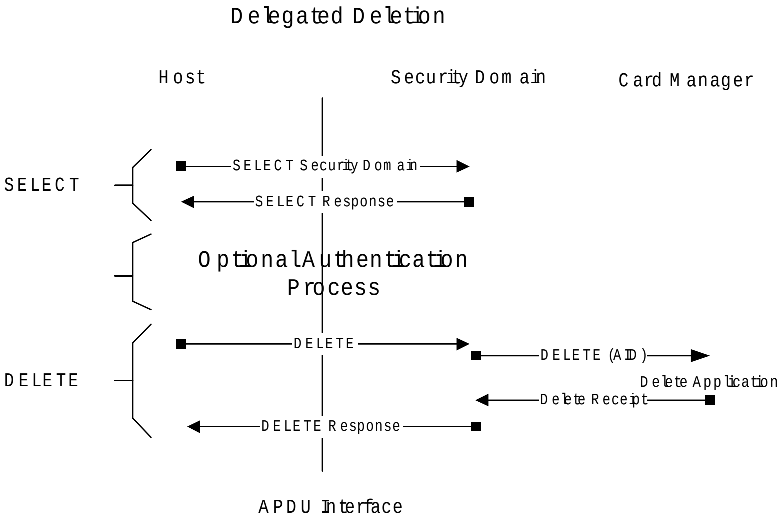
The following is a list of additional runtime behavior requirements for the Card Manager during the Delegated Deletion process:

- check that the Security Domain performing the load has the Delegated Management privilege;
- check that the Security Domain performing the delete is the same Security Domain associated with the Executable Load File or application being deleted.

The Card Manager carries out the deletion and then generates a Receipt DAP confirming the successful deletion. The receipt, including the DAP, is returned to the Security Domain performing the Delegated Deletion and must be transmitted to the off-card entity. The Application Provider, as a way to provide proof that the operation was successfully performed, is expected to forward the Delete Receipt to the Card Issuer. The purpose of this Receipt is to assist the Card Issuer in keeping their card management systems synchronized with their card base.

Delegated Deletion Flow

The following flow is an example of deleting an application on the card through a Security Domain with the Delegated Management privilege:



7.9 Delegated Management Tokens and Receipts

This section further describes the information contained in Tokens and Receipts.

The cryptography required for the generation of both Tokens and Receipts is considered industry and product specific and thus is not specified. The data elements and format of the Tokens and Receipts as well as their positioning within the APDU command/response pairs is specified in this section.

7.9.1 Load Token

The Load Token allows the Card Manager to verify the load process prior to actually processing the INSTALL [for load] command.
Note that at this point the Card Manager cannot verify the content of the Load File as it has not yet been received. Once the complete Load File has been received, the Card Manager must verify that the Load File DAP is valid and, if not, must abort the load process at this stage.

The Load Token is a DAP of the following fields within an INSTALL [for load] command and is appended to the INSTALL [for load] command.

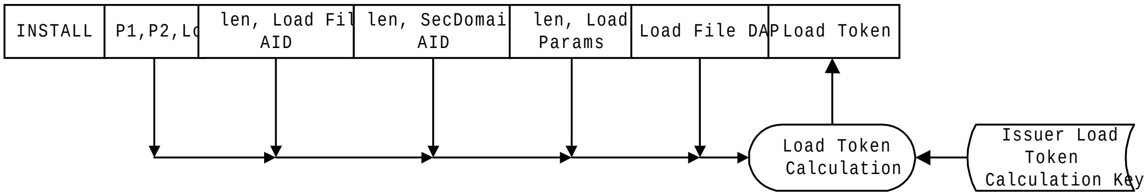


Figure 7-1: Load Token calculation

The Load File DAP is included in the calculation to ensure that the Load File which is approved by the Load Token is the same Load File that is subsequently received by the Card Manager through the series of LOAD commands that follow the INSTALL [for load] command.

7.9.2 Load Receipt

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 transaction including card unique data and a Receipt DAP generated by the Card Manager. The Load Receipt is the response message for the last LOAD command issued in a sequence of LOAD commands to the Security Domain. The Receipt DAP is calculated by the Card Manager using the following data:

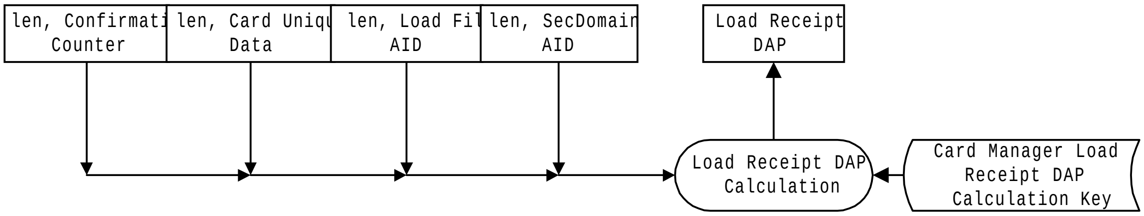


Figure 7-2: Calculation of Load Receipt DAP

The Load Receipt DAP is concatenated with the Confirmation Counter and Card Unique Data and returned as the response message for the last LOAD command issued to the Security Domain.

len, Load Receipt DAP	len, Confirmat Counter	len, Card Identification	Data

Figure 7-3: Structure of Load Receipt

7.9.3 Install Token

The Install Token allows the Card Manager to verify the install process prior to actually processing the INSTALL [for install] command.
The Install Token is a DAP of the following fields within an INSTALL [for install] command and is appended to the INSTALL [for install] command.

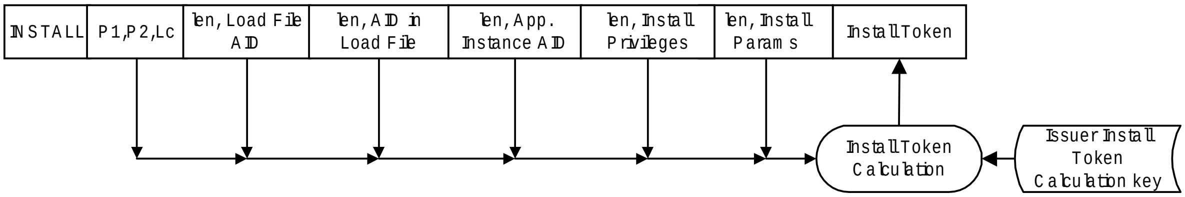


Figure 7-4: Install Token calculation

7.9.4 Install Receipt

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 transaction including card identification data and a Receipt DAP generated by the Card Manager. The Install Receipt is returned in the response message to the INSTALL [for install] command sent to the Security Domain. The Receipt DAP is calculated by the Card Manager using the following data:

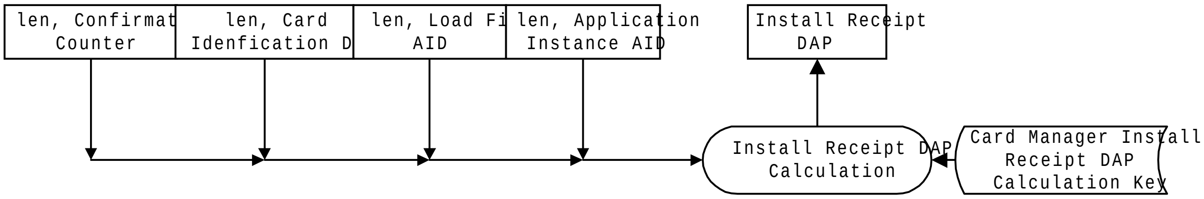

len, Confirmat Counter
len, Card Idenfication D
len, Load Fi AID
len, Application Instance AID
Install Receipt DAP
Install Receipt DAP Calculation
Card Manager Install Receipt DAP Calculation Key

Figure 7-5: Calculation of Install Receipt DAP

The Install Receipt DAP is concatenated with the Confirmation Counter and Card Unique Data to be returned as the response message for the INSTALL [for install] command issued to the Security Domain.

len, Install Receipt DAP	len, Confirmat Counter	len, Card Identification Data

Figure 7-6: Structure of Install Receipt

7.9.5 Delete Receipt

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 transaction including card unique data and a Delete DAP generated by the Card Manager. The Delete Receipt is returned in the response message to the DELETE command issued to the Security Domain. The Receipt DAP is calculated by the Card Manager using the following data:

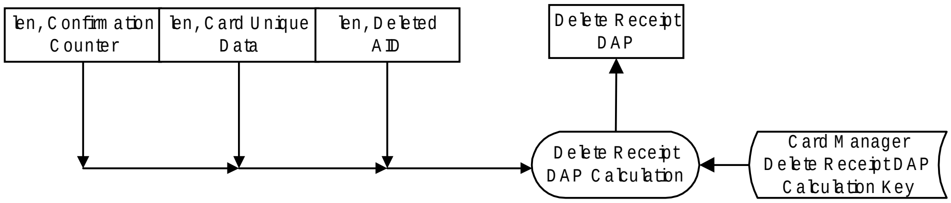

len, Confirmation Counter
len, Card Unique Data
len, Deleted AID
Delete Receipt DAP
Delete Receipt DAP Calculation
Card Manager Delete Receipt DAP Calculation Key

Figure 7-7: Calculation of Delete Receipt DAP

The Delete Receipt DAP is concatenated with the Confirmation Counter and Card Unique Data to be returned as the response message for the DELETE command issued to the Security Domain.

len, Delete Receipt DAP	len, Confirmat Counter	len, Card Identification	Data

Figure 7-8: Structure of Delete Receipt

7.10 Security Domain APDU Command Summary

Security Domains must implement a required set of APDU commands.

Table 7-2 lists the APDU commands that Security Domains **with** Delegated Management privilege must implement.

Table 7-1: Security Domain **with** Delegated Management APDU Interface

Command	**Description**
SELECT	Selects an application
INSTALL	Prepares application installation
LOAD	Loads files into the card
DELETE	Deletes an application or Load File

Table 7-3 lists the APDU commands that Security Domains **without** Delegated Management privilege must implement.

Table 7-3: Security Domain **without** Delegated Management APDU Interface

Command	**Description**
SELECT	Selects an application

See Chapter 9, ‘APDU Command Reference,’ for a full description of the APDU commands.

7.10.1 Installing Security Domains

As the interface between a Security Domain and the Card Manager is not defined by Open Platform (specifically relating to Delegated Management and DAP verification), it is assumed 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.

8. Open Platform API

For an example of an implementation of the Application Programming Interface (API) on a Java Card™, see appendix A

For an example of an implementation of the Application Programming Interface (API) on a Windows for Smart Card®, see appendix B

This page intentionally left blank.

Part IV
APDU
Command
Reference

This page intentionally left blank.

9. APDU Command Reference

This chapter contains Open Platform APDU command reference information. All the commands are listed alphabetically.

Table 9-1 shows a summary of the commands listing which are required for implementation in Card Manager and Security Domains.

Table 9-1: Open Platform commands

CLA	INS	Command	Card Manager	Security Domain with Delegated Management	Security Domain with DAP or mandated DAP Verification	Security Domain w/out Delegated Management
'80' or '84'	'E4'	DELETE	✔	✔		
'80' or '84'	'CA'	GET DATA	✔			
'80' or '84'	'F2'	GET STATUS	✔			
'80' or '84'	'E6'	INSTALL	✔	✔		
'80' or '84'	'E8'	LOAD	✔	✔		
‘80’ or ‘84’	‘D8’	PUT KEY	Optional	Optional	Optional	Optional
'00'	'A4'	SELECT	✔	✔	✔	✔
'80' or '84'	'F0'	SET STATUS	✔	Optional	Optional	Optional

9.1 General Coding Rules

9.1.1 Life Cycle Status Coding

The Executable Load File life cycle status 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	0	LOGICALLY_DELETED
0	0	0	0	0	0	0	1	LOADED

The Application life cycle status 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	0	LOGICALLY_DELETED
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
0	1	1	1	1	1	1	1	BLOCKED
1	1	1	1	1	1	1	1	LOCKED

Executable Load Files and Applications cannot be set to more than one life cycle status value at a time. The allowed Application life cycle state transitions are described in Chapter 5, 'Life Cycle Models'.

The Card Manager life cycle status value 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	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	CM_LOCKED
1	1	1	1	1	1	1	1	TERMINATED

The Card Manager cannot be set to more than one life cycle status value at a time. The allowed Card Manager life cycle state 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:

b8	b7	B6	b5	b4	b3	b2	b1	Meaning
1								Security Domain
	1							DAP verification
		1						Delegated Management
			1					Card Manager lock privilege
				1				Card terminate privilege
					1			default selected
						1		PIN Change privilege
							1	Mandated DAP verification

An application may support one or more of these Application Privileges.

The individual bits indicate the follows:

- b8=1 indicates that the application is a Security Domain without Delegated Management privilege. (Mutually exclusive with b6).
- b7=1 indicates that the application is a Security Domain with DAP verification capability. (Mutually exclusive with b1).

- b6=1 indicates that the application is a Security Domain with Delegated Management privilege. (Mutually exclusive with b8).
- b5=1 indicates that the application has Card Manager locking privilege.
- b4=1 indicates that the application has Card terminate privilege.
- b3=1 indicates that the application has default selection privilege.
- b2=1 indicates that the application has PIN Change privilege.
- b1=1 indicates that the application is a Security Domain with mandated DAP verification capabilities. (Mutually exclusive with b7).

9.1.3 General Error Conditions

The following error conditions may be returned by any of the commands described here:

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

9.1.4 Class Byte Coding

The class byte of all Open Platform commands conform to the ISO 7816-4 standards and are coded according to the following table:

CLASS	Meaning
'00'	ISO type command
'80'	Proprietary command
'84'	Proprietary command with secure messaging

9.2 DELETE command

9.2.1 Definition and Scope

The DELETE command is used to delete a uniquely identifiable object. The object may be an Executable Load File, Application or some other item; however, in order to delete an object, the object must be uniquely identifiable by the selected application.

9.2.2 Command Message

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

Code	Value	Meaning
CLA	'80' or '84'	Open Platform command
INS	'E4'	DELETE
P1	'xx'	Reference control parameter P1
P2	'00'	always '00'
Lc	'xx'	Length of data field
Data	'xxxx…'	TLV coded object identifier (and message DAP if present)
Le	'00'	

9.2.2.1 Reference Control Parameter P1

Reference control parameter P1 is used to indicate that additional DELETE commands are to follow, or that the last DELETE command has been reached.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0								Indicate last DELETE
1								Indicate more DELETE commands to follow
	0	0	0	0	0	0	0	RFU

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 shall contain the TLV coded name(s) of the object(s) to be deleted.

Deleting an Executable Load File or Application shall be specified using the ISO Tag for Application Identifier (AID), ‘4F’.

The following table shows the general structure of the DELETE command message:

Presence	Length (no. of bytes)	Name
Mandatory	1-2	Tag
Mandatory	1	Length of object identifier
Mandatory	1-n	Object identifier

9.2.3 Response Message

A response message is always returned but only relevant in the case of Delegated Management. In the case where the Card Manager is processing the DELETE, a single byte of ‘00’ will be returned indicating that no additional response data is present.

9.2.3.1 Data Field Returned in the Response Message

The data field of the response message contains the result of the delete procedure. Delete receipts are mandatory for Delegated Management.

Presence	Length (no. of bytes)	Name
Mandatory	1	Length of Delete Receipt
Conditional	0-n	Delete Receipt

Format of the Delete Receipt used for Delegated Deletion.

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

9.2.3.2 Processing State Returned in the Response Message

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

The ICC may return the following warning conditions:

SW1	SW2	Meaning
'62'	'00'	Application has been logically deleted

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return 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

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.

9.3.2 Command Message

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

Code	Value	Meaning
CLA	'80' or '84'	Open Platform command
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 message DAP
Data	'xxxx…'	Message DAP if present
Le	'00'	

9.3.2.1 Parameter P1 and P2

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

9.3.2.2 Data Field Sent in the Command Message

The data field of the command message is empty unless a message DAP is required.

9.3.3 Response Message

9.3.3.1 Data Field Returned in the Response Message

The data field of the response message contains the TLV coded data object referred to in parameters P1 and P2 of the command message.

9.3.3.2 Processing State Returned in the Response Message

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

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return the following error conditions:

SW1	SW2	Meaning
‘6A’	‘88’	Referenced data not found

9.4 GET STATUS command

9.4.1 Definition and Scope

The GET STATUS command is used to retrieve Card Manager, Executable Load File and Application related 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 is coded according to the following table:

Code	Value	Meaning
CLA	'80' or '84'	Open Platform command
INS	'F2'	GET STATUS
P1	'xx'	Reference Control Parameter P1
P2	'xx'	Reference Control Parameter P2
Lc	'xx'	Length of data field
Data	'xxxx…'	Search criteria (and message DAP if present)
Le	'00'	

9.4.2.1 Reference Control Parameter P1

Reference control parameter P1 is used to select a certain subset of card elements to be included in the response.

The reference control parameter P1 is coded according to the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1								Indicate Card Manager
	1							Indicate Application(s)
		1						Indicate Executable Load File(s)
			0	0	0	0	0	RFU

The following combinations for P1 are acceptable:

‘80’ – Card Manager only.
‘40’ – Applications only.
‘20’ – Executable Load Files only.
‘E0’ – Card Manager, Applications and Executable Load Files.
‘C0’ – Card Manager and Applications.
‘60’ – Applications and Executable Load Files.
‘A0’ – Card Manager and Executable Load Files.

Security Domains are considered applications in this context and are distinguished from other applications on the card via the application privileges byte.

9.4.2.2 Reference Control Parameter P2

The reference control parameter P2 is coded according to the following table:

b8	b7	b6	B5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	0		RFU
							0	Get first or all occurrence(s)
							1	Get next occurrence(s)

The get next occurrence does not apply when retrieving the status of the Card manager.

9.4.2.3 Data Field Sent in the Command Message

The data field in the command message contains the TLV coded search qualifier(s). The tags '9F70' (application life cycle status) can be applied and '4F' (application AID) must be applied. The life cycle search qualifier is bit map coded according to the life cycle status byte.

In order to search for all the occurrences that match the selection criteria according to reference control parameter P1, a search criteria of ‘4F’ ‘00’ (i.e. all matching AID entries) must be present in the command message.

In order to search for Visa applications that are selectable, reference control parameter P1 must be equal to ‘40’ and a search criteria of ‘4F’ ‘05’ ‘A0’ ‘00’ ‘00’ ‘00’ ‘03’ ‘9F70’ ‘01’ ‘07’ must be present in the command message.

9.4.3 Response Message

9.4.3.1 Data Field Returned in the Response Message

Based upon the search criteria of the command data field and the selection criteria of Reference control parameter P2, multiple occurrences of the following data structure may be returned:

Length	value	Meaning
1	'xx'	Length of AID
1-n	'xxxx…'	AID of Card Manager, Executable Load File or Application
1	'xx'	life cycle state of the Card Manager, Executable Load File or Application
1	'xx'	Application Privileges

The Card Manager, Application and Executable Load File life cycle states are coded according to Section 9.1, ‘General Coding Rules‘.

The Application Privileges are coded according to Section 9.1, 'General Coding Rules'. The Card Manager and Executable Load File privileges byte, while present, are not relevant in this context.

9.4.3.2 Processing State Returned in the Response Message

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

The ICC may return the following warning conditions:

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

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return the following error conditions:

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found
'6A'	'80'	Incorrect values in command data

9.5 INSTALL command

9.5.1 Definition and Scope

Installing an application or Security Domain requires the invocation of several different on-card functions. The INSTALL command is used to instruct a Security Domain with the Delegated management privilege or the Card Manager as to which installation step it shall perform during an application installation process.

9.5.2 Command Message

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

Code	Value	Meaning
CLA	'80' or '84'	Open Platform command
INS	'E6'	INSTALL
P1	'xx'	Reference control parameter P1
P2	'xx'	‘00’
Lc	'xx'	Length of data field
Data	'xxxx…'	Install data (and message DAP if present)
Le	'00'	

9.5.2.1 Reference Control Parameter P1

The reference control parameter of the INSTALL command is coded as follows:

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

The individual bits are used as follows:

- b4=1 indicates that the application to be installed (or already installed) is to be made selectable.
- b3=1 indicates that an application is to be installed onto the card.
- b2=1 indicates that a Load File is to be loaded.

It is possible to install and make an application selectable using one single INSTALL command i.e. setting both b4 and b3.

9.5.2.2 Security Control Parameter P2

The security control parameter of the INSTALL command is RFU and 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]

Data field of INSTALL APDU for Load File loading (b2 of P1 set to 1):

Presence	Length (no of 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 DAP
Conditional	0-n	Load File DAP
Mandatory	1	Length of load parameter field
Conditional	0-n	Load parameter field
Mandatory	1	Length of Load Token
Conditional	0-n	Load Token (Load DAP)

The Load File DAP (hash) and Load Token are mandatory for Delegated Management. The Load File DAP is optional and Load Token shall not be present if Delegated Management is not used.

9.5.2.3.2 Data Field for INSTALL [for install]

Data field of INSTALL APDU for application installation (b3 of P1 set to 1):

Presence	Length (no. of bytes)	Name
Mandatory	1	Length of Load File AID
Mandatory	5-16	Load File AID
Mandatory	1	Length of AID within Load File
Mandatory	5-16	AID within Load File
Mandatory	1	Length of application instance AID
Mandatory	5-16	Application instance identifier (AID)
Mandatory	1	Length of application privileges
Mandatory	1	Application privileges
Mandatory	1	Length of Install parameter field
Mandatory	2-n	Install parameter field
Mandatory	1	Length of Install Token
Conditional	0-n	Install Token (Install DAP)

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

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

Open Platform cards use the Application Specific Install parameter field (identified by the tag ‘C9’) to enable parameters specific to the installation of the application to be known to the application. If an application does not require application specific parameters the corresponding length contains '00'. This tag and those defined for system specific parameters requires that the install method parameter field contain at least the value ‘9F’ ‘00’.

Open Platform cards currently require a single byte for Application Privileges coded according to section 9.1.2, ‘Application Privileges Coding’. If an application is only being installed,(i.e. and not made selectable with the same INSTALL command), any privileges can be set except for the “default selected” privilege.

The Instance AID, the Application Privileges and the Application Specific Parameters shall be made known to the application. For the Instance AID and the Application Privileges, the length and value are made known and for the Application Specific Parameters, the length and value of the TLV object are made known (i.e. the tag ‘C9’ is discarded).

9.5.2.3.3 Data Field for INSTALL [for Make Selectable]

Data field of INSTALL APDU for making an installed application selectable (b4 of P1 set to 1 or b3 and b4 set to 1):

Presence	**Length (no. of bytes)**	**Name**
Mandatory	1	Length of Load File AID = ‘00’
Not present	-	-
Mandatory	1	Length of AID within Load File = ‘00’
Not present	0	-
Mandatory	1	Length of application instance AID
Mandatory	5-16	Application instance identifier (AID)
Mandatory	1	Length of application privileges
Mandatory	1	Application Privileges
Mandatory	1	Length of Install parameter field = ‘00’
Not present	-	-
Mandatory	1	Length of Install Token
Conditional	0-n	Install Token

9.5.2.4 Load and Install Parameters

The Load and Install parameters fields are TLV structured values comprising of optional system specific and application specific (install parameters) values.

The following tags may apply:

Tag	Length	Value (Name)
'C9'	Variable	Application specific parameters
'EF'	Variable	System specific parameters
'C6'	2	Non volatile code space limit
'C7'	2	Volatile data space limit
'C8'	2	Non volatile data space limit

9.5.3 Response Message

A response message is always returned but only relevant in the case of Delegated Management. In the case where the Card Manager is processing the INSTALL, a single byte of '00' will be returned indicating that no additional response data is present. In addition if a INSTALL [for load] is being issued to a Security Domain with Delegated Management privileges, a single byte of '00' will 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 Installation, the data field of the response message contains the Install Receipt as LV coded data.

Presence	Length (no. of bytes)	Name
Mandatory	1	Length of Install Receipt
Conditional	0-n	Install Receipt

Format of the Install Receipt used in Delegated Installation:

Length	Data Element
1	Length of Install Receipt (DAP)
1-n	Install Receipt (DAP)
1	Length of Confirmation Counter
1-n	Confirmation Counter
1	Length of Card Unique Data
1-n	Card Unique Data

9.5.3.2 Processing State Returned in the Response Message

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

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return the following error conditions:

SW1	SW2	Meaning
‘65’	‘81’	Memory failure
‘6A’	‘80’	Incorrect parameters in data field
‘6A’	‘84’	Not enough memory space
‘6A’	‘88’	Referenced data not found

9.6 LOAD command

9.6.1 Definition and Scope

This section defines the Load File structure as transmitted in the LOAD command data field for loading an application. It does not define the ICC internal handling or storage of the Load File.

Multiple LOAD commands transfer a Load File block to the card by breaking the Load File into smaller components for transmission. Each LOAD command is 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.

After receiving the complete 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'	Open Platform command
INS	'E8'	LOAD
P1	'xx'	Reference control parameter
P2	'xx'	Block number
Lc	'xx'	Length of data field
Data	'xxxx..'	Load data (and message DAP if present)
Le	'00'	

9.6.2.1 Reference Control Parameter P1

The following table codes the Reference Control Parameter P1

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0								More blocks
1								Last block
	0	0	0	0	0	0	0	RFU

9.6.2.2 Reference Control Parameter P2

Reference control parameter P2 contains the block number, which is 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 the Load File.

A complete Open Platform Load Files is structured as follows:

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

A DAP block is optional within a Load File unless the card to which the Load File is being transferred contains a Security Domain with Mandated DAP verification privileges.

A Load File may contain multiple DAP blocks.

The lengths defined in the above table must be coded according to BER i.e. all lengths less than 128 are coded on 1 byte, length values of 128 to 255 are coded on 2 bytes etc.

9.6.3 Response Message

A response message is always returned but only relevant in the case of the last LOAD command during Delegated Management. For all LOAD commands, preceding the last LOAD command, a single byte of '00' will be returned indicating that no additional response data is present.
In the case where the Card Manager is processing the last LOAD command, a single byte of '00' will be returned indicating that no additional response data is present.

9.6.3.1 Data Field Returned in the Response Message

The data field of the response message is normally a single byte of'00' except following the last LOAD command at which time the Load Receipt is returned. Load Receipts are mandatory during Delegated Management.

Presence	Length (no. of bytes)	Name
Mandatory	1	Length of Load Receipt
Conditional	0-n	Load Receipt

Format of the Load Receipt used in Delegated Loading:

Length	Data Element
1	Length of Load Receipt DAP
1-n	Load Receipt DAP
1	Length of Confirmation Counter
1-n	Confirmation Counter

1	Length of Card Unique Data
1-n	Card Unique Data

9.6.3.2 Processing State Returned in the Response Message

A successful execution of the command is coded by ‘90’ ‘00’.

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return 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

9.7 PUT KEY

9.7.1 Definition and Scope

The PUT KEY command is used to

1. replace a single key or multiple keys within an existing key set version;
2. replace an existing key set version with a new key set version; or
3. add a new key set version containing single or multiple keys.

A key is uniquely identified by the combination of its key set version and its key index. The Card Manager or a Security Domain may have multiple key set versions and multiple keys may exist within a given key set version.

9.7.2 Command Message

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

Code	Value	Meaning
CLA	'80' or '84'	Open Platform command
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 message DAP if present)
Le	'00'	

9.7.2.1 Reference Control Parameter P1

Reference control parameter P1 is used to define the current key set version and whether more PUT KEY commands will follow this one.

The current key set version identifies a key set version that is already present on the card. A value of '00' in the current key set value indicates that a new key set version is being added. (The new key set version is indicated in the data field of the command message).

The key set version is coded from '01' to '7F'

The reference control parameter P1 of the PUT KEY command message is coded according to the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0								Last (or only) command
1								More PUT KEY commands
	x	x	x	x	x	x	x	current key set version

9.7.2.2 Reference Control Parameter P2

Reference control parameter P2 is used as key index and to indicate if 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 index of this key. When multiple keys are contained in the command message data field, P2 indicates the index of the first key in the command data field. Subsequent keys in the command message data fields follow on from this index

The key index is coded from '01' to '7F'

The reference control parameter P2 of the PUT KEY command message is coded according to the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0								Single key
1								Multiple keys
	x	x	x	x	x	x	x	key index

9.7.2.3 Data Field Sent in the Command Message

The command message data field contains the new key set version number followed by one or multiple key set data field.

The new key set version identifies the version of a new key set to be created on the card (current key set version in P1 = ‘00’) or the version with which the current key set (current key set version in P1 != ‘00’) will be replaced.

When replacing keys, the new keys must be presented to the card in the same format as they are already present on the card e.g. it is not possible to change the size of an existing key.

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.

If the data field contains multiple keys, the keys all belong to the same key set version and the sequence in the command data field reflects the key index sequence. The following diagram shows the coding of the data field, optionally loading a second and a third key:

New key set version

key set data field (implicit key index P2+0)

key set data field (implicit key index P2+1)

key set data field (implicit key index P2+2)

The key set data field is structured according to the following table:

Length	Meaning
1	Algorithm ID of key
1-n	Length of key
variable	Key data value
0-n	Length of Key check value
variable	Key check value (if present)

9.7.3 Response Message

9.7.3.1 Data Field Returned in the Response Message

The data field of the response message contains in clear text the key-set version followed by the key check values, if any, as presented in the command message data field. The personalization server may use the returned key-set version and key check values to verify the correct loading of the key-set version

9.7.3.2 Processing State Returned in the Response Message

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

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return the following error conditions:

SW1	SW2	Meaning
'65'	'81'	Memory failure
'6A'	'84'	Not enough memory space
'6A'	'88'	Referenced data not found
'94'	'84'	Algorithm not supported
'94'	'85'	Invalid key check value

9.8 SELECT command

9.8.1 Definition and Scope

The SELECT command is used for selecting an application. The Card Manager only processes SELECT commands indicating the 'select by name' option. All other options are 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	'xxxxx…'	AID of application or file to be selected
Le	'00'	

9.8.2.1 Reference Control Parameter P1

Reference control parameter P1 is coded according to the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
					1			Select by name

9.8.2.2 Reference Control Parameter P2

Reference control parameter P2 is 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

9.8.2.3 Data Field Sent in the Command Message

The data field of the command contains the AID of the application to be selected.

9.8.3 Response Message

9.8.3.1 Data Field Sent in the Response Message

Response data consists of information specific to the selected application or file.

The following table defines the coding of File Control Information for the Card Manager and Security Domains:

Tag	Description	Presence
'6F'	File Control Information (FCI template)	mandatory
'84'	Application / file AID	mandatory
'A5'	Proprietary data	mandatory
'9F6E'	Application production life cycle data	mandatory
‘9F65'	Maximum length of data field in command message	mandatory

9.8.3.2 Return Processing State

A successful execution of the command is coded by ‘90’ ‘00’.

The ICC may return the following warning conditions when the Card Manager is being selected:

SW1	SW2	Meaning
'62'	'83'	Card life cycle state is CM_LOCKED

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return the following error conditions when an application is being selected:

SW1	SW2	Meaning
'6A'	'81'	Function not supported e.g. Card life cycle state is CM_LOCKED
'6A'	'82'	Selected application / file not found

9.9 SET STATUS command

9.9.1 Definition and Scope

The SET STATUS command is used to modify the life cycle state of the card or the currently selected application.

In Card Manager life cycle state SECURED, this command shall only be used according to the Card Issuer's policy regarding secure communication.

9.9.2 Command Message

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

Code	Value	Meaning
CLA	'80' or '84'	Open Platform command
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 message DAP if present)
Le	'00'	

9.9.2.1 Status Type Parameter

The status type parameter of the SET STATUS command message is coded according to the following table to indicate if the change in status shall apply to the Card Manager or an Application. Security Domains are considered applications in this context. Other status types are Reserved for Future Use or application specific:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1								Indicate Card Manager
	1							Indicate Application(s)
		0	0	0	0	0	0	RFU

9.9.2.2 State Control Parameter P2

State control parameter P2 is coded according to Section 9.1.1, 'Life Cycle Status Coding'.

9.9.2.3 Data Field Sent in the Command Message

The data field shall contain the AID of the Card Manager or Application.

9.9.3 Response Message

9.9.3.1 Data Field Returned in the Response Message

The data field of the response message is not present.

9.9.3.2 Processing State Returned in the Response Message

A successful execution of the command is coded by ‘90’ ‘00’.

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return the following error conditions:

SW1	SW2	Meaning
‘6A’	‘80’	Incorrect values in command data
‘6A’	‘88’	Referenced data not found

This page intentionally left blank.

Part V
Security and Key Management

This page intentionally left blank.

10. Secure Communication

The Open Platform documentation broadly defines the notion of secure communication over and above that defined in EMV and 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 a Secure Channel exists. The following section defines the Secure Channel protocol used by the Card Manager and Security Domains. Applications that utilize the services of Security Domains can use this same Secure Channel protocol. This channel provides a means for the card, being the Card Manager, a Security Domain or application, to communicate with an off-card entity within a logically secure environment.

10.1 Secure Channel

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

The 3 levels of security provided by the Secure Channel are:

- Mutual authentication - in which the card and the off-card entity each proving that they have knowledge of the same secrets;
- Integrity and 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 unauthenticated entity.

A further level of security applies to secret keys that shall always be transmitted as confidential data.

10.1.1 Usage rules

These rules apply to the Secure Channel as defined herein.

The following rules have been defined and apply to:

- All applications including the Card Manager and Security Domains:
  - Selecting the application (SELECT command) never requires the use of a Secure Channel. The selection process, by its very nature, is always external to the Secure Channel.

  - The initiation of a Secure Channel also constitutes mutual authentication between the card and an off-card entity.
  - The level of security for secure messaging commands following the initiation of a Secure Channel is defined during the initiation of the Secure Channel.
  - If keys are being transmitted to the card, immaterial of whether a level of confidentiality and/or integrity is being used across the complete APDU message, the keys within the data field must always be secured with an additional level of confidentiality.
  - The Data Encryption Standard (DES), and specifically triple DES, is used in all cryptographic functions relating to a Secure Channel.
  - All keys used within the Secure Channel must be 16 byte (128 bit) keys providing 112-bit encryption strength.

- The Card Manager:
  - Retrieving basic card and/or issuer information (GET DATA) never requires the use of a Secure Channel. The GET DATA command may be performed outside of, or within, a Secure Channel.
  - All other Card Manager APDU commands are either involved in the initiation of the Secure Channel or must be used within a Secure Channel.
  - Prior to the card reaching a secure state (i.e. state of OP_READY or INITIALIZED), the card assumes it is in a secure environment, and the card and the off-card entity must at least authenticate each other to open a Secure Channel.
  - On reaching a secure state (i.e. sate of SECURE or CM_LOCKED), the card assumes it is always in an insecure environment. The card and the off-card entity must authenticate each other and all communications from the off-card entity to the card must contain integrity checking.
- Security Domains with Delegated Management privileges:
  - All Security Domain APDU commands are either involved in the initiation of the Secure Channel or must be used within a Secure Channel.
  - In the card state of OP_READY or INITIALIZED, the Security Domain and the off-card entity must at least authenticate each other to open a Secure Channel.
  - In the card state of SECURE or CM_LOCKED, the Security Domain and the off-card entity must authenticate each other and all communications from the off-card entity to the card must contain integrity checking.

Applications using the services of a Security Domain may use the exact same rules as defined above when using a Secure Channel or define their own application specific rules.

10.2 Mutual Authentication

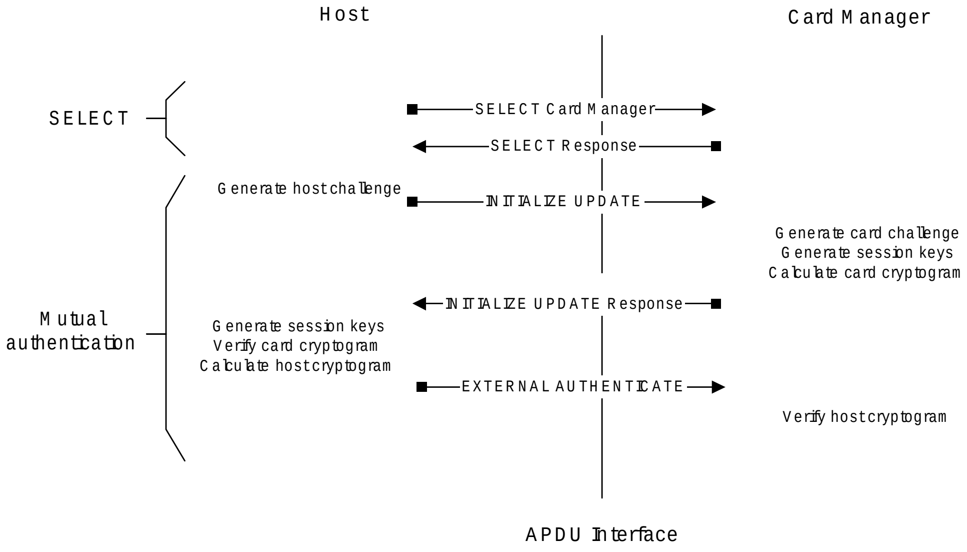
Mutual authentication is achieved through the process of initiating a Secure Channel and provides assurance to both the card and the off-card entity that they are communicating with an authenticated entity. This means that both entities have knowledge of the same secret information (card static keys). If any step in the mutual authentication process fails, the process shall be restarted i.e. new challenges and session keys generated.

- The Secure Channel is always initiated (see section 14.1 for the INITIALIZE UPDATE command) by the off-card entity by passing a “host” challenge (random data unique to this session) to the card.
- The card, on receipt of this challenge, generates its own “card” challenge (again random data unique to this session).
- The card, using the host challenge, the card challenge and its internal static keys, creates new secret data (session keys) and generates a first cryptographic value (card cryptogram) using one of its newly created session keys (see section 13.2).
- This card cryptogram along with the card challenge 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 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 section 14.2 for the EXTERNAL AUTHENTICATE command).
- Again, 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.

Mutual authentication Flow

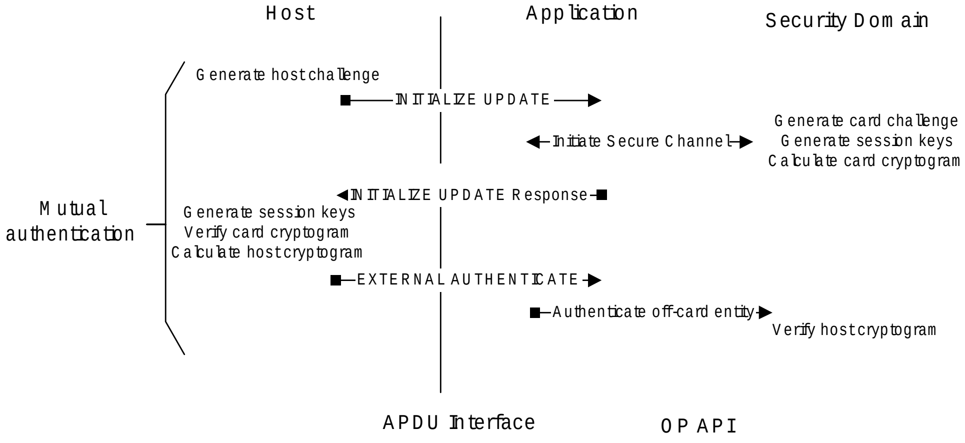
The following flow is an example of Mutual authentication between a card and an off-card entity. This flow shows mutual authentication occurring between the Card Manager and an off-card entity.

Mutual authentication Flow (Card Manager)



Expanding the authentication process shown in the flow described in section 7.6, it can be seen how an application would use the services of a Security Domain to achieve mutual authentication.

Mutual authentication Flow (Application using Security Domain services)



10.3 Secure Messaging

Secure messaging allows an off-card entity to add confidentiality and/or integrity to the composition of an APDU command prior to the command being transmitted to a card.

10.3.1 Message Integrity

Message integrity provides a means for the card to verify that commands are received from an authenticated off-card entity and have not been altered.

Message integrity also provides a means for the card to verify that commands received from an off-card entity have only been transmitted in sequence by the authenticated off-card entity.

This level of integrity is provided by applying multiple chained DES operations (using a session key generated during the mutual authentication process) across the header and data field of an APDU command to generate a MAC (Message Authentication Code).

The card, on receipt of the message containing a MAC, using the same session key, performs the same operation and by comparing its internally generated MAC with the 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 MAC applies to the clear text data.)

The integrity of the sequence of commands being transmitted to the card is achieved by using the 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.

10.3.2 Message Data Confidentiality

Data confidentiality provides a means for data to be transmitted across an open network without the fear of the data being intercepted for the purpose of analysis. This level of security cannot be applied to the message as a stand-alone secure messaging system and can only be applied along with message integrity.

Applying multiple chained DES operations (using a session key generated during the mutual authentication process) across the data field of the command message to be transmitted to the card achieves this level of confidentiality.

There is no relationship between multiple messages with regards to message data confidentiality i.e. there is no chaining requirement across messages.

This page intentionally left blank.

11. Cryptographic Keys

In order to support Open Platform security requirements, cryptographic keys must be present within the entities defined by the Open Platform card architecture i.e. the Issuer Security Domain (Card Manager) and other Security Domains that may be present on the card. Applications may also require the use of keys but these are outside the scope of this specification. Off-card key management is also outside the scope of this specification.

Keys for these entities are stored in mutable persistent memory and are grouped within key sets.

Key sets and their keys have the following characteristics:

- Each key set within an entity is uniquely identified by a Key Set Version.
- A key set may consist of one or more keys.
- A Key Index uniquely identifies each key within a key set.

11.1 Card Manager (Issuer Security Domain) Keys

The Card Manager, which is also the Issuer Security Domain, by default must have knowledge of a variety of keys. These keys are used for the following functions:

- Security for the management of the card and its content,
- Security support for applications using the services of the Issuer Security Domain.

The following are the minimum static key requirement for the Card Manager (Issuer Security Domain) depending on functionality.

Key	Usage	Length	Remark
Encryption	Authentication & encryption (DES)	16 bytes	Mandatory
Message Authentication Code (MAC)	MAC Verification (DES)	16 bytes	Mandatory
Key Encryption Key (KEK)	Key decryption (DES)	16 bytes	Mandatory
Token	Token verification (RSA Public Key)	1024 bits	Delegated Management only
Receipt	Receipt Generation (DES)	16 bytes	Delegated Management only

11.1.1 Secure Channel Keys

A Card Manager must 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 Encryption Key, the MAC key and the Key Encryption Key (KEK). While the KEK is always used as a static key, the Encryption and MAC keys are only used to generate session keys during the initiation of a Secure Channel.

11.1.2 Token key

If the card supports Delegated Management, the Card Manager must have one key set containing a 1024 bit RSA public key to be used to verify a load or install token. This key set version shall not be used for Secure Channel initiation.

11.1.3 Receipt Key

If the card supports Delegated Management, the Card Manager must have one key set containing a double length DES key to be used to generate load, install or delete receipts. This key set version shall not be used for Secure Channel initiation.

11.2 Security Domain Keys

Security Domains must also have knowledge of a variety of keys. These keys are used for the following functions:

- Security for the management of the Security Domain,
- Security for the management of card content (Delegated Management),
- Security support for applications using the services of the Security Domain,
- Verification of card content updates (DAP verification).

The following are the minimum static key requirement for a Security Domain depending on functionality. If DAP verification is supported only one method of verification is required i.e. DES or Public Key.

Key	Usage	Length	Remark
Encryption	Authentication & encryption (DES)	16 bytes	Mandatory
Message Authentication Code (MAC)	MAC Verification (DES)	16 bytes	Mandatory
Key Encryption Key (KEK)	Key decryption (DES)	16 bytes	Mandatory
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)	16 bytes	DAP verification only

11.2.1 Secure Channel Keys

A Security Domain must 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 Encryption Key, the MAC key and the Key Encryption Key (KEK). While the KEK is always used as a static key, the Encryption and MAC keys are only used to generate session keys during the initiation of a Secure Channel.

11.2.2 DAP verification Key

If the Security Domain supports DAP verification, it must have one key set containing a double length DES key or 1024 bit RSA public key that can be used to verify a load file data block signature. This key set version shall not be used for Secure Channel initiation.

This page intentionally left blank.

12. Cryptographic and Hashing Algorithms

An Open Platform card may support many types of security functions for use by applications. This section defines the cryptographic algorithms and the hashing method that are required for Open Platform.

12.1 Data Encryption Standard (DES)

The data encryption standard 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.

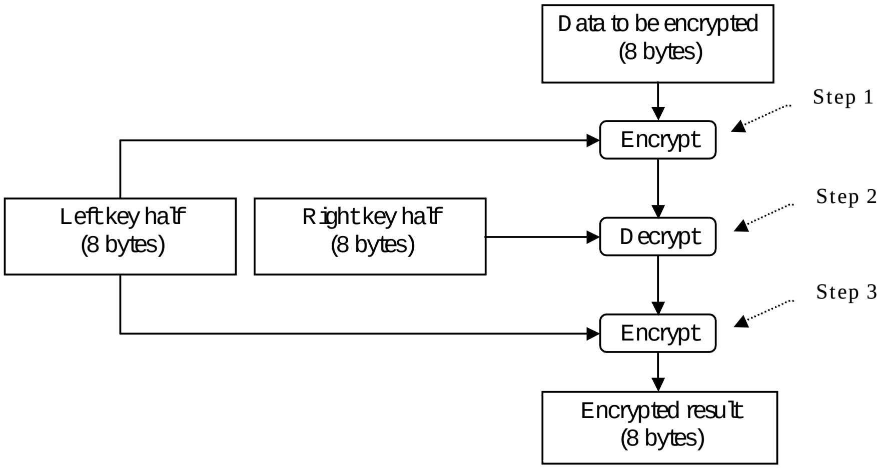
12.1.1 Encryption/Decryption

All encryption and decryption algorithms implemented in the Card manager or Security Domains require the security of triple DES (DES3ENC or DES3DEC) using a double length key.

The DES algorithm shall be used for the following purposes:

- DES session key creation;
- Authentication cryptogram generation and verification;
- Key encryption/decryption;
- DES key check value generation and verification;
- Encryption/decryption of the data field of the command message;
- APDU MAC generation and verification;
- Generation and verification of Delegated Management receipts; and,
- If DES is used for DAP verification, generation and verification of the load file data block signature.

The following figure defines the required method of triple DES encryption. Triple DES decryption is the exact opposite of this operation.



Triple DES Encryption (DES3ENC)

12.1.2 Cypher Block Chaining (CBC)

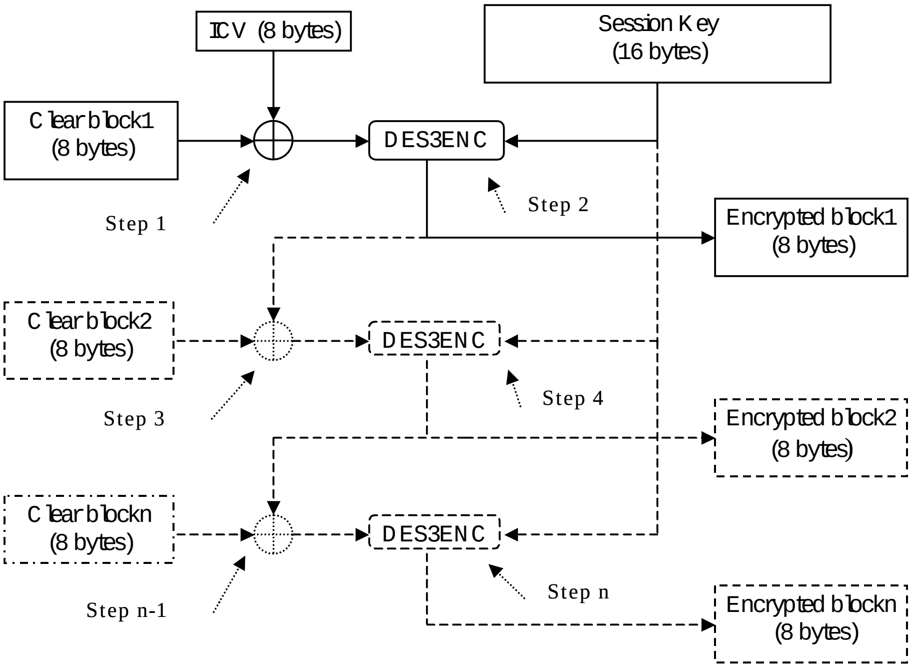
Cypher block chaining is a method of linking multiple 8 byte blocks through the encryption mechanism. This method of chaining is used both in signing and encrypting data.

CBC mode shall be used for the following purposes:

- Authentication cryptogram generation and verification;
- Encryption/decryption of the data field of the APDU command message;
- APDU MAC generation and verification;
- Generation and verification of Delegated Management receipts; and,
- If DES is used for DAP verification, generation and verification of the load file data block signature.

12.1.2.1 Chained data encryption

When encrypting data (i.e. data field of the command message), each result of an 8-byte block encryption becomes part of the encrypted result as well as being used as input for the next operation. In encryption mode, the Initial Chaining Vector (ICV) is always 8 bytes of binary zero ('00'). Decryption of data is the opposite operation using the decryption (DES3DEC) operation in place of each encryption (DES3ENC) operation.

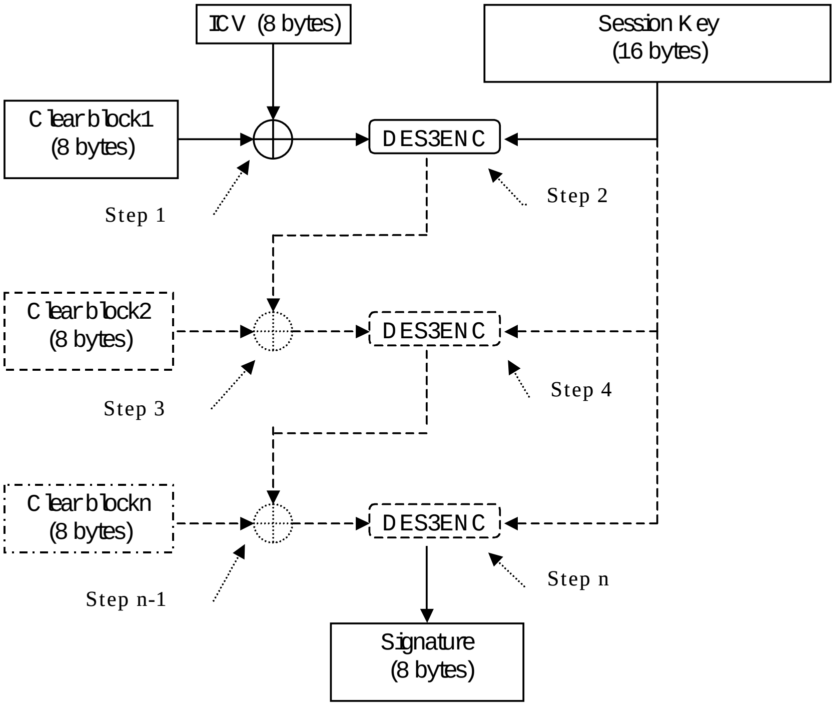


Chained Data Encryption

12.1.2.2 Signature method (full triple DES)

When signing, each resultant encryption is used as input for the next encryption and only the last result is retained. This last result is the signature (MAC or authentication cryptogram) of the data and is typically transmitted along with the data. The ICV value depends on the signing operation being performed.

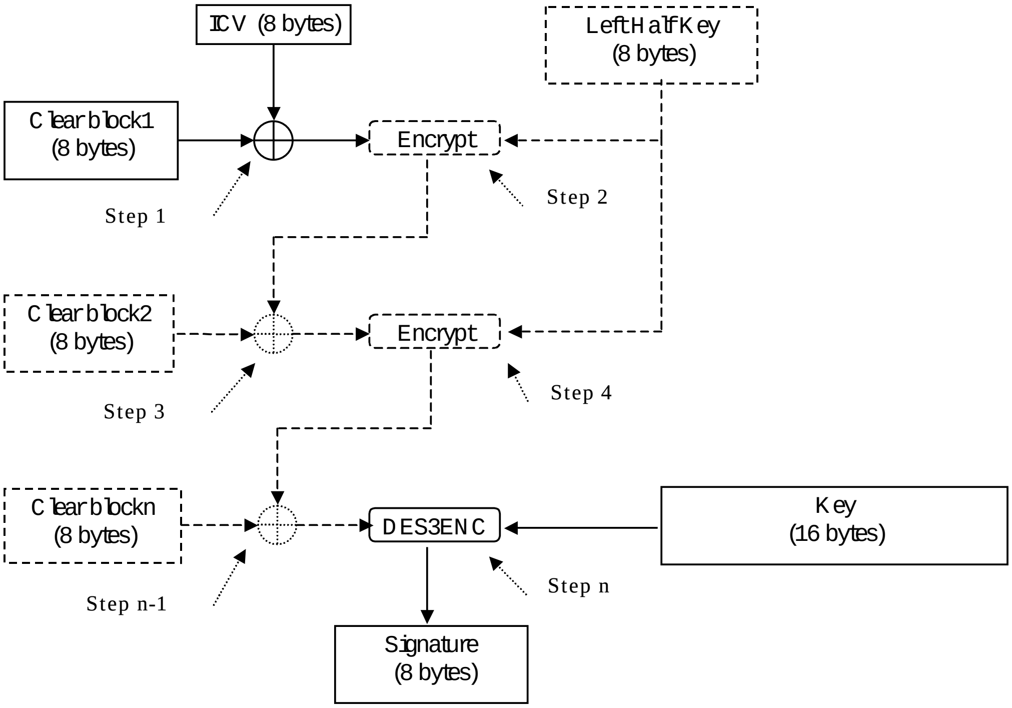
Verification of a signature requires the exact same operation with an additional comparison step.



Signature method (full triple DES)

12.1.2.3 Signature Method (single DES plus final triple DES)

An additional signing method is also present specifically for DES supported DAP generation and verification and receipt generation and verification. This method is specifically used to increase performance when a large amount of DES operations are taking place and uses a combination of single DES operations with only the final operation employing triple DES. The ICV value depends on the signing operation being performed.



Signature Method (single DES plus final triple DES)

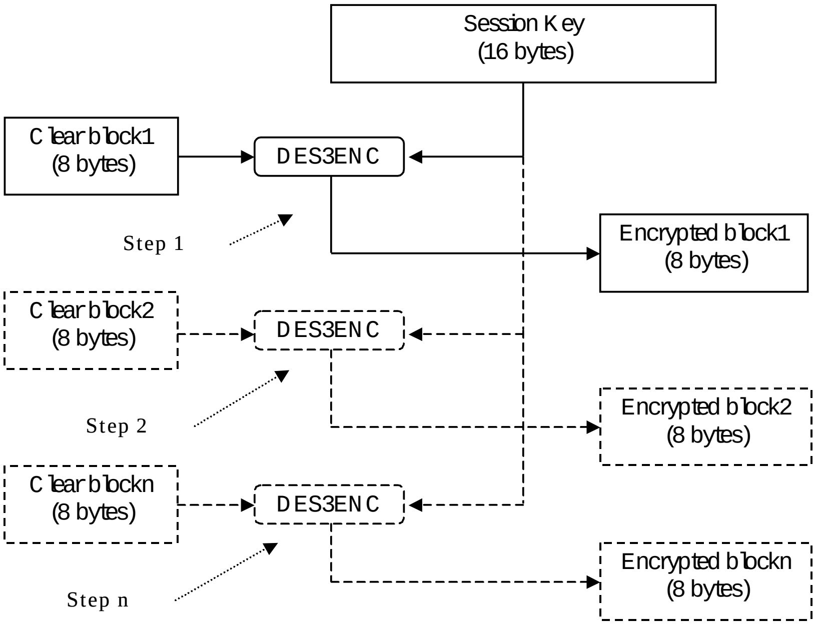
12.1.3 Electronic Code Book (ECB)

Electronic Code Book encryption is a method of encrypting one or multiple 8 byte blocks of data (typically keys).

ECB mode shall be used for the following purposes:

- DES Session key calculation;
- Key encryption/decryption; and,
- DES Key check value generation and verification.

When encrypting, each result of an 8-byte block encryption becomes part of the encrypted result. Decryption of data is the exact opposite operation where the decryption (DES3DEC) operation replaces each encryption (DES3ENC) operation.



Encryption

12.2 Secure Hash Algorithm (SHA-1)

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.

The SHA-1 function is only required on public key implementations and is used for the following purposes:

- Hashing of the load file for Delegated Management;
- Token generation; and,
- If public key DAP verification is supported, load file data block signature generation and verification.

The result of any SHA-1 digest is a 20-byte value typically used in a subsequent public key signing or verification operation.

12.3 Public Key Cryptography Scheme 1 (PKCS1)

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. The public key algorithm used for Open Platform is RSA (Rivest, Shamir and Alderman) and as all operations are signature generations of verifications, specifically PKCS1 is used.

The process of generating a signature uses a private key applied to the resulting 20-byte hash of the data being signed. The resultant signature is the same size as the public key modulus i.e. 1024 bits (128 bytes).

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.

All signature generation, using the Private key, is performed off-card and signature verification, using the Public key, is performed on-card for the following purposes:

- Token verification; and,
- If public key DAP verification is supported, load file data block signature verification.

This page intentionally left blank.

13. Cryptographic and Hashing Usage

13.1 DES Session Keys

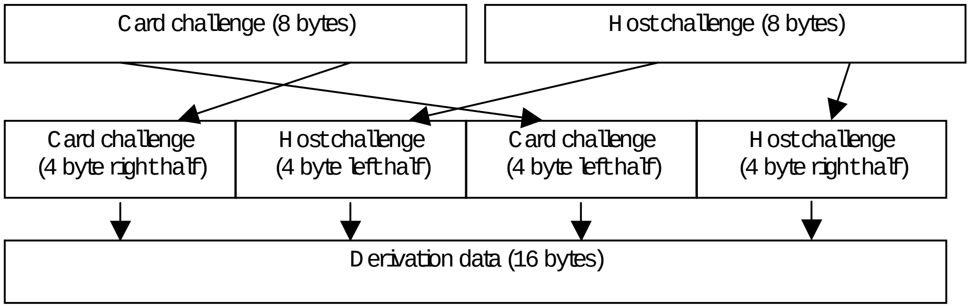
DES session keys are generated every time a Secure Channel is initiated and are used in the mutual authentication process. These same session keys may be used for subsequent commands if the security level indicates that Secure Messaging is required.

Session keys are generated to ensure that a different set of keys is used for each secure communication session. While this is not required for key encryption operations, it is important for authentication operations, MAC verification and generation and command message encryption and decryption. It is therefore only necessary to create session keys from the static Encryption and MAC keys.

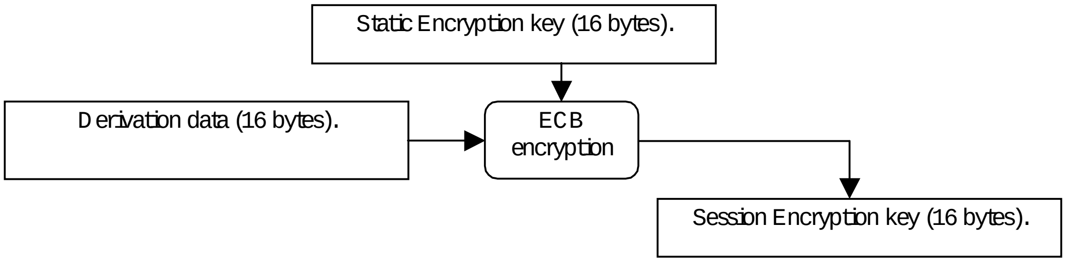
DES session keys are created using the static Encryption and MAC keys and the random host and card challenges. Creating session keys involves 3 steps.

1. Generating the session key derivation data. (The same derivation data is used to create both the Encryption and MAC session keys.);
2. Creating the Encryption session key; and,
3. Creating the MAC session key.

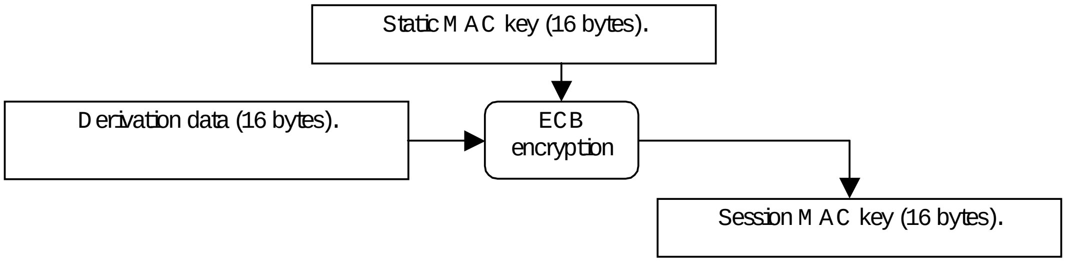
The DES operation used to generate these keys is always triple DES in ECB mode.


Card challenge (8 bytes)
Host challenge (8 bytes)
Card challenge (4 byte right half)
Host challenge (4 byte left half)
Card challenge (4 byte left half)
Host challenge (4 byte right half)
Derivation data (16 bytes)

Step 1 - Generate Derivation data


Static Encryption key (16 bytes).
Derivation data (16 bytes).
ECB encryption
Session Encryption key (16 bytes).

Step 2 - Create Encryption session key


Static MAC key (16 bytes).
Derivation data (16 bytes).
ECB encryption
Session MAC key (16 bytes).

Step3 - Create MAC session key

13.2 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 Encryption session key and the signing method described in section 12.1.2.2.

13.2.1 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 for the APDU MAC function (see section 13.3), the data must be padded with a further 8-byte block ('80 00 00 00 00 00 00 00').

The signature method, using the Encryption 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.

13.2.2 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 for the APDU MAC function (see section 13.3), the data must be padded with a further 8-byte block ('80 00 00 00 00 00 00 00').

The signature method, using the Encryption 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.

13.3 APDU command MAC generation and verification

The Secure Channel mandates the use of a MAC on the EXTERNAL AUTHENTICATION 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 MAC.

A 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.)

MAC generation and verification uses the MAC session key, an ICV and the signature method described in section 12.1.2.2.

Padding and modification of the APDU command header 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) must be incremented by 8 to indicate the inclusion of the MAC in the data field of the command message.
- The class byte must 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 rules for MAC padding are as follows and apply to the data block containing the APDU command header and data field of the command message:

- Append an '80' to the right of the data block.

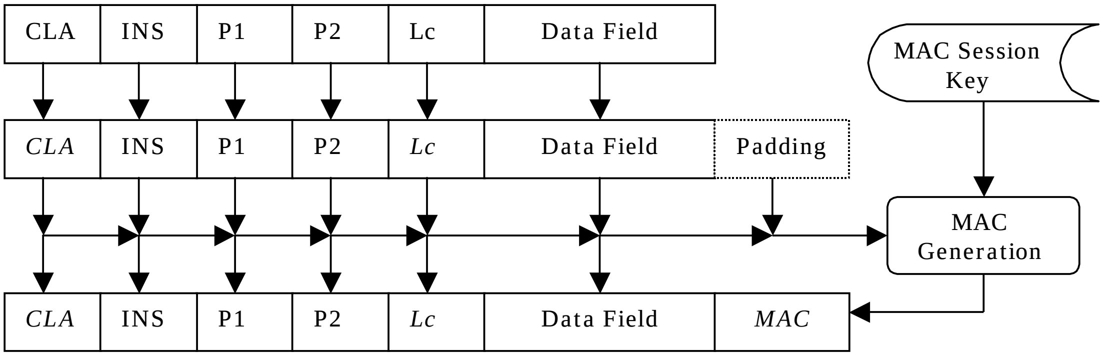
- If the resultant data block length is a multiple of 8, no further padding is required.
- Append binary zeroes to the right of the data block until the data block length is a multiple of 8.

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, as it is the first command in a sequence of Secure Messaging commands, the ICV is always zero.
- Following the EXTERNAL AUTHENTICATE command, the ICV is the MAC value generated for the previous command transmitted to the card.

The signature method, 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 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).



If no other secure messaging is required, the message is now prepared for transmission to the card. The card, in order to verify the MAC, must 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 MAC. The verified MAC must be retained and used as the ICV for any subsequent MAC verification. (This is true regardless of whether the APDU completed successfully or not i.e. a verified MAC shall never be discarded in favor of a previous verified MAC value.)

13.4 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 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. This excludes the header and the MAC but includes any data within the data field that has been encrypted for another purpose e.g. keys encrypted with the Key Encryption Key.

Command message encryption and decryption uses the Encryption session key and the encryption method described in section 12.1.2.1.

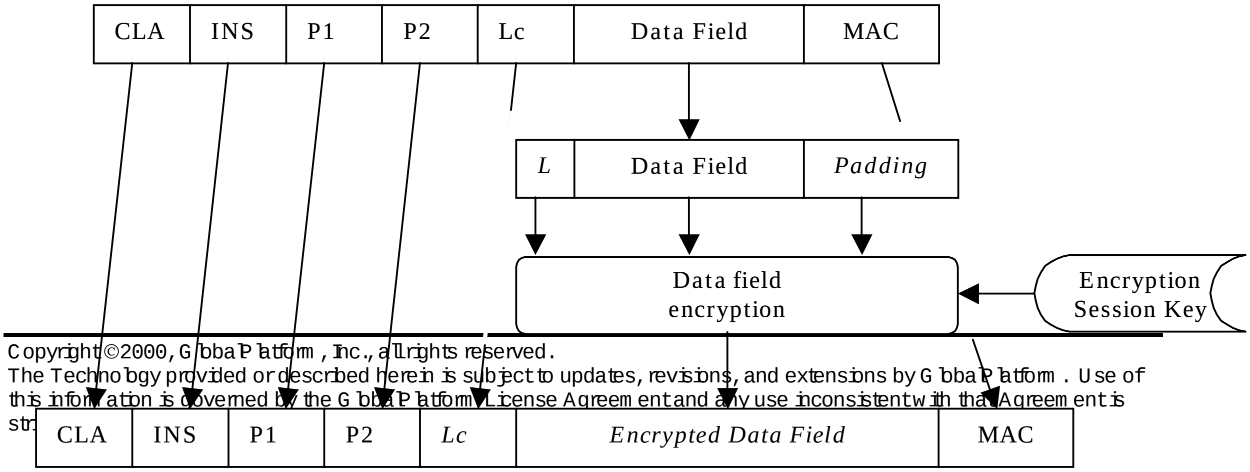
Prior to encrypting the data, the data must be padded. Unlike the MAC, this padding now becomes part of the data field and this necessitates further modification of the Lc value.

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.
- Append an ‘80’ to the right of the data field .
- If the resultant data field block length is a multiple of 8, no further padding is required.
- Append binary zeroes to the right of the data field until the data field length is a multiple of 8.

The number of bytes appended to the data field in order to fulfill the above padding must 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 Encryption key and the result of each encryption becomes part of the encrypted data field in the command message.



Note that 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 MAC. This decryption uses an ICV of binary zeroes and the same encryption session key employed by the off-card entity. The padding must be removed and Lc must be modified to reflect the length prior to encryption i.e. original clear text data plus MAC length.

13.5 Key Encryption and Decryption

Key encryption is used when transmitting key data to the card and is over and beyond the security level required for the Secure Channel i.e. all DES keys transmitted to a card (PUT KEY command) should be encrypted.

The key encryption process uses the static Key Encryption Key and the encryption method described in section 12.1.3.

As all DES keys are by their very nature a multiple of 8-byte lengths no padding is required for key encryption operations.

The encryption is performed across the key 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.

13.6 Key verification

In order to verify that the card received the correct key, each DES key transmitted to the card in the PUT KEY command has an associated key check value.

The check value generation process uses the clear text key using the encryption method described in section 12.1.3.

The data to be encrypted is binary zeroes and as it is defined to be 8 bytes, no padding is required.

The encryption is performed across the 8 binary zeroes and only the 3 right most bytes of the result are required as a check value. As with the encrypted keys the check value becomes part of the clear text data field in the command message.

Following the decryption of the keys, on-card verification of the key is achieved by performing the exact same operation as defined above and comparing the 3 right most bytes of the encryption with the check value in the data field.

13.7 Load File Data Block Signature Generation and Verification

Load File Data Block signature verification is typically used by a controlling 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 must be present on the card. The Security Domain must have knowledge of either a double length DES secret key or a 1024 bit public key.

13.7.1 DES Scheme

A controlling entity generates a DES signature across the complete load file data block using the DAP verification double length key, an ICV of binary zeroes and the signature method described in section 12.1.2.3. This signature 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).

Prior to generating the signature the data block must be padded in the following manner (following the signature operation, the padding is removed):

- 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.

The signing can now be performed across the padded data block. (Note that the ICV and the chaining are only used to link the blocks of the data currently being signed and therefore the ICV is binary zeroes.) The resultant signature is retained and 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 entity along with this 8-byte signature are placed in the load file as a DAP block to be transmitted to the card.

Exactly how a Card Manager and Security Domain interface during the DAP verification process is implementation specific but the verification operation mandates that the signature be verified using the Security Domain's double length DES key.

13.7.2 PKC Scheme

A controlling entity generates a signature across the complete load file data block. This RSA signature is the encryption with the DAP verification private key of the SHA-1 message digest of the load file data block. Padding of the data is as defined by the SHA-1 and PKCS1. This signature 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).

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 entity along with this 128-byte signature are placed in the load file as a DAP block to be transmitted to the card.

Exactly how a Card Manager and Security Domain interface during the DAP verification process is implementation specific but the verification operation mandates that the signature be verified using the Security Domain's Public key.

13.8 Token Generation and Verification

Tokens are used during Delegated Management in order to ensure that the load file being transmitted to the card and the applications being installed upon a card by an entity other than the card issuer, have been previously authorized by the issuer.

All cards supporting Delegated Management must perform token verification and the Issuer Security Domain (Card Manager) must have knowledge of a token verification public key to perform this verification operation.

13.8.1 Load Token

An issuer generates a load token and this is a signature authorizing the transmission of a load file to the card.

Generating a load token ensures the following:

- That only the application code included in this signature can be loaded to the card;
- DAP blocks included in the load file cannot be modified or removed;
- The AID of the package can only be that included in the signature; and,
- All applications within the load file can 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
P1	1
P2	1
Lc	1
Load file AID length indicator	1
Load file AID	5-16
Security Domain AID length indicator	1
Security Domain AID	0 or 5-16
Load parameters length indicator	1
Load parametersr	Var
Hash of Load file (see 13.8.1.1)	20

This RSA signature is the encryption with the token verification private key of the SHA-1 message digest of the above data. Padding of the data is as defined by the SHA-1 and PKCS1.

Each of the delegated load process commands (INSTALL [for load] command followed by multiple LOAD commands) is passed by the Security Domain to the Card Manager. The Card Manager is responsible for verifying the Load Token (signature) using the Card Manager's token verification public key. The Card Manager is also responsible for ensuring that the hash of the load file present in the INSTALL [for load] command, is a valid SHA-1 message digest of the load file.

13.8.1.1 Hash of the Load File

When hashing the load file using the SHA-1 digest method, the hash encompasses the full load file to be transmitted to the card in the order it is transmitted to the card. This begins with the DAP blocks, if present, and in the order they are present, and the load file data block. Inclusion implies that the full TLV object is included in the hashing algorithm i.e. the tags, lengths and values of both the DAP block(s) and the load file data blocks.

13.8.2 Install Token

An issuer generates an Install Token and this is a signature authorizing the installation to the card of an application contained in a previously loaded file.

Generating an install token ensures the following:

- That only an application residing in the executable load file included in this signature can be installed on the card;
- That only the application included in this signature and present in the executable load file can be installed on the card;
- That only the instance AID included in this signature can be used;
- That the application can only be installed with the privileges included in this signature; and,
- That only the parameters included in this signature can be used to install the application.

The Install Token is an RSA signature of the following data that will be included in the actual INSTALL [for install and make selectable] command to be transmitted to the card.

Name	Length
P1	1
P2	1
Lc	1
Executable Load file AID length indicator	1
Executable Load file AID	5-16
AID within the executable load file length indicator	1
AID within the executable load file	5-16
Instance AID length	1
Instance AID	5-16
Application privileges length indicator	1
Application privileges	1
Install parameters length indicator	1
Install parameters (system and application)	Var

This RSA signature is the encryption with the token verification private key of the SHA-1 message digest of the above data. Padding of the data is as defined by the SHA-1 and PKCS1.

The delegated install process command (INSTALL [for install/make selectable] command) is passed by the Security Domain to the Card Manager.

On receipt of the above command, the Card Manager must verify the install token using the Card manager's token verification public key.

13.9 Receipt Generation

Receipts are generated by the card during Delegated Management operations as proof to the issuer that the operation (load, install or delete) was successfully performed.

All cards supporting Delegated Management are responsible for generating receipts and the Issuer Security Domain (Card Manager) must have 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 section 12.1.2.3.

In addition to the receipt key, the Card Manager is also responsible for keeping track of a confirmation counter. The confirmation number is a 16 bit value that is incremented by one following each receipt generation.

13.9.1 Load Receipt

The load receipt provides proof to the issuer that the identified Security Domain performed a load of an application to a specific card.

A load 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
Executable Load file AID length indicator	1
Executable Load file AID	5-16
Security Domain AID length indicator	1
Security Domain AID	5-16

Prior to generating the signature, the data must be padded in the following manner (following the signature generation the padding is discarded):

- 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.

The signature method is applied across the padded data and the signature along with other required data is returned as the response message to the last LOAD command.

The entity performing the delegated loading 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.

13.9.2 Install Receipt

The install receipt provides proof to the issuer that the identified Security Domain performed an install of an application to a specific card. An install receipt is not generated for an INSTALL [for load] command.

An install 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
Executable Load file AID length indicator	1
Executable Load file AID	5-16
Instance AID length indicator	1
Instance AID	5-16

Prior to generating the signature the data must be padded in the following manner (following the signature generation the padding is discarded):

- 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.

The signature method is applied across the padded data and the signature along with other required data is returned as the response message to the INSTALL [for install] command.

The entity performing the delegated installation 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.

13.9.3 Delete Receipt

The delete receipt provides proof to the issuer that the identified Security Domain deleted an application or executable load file from a specific card.

A delete 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
Application or Executable Load file AID	5-16

Prior to generating the signature the data must be padded in the following manner (following the signature generation the padding is discarded):

- 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.

The signature method is applied across the padded data and the signature along with other required data is returned as the response message to the DELETE command.

The entity performing the delegated delete 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.

This page intentionally left blank.

14. Secure Channel APDU commands

14.1 INITIALIZE UPDATE Command-Response APDU

14.1.1 Definition and Scope

The INITIALIZE UPDATE command is used to transmit card and session data between the card and the host. This command initiates the initiation of a Secure Channel.

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.

14.1.2 Command Message

The INITIALIZE UPDATE command message is coded according to the following table:

Code	Value	Meaning
CLA	‘80’	Open Platform command
INS	‘50’	INITIALIZE UPDATE
P1	‘xx’	Key set version
P2	‘xx’	Key index
Lc	‘08’	Length of data
Data	‘xxxx…’	Host challenge
Le	‘00’	

14.1.2.1 Key set version (P1)

The key set version defines the key set version within the Security Domain to be used to initiate the Secure Channel, If this value is zero, the first available key chosen by the Security Domain will be used.

14.1.2.2 Key Index (P2)

This key index identifies a key index within the key set version. If this value is zero, the first key within the key set version will be used.

14.1.2.3 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.

14.1.3 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
Card challenge	8 bytes
Card cryptogram	8 bytes

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 key index used in initiation the Secure Channel.

The card challenge is an internally generated random number.

The card cryptogram is an authentication cryptogram (see section 13.2.1).

14.1.3.1 Return Processing State

A successful execution of the command is coded by ‘90’ ‘00’.

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return the following error condition:

SW1	SW2	Meaning
‘6A’	‘88’	Referenced data not found

14.2 EXTERNAL AUTHENTICATE Command-Response APDU

14.2.1 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 must precede this command.

14.2.2 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	‘xxxx…’	Host cryptogram and MAC
Le	‘00’	

14.2.2.1 Security level (P1)

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.

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	0	0	0	0	0	1	1	Encryption and MAC.
0	0	0	0	0	0	0	1	MAC
0	0	0	0	0	0	0	0	No secure messaging expected.

14.2.2.2 Parameter (P2)

Parameter P2 is coded to ‘00’.

14.2.2.3 Data Field Sent in the Command Message

The data field of the command message contains the host cryptogram (see section 13.2.2) and the APDU command MAC.

14.2.3 Data Field Returned in the Response Message

The data field of the response message is not present.

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

This command returns general error conditions as listed in section 9.1.3 General Error Conditions.

Additionally the ICC may return the following error condition:

SW1	SW2	Meaning
'63'	'00'	Authentication of host cryptogram failed

Appendices

This page intentionally left blank.

Appendix A

Appendix A details the Open Platform requirements for a Java Card™ implementation.

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 4.3 '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 card.

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 visa.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 must 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 must 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 openplatform 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 = 0x07
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 `CM_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 Global PIN.
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 Global PIN.
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 transition 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 CM_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 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 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 Global PIN.

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 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 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 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 Global PIN. The length of the PIN is retrieved from the PIN structure. The Card Manager locates the AID of the selected application in the registry and determines if the application has the required privilege. If the PIN is changed, the PIN try counter must 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 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 Global PIN. If the comparison is successful, the retry counter must be reset. If the comparison is unsuccessful, the retry counter must 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.

Appendix B

Appendix B details the Open Platform requirements for Windows for Smart Cards® implementation.

At this point in time the requirements of Open Platform for a Windows for Smart Card® platform have not been defined.