

Work Group 1

CALYPSO SPECIFICATION

Application Downloading

Draft

In use

Obsolete

Author

Editor



Revision list

Version	Date	Author	Modifications
1.0	090128	D. Garnier	Release of version 1.0

TABLE OF CONTENTS

FOREWORD	5
SUMMARY	8
1. INTRODUCTION	9
1.1. Scope	9
1.2. Intended Audience	9
1.3. Organisation	9
1.4. References	10
1.5. Acronyms	11
1.6. Notations	11
2. CONTEXT	12
2.1. Calypso Elements	12
2.2. Calypso Roles	12
2.3. Security Objectives	13
2.4. Compatibility with ISO24014	13
2.5. Form Factors	13
3. FUNCTIONAL REQUIREMENTS	14
3.1. Java Card Support	14
3.2. GlobalPlatform Support	14
3.3. Secure Communication	14
3.4. Security Domain Creation	14
3.5. Calypso Application Management	14
3.5.1. Calypso Application States	15
3.5.2. Installation	15
3.5.3. Pre-personalisation and Activation	16
3.6. Data Grouping Identifiers	19
3.6.1. Definition	19
3.6.2. Cinematic	20
3.6.3. Data content for DGI '9484'	21
3.6.4. Data content for DGI '2000'	21
3.6.5. Data content for DGI '2100'	21
3.6.6. Data content for DGI '2200'	22
3.6.7. Data content for DGI '3100'	22
3.6.8. Data content for DGI '3800'	23
3.6.9. Data content for DGI '410n'	23
3.6.10. Data content for DGI 'ssnn' ('01nn' to '1Enn')	24
3.6.11. Data content for DGI '8185'	24
3.6.12. Data content for DGI '84F2'	24
3.7. Error Handling	25
3.7.1. Rules	25
3.7.2. Cinematic	25
4. ANNEX A: SAM APPLICATION	29
4.1. Data Grouping Identifiers	29
4.1.1. Definition	29
4.1.2. Cinematic	30



4.1.3.	Data content for DGI '4110'	30
5.	ANNEX B: DISCUSSION & RATIONALES	32
5.1.	GlobalPlatform Support.....	32
5.2.	GlobalPlatform Roles	32
5.3.	Secure Communication.....	33
5.3.1.	Nominal Case.....	33
5.3.2.	Mobile Phones	34
5.4.	GlobalPlatform Management.....	38
5.4.1.	Security Domain Management	38
5.4.2.	Application Management	41
5.5.	GlobalPlatform Life Cycle.....	48
5.6.	Application Personalisation	49
5.6.1.	Nominal Case.....	49
5.6.2.	Support for older GlobalPlatform Specifications.....	50
5.7.	Error Handling.....	51

FOREWORD

This document presents a mechanism to install and configure a Calypso Revision 3 application in a smartcard compatible with the Java Card specification, i.e. a Java Card. It specifies the lifecycle of the application, the commands to pre-personalise it and the security and communication protocols required to communicate with the Java Card.

The objective of this document is to support the deployment of the Calypso technology on a wide range of devices not originally designed to support it, such as mobile phones, smart keys and multi-application smartcards.

Advantages of Java Card

Today, every deployment of the Calypso technology is based on “proprietary” smartcards. A proprietary smartcard uses proprietary mechanisms to manage the applications present in the smartcard. The simplest (and most common) type of proprietary smartcard supports a single application that cannot be updated post-issuance. For example, a simple proprietary smartcard running the VISA application cannot be updated with the MasterCard application.

An application for a proprietary smartcard is always designed to run on a specific model of proprietary smartcard. Furthermore the OS of the proprietary smartcard is usually tailored for a certain application. This means that it is not possible to run an application designed for the proprietary smartcard of a certain vendor on the proprietary smartcard of another vendor.

Since most proprietary smartcards support a single application, they are usually managed by a single entity. For transport applications, this means that every public transport network must issue and manage its own contactless smartcards.

Java Cards are quite unlike proprietary smartcards. The main difference is that Java Cards run standardized applications and are managed through standard mechanisms and protocols. Applications are designed according to the Java Card API and such an application will run on every Java Card model from every vendor. Similarly, every Java Card model from every vendor can be managed with standard mechanisms and protocols. Furthermore, Java Card applications are programs implemented with a subset of the Java language, making them extremely flexible and capable of supporting a wide range of requirements.

Thanks to its flexibility, the OS of a Java Card does not have to explicitly support an application. For example, a Java Card has no built-in support for the Calypso Revision 3 specification: an implementation of this specification for the Java Card platform implements every command and mechanism itself. However, the Java Card platform defines multiple APIs to simplify the implementation of an application. For example, every Java Card supports a set of standard cryptographic algorithms and protocols: a Java Card application programmer does not have to implement them again and the implementation can be tested and validated once and for all.

Every Java Card supports ISO 7816 and many also support ISO 14443, which are the standards required by Calypso Revision 2 and 3. Furthermore, the Java Card standard was designed to make it possible to emulate proprietary smartcards: in many cases, a proprietary smartcard can be replaced by a standard Java Card running a custom application. The net result is that there is usually no difference between a proprietary smartcard and a Java Card from the point of view of the card reader. This is the case for Calypso Revision 3: since the Calypso Revision 3 specification was designed with the Java Card standard in mind, a Calypso Revision 3 terminal manages seamlessly proprietary smartcards or Java Cards loaded with the Calypso Revision 3 application.

Secondly, the Java Card technology has been designed from the ground-up to support multiple applications from multiple service providers in a single smartcard while retaining an airtight isolation between each application. For example, a Java Card can support at the same time a Calypso Revision 3 application, a banking application and a loyalty application to collect airline miles. Each application is isolated from every other application so a malicious application cannot be used as a launching pad for attacking a sensitive application. Each application is managed by its application provider independently of every other provider. The global life cycle of the smartcard is still managed by the smartcard issuer, which retains the possibility of blocking a misbehaving application. The flexibility and security provided by the Java Card platform makes it possible to load new applications at any time, including after the deployment of the smartcard.

Another interesting feature of the Java Card standard is the possibility of supporting multiple instances of a single application. For example, the Calypso Revision 3 does not have to be loaded twice to support both the public transportation network in Paris and in Brussels. The Calypso Revision 3 is loaded once then two instances are created: the instance for Paris and the instance for Brussels. These two instances are completely independent from each other: each one would be managed by its respective owner. Those two public transport networks would not have to work together: they do not have to be aware that the Java Card supports another Calypso application. The number of possible instances of a Calypso Revision 3 application is only limited by the amount of free memory available in the Java Card. Since the amount of data required by a Calypso Revision 3 application is measured in hundreds of bytes, a typical Java Card may be able to support dozens of instance of Calypso Revision 3.

Java Cards support the management protocols specified by the GlobalPlatform consortium. The GlobalPlatform specification defines the life cycle of the smartcard, the life cycle of an application, the mechanism to install, instantiate and delete an application in a smartcard, and the secure communication protocols to establish a connection between an instance of an application and a remote entity. Compared with proprietary mechanisms, those standard mechanisms are inherently more secure since their specifications are reviewed by independent experts. In many cases, the implementations are reviewed as well in order to achieve a security certification.

One of the first potential applications of the Java Card technology is to load multiple applications in a single smartcard. Today, many organisations already issue single-application smart cards in volume:

- Public transport operators,
- Banks,
- Governments (nationally with ID cards and passports or locally with “city cards”).

Loading more than one application in a single smartcard would cut the costs associated with issuing cards and would enable users to carry fewer cards. Public transport operators could delegate smartcard issuing to other actors, or they could act as the smartcard issuer and then rent areas of their smartcard to others.

New Form Factors

While multi-application Java Cards can be used to make smartcard issuing less expensive and to simplify the life of their users, *smart keys* and *mobile phones* actually deliver more functions than full-sized smartcards.

A *smart key* is a small USB key that embeds a fixed or removable smartcard and supports the ISO 7816 and ISO 14443 protocols. Therefore, it appears as a regular smartcard when presented to a contactless reader such as a validation gate. But the main advantage of a smart key over a traditional smartcard is that thanks to its USB interface, a user can plug it directly into a personal computer, while a traditional smartcard requires a specific card reader. This feature makes it is possible to establish a secure

communication link over the Internet between the smartcard and the smartcard's management server using nothing but the GlobalPlatform management protocols. This communication link enables the user to register and to buy new contracts simply and securely through the website of the transport operator, in a few seconds, even if the Calypso Revision 3 is not already loaded in the smart key. Furthermore, this mechanism would entice nothing but standard protocols:

- HTTP and SSL to communicate with the user's browser,
- The GlobalPlatform protocols to load and personalise a new application,
- The Calypso Revision 3 protocols to load the new contracts in the application.

The same functions can be delivered through mobile phones:

- SIM cards used in mobile phones are mostly Java Cards. This enables mobile network operators to load and personalise new applications remotely and to modify the behaviour of the mobile phone.
- Mobile phones are permanently connected to a network that may be used to deliver the new applications and the new contracts to the SIM card.
- Transport operators can leverage the user interface offered by mobile phones to communicate with the user.

A mobile phone is a more complex environment than a multi-application card and a smart key but it remains possible to use the standard Java Card protocols to manage an application running in a SIM card. Furthermore, this mechanism would enable a user to purchase new contracts anytime, anywhere. Some users may prefer to take advantage of the full-sized human interface provided by a personal computer to manage the applications stored in their mobile phones and to purchase new contracts. In this case, it remains possible to deliver the new applications and contracts remotely to the SIM card embedded in the mobile phone.

Finally...

Java Card is a mature and secure multi-application platform which is already used for billions of smartcards worldwide. Java Card implementations are available from a multitude of vendors, ensuring healthy competition. Moreover, all Java Cards comply with a well-established and public specification, which simplifies the comparison of the different smart cards. Vendors compete on performance, price, robustness, support for optional functions... An application developed according to the Java Card standard will work on every Java Card. Furthermore, Java Cards are also already available in a wide-range of form factors: Java Cards are not limited to the traditional "banking card" form factor but they are also present in mobile phones and smart keys.



SUMMARY

The objective of this document is to promote interoperable deployments of the Calypso Revision 3 application by standardizing the process required to download a Java Card implementation of Calypso Revision 3 on a Portable Object compatible with GlobalPlatform. This goal is achieved through three separate sections. After a short introductory section, the second section summarizes the context and the constraints that bear on this process and the third and final section defines a set of new, binding functional requirements for this process.

This document mandates specific solutions for the following issues:

How to secure the communication link between a service provider and the Portable Object?

The GlobalPlatform specification defines several secure communication protocols, each one with its own set of options. This document mandates **SCP02 option "i"= '55'** since it matches the requirements of the Calypso Revision 3 specification while being already widely available.

However, mobile phones represent a special case. Presently there is no official standard that would allow the creation of a communication channel protected with SCP02 between a remote entity and a Calypso application running in a SIM or (U)SIM card. However, several solutions have been proposed and are currently being standardized by the ETSI and GlobalPlatform organisations. This document mandates the **"SCP02 option "i"= '55' over SCP80"** mechanism.

How to personalize a Calypso Revision 3 remotely and securely?

A mechanism to pre-personalize a Calypso Revision 3 application is mandated, based on standard GlobalPlatform commands, completed by the Card Personalization Specification defined by EMV. Guidelines to handle errors during this process are provided.

Most notably, this document does not attempt to mandate a single application management scenario since such a scenario is highly dependant upon the specific deployment context of each transportation network. However, this question is discussed at length in an annex.

Finally, another annex demonstrates how to apply the same mechanisms to a Secure Application Module compatible with Calypso Revision 3 running in a GlobalPlatform environment.



1. INTRODUCTION

1.1. Scope

This document applies to the management (loading, installing, personalizing and deleting) of applications running in Portable Objects that embed a Secure Element compliant with the Java Card and GlobalPlatform specifications.

1.2. Intended Audience

This document is intended to assist any organization working on the remote management of Calypso Revision 3 applications:

- Service providers,
- Application developers,
- Card Management System developers,
- Card Management System managers,
- Trusted Third-Party managers
- Portable Object manufacturers,
- Secure Element manufacturers,
- Mobile Network Operators,
- ...

The reader is assumed to be familiar with the Java Card, GlobalPlatform and Calypso Revision 3 specifications.

1.3. Organisation

This document is composed of two main sections:

- The first section summarizes the constraints and the high-level security objectives that bear on the downloading phase of a Calypso Revision application.
- The second section specifies a set of binding functional requirements for the implementation of this process.



1.4. References

[CLS3]	<p>CALYPSO SPECIFICATION Portable Objects Application – Revision 3 Version 3.0 (dated 25 December 2006) Ref: 060708-CalypsoAppli Calypso Networks Association</p>
[SAMAC]	<p>Calypso Activation SAM – User manual Version 1.1 (dated 6 August 2008) Ref: 080118-UM-ActivationSam Calypso Networks Association</p>
[GP22]	<p>GlobalPlatform Card specification Version 2.2, March 2006 GlobalPlatform (www.globalplatform.org)</p>
[CCCM]	<p>GlobalPlatform Card Confidential Card Content Management Card Specification v 2.2 - Amendment A Version 1.0, October 2007 GlobalPlatform (www.globalplatform.org)</p>
[CPS]	<p>EMV Card Personalization Specification Version 1.1, July 2007 EMVCo (www.emvco.com)</p>
[24014]	<p>ISO 24014-1 Public transport – Interoperable fare management system Final Draft ISO (www.iso.org)</p>
[102225]	<p>ETSI TS 102 225 Smart Cards: Secured packet structure for UICC based applications (Release 7) ETSI (www.etsi.org)</p>
[102226]	<p>ETSI TS 102 226 Smart Cards: Remote APDU structure for UICC based applications (Release 7) ETSI (www.etsi.org)</p>
[IN-REQ]	<p>Requirements for loading a Calypso application Version 1.3 (dated 21 September 2007) Innovatron (www.innovatron.fr)</p>



1.5. Acronyms

APDU	Application Protocol Data Unit
APSD	Application Provider Security Domain
C-MAC	Command MAC
CA	Controlling Authority
CMS	Card Management System
DF	Dedicated File
DGI	Data Grouping Identifier
EF	Elementary File
EMV	Eurocard MasterCard Visa
FCI	File Control Information
GP	GlobalPlatform
GP-CMS	GlobalPlatform Card Management System
ICV	Initial Chaining Vector
ISD	Issuer Security Domain
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
LSB	Least Significant Byte
M/O	Mandatory/Optional
MAC	Message Authentication Code
MIDP	Mobile Information Device Profile
MSB	Most Significant Byte
MNO	Mobile Network Operator
OTA	Over-the-Air
OTI	Over-the-Internet
RAM	Remote Application Management
RFU	Reserved for Future Use
R-MAC	Response MAC
ROM	Read-Only Memory
SAM	Secure Application Module
SCP	Secure Channel Protocol
SD	Security Domain
SE	Secure Element
SFI	Short File Identifier
SIM	Subscriber Identity Module
SSD	Supplementary Security Domain
SW	Status Word
SWP	Single-Wire Protocol
TLV	Tag Length Value
TTP	Trusted Third-Party

1.6. Notations

All hexadecimal values are quoted. For instance, '10' (hexadecimal) represents 16 (decimal).

Topics outside of the scope of this document that may still be of interest are signalled with this specific format.

2. CONTEXT

This chapter presents the context of the downloading process i.e. pre-existing constraints and requirements that must be taken into account.

2.1. Calypso Elements

This document focuses on the following elements of the Calypso ecosystem:

Secure Element (SE):

A secure microprocessor small enough to be embedded in a smartcard (full-sized or SIM-sized). It is typically accessed through the ISO7816 interface but may support additional interfaces (USB, SWP, ISO14443, ...)

Portable Object (PO):

A portable object with an ISO14443 interface that hosts a SE. A SE is usually embedded in a smartcard but not always. Some Portable Objects use a fixed SE soldered to a motherboard (like USB Keys and some mobile phones) while others use removable SE (a mobile phone with a SIM that serves as a SE).

Examples:

- A contactless smartcard,
- A mobile phone with a NFC interface,
- An USB key with a NFC interface (also called a "Smart Key").

Calypso Application:

An instance created from a Calypso Revision 3 package installed in a SE.

Card Management System (CMS):

A piece of software that enables its users to generate management commands for a SE and to send them to the SE, remotely or locally. It can be used to modify or retrieve the data stored in the SE, or to trigger a certain operation within the SE. A GP-CMS uses the protocols defined by GlobalPlatform to manage the SE.

A CMS is sometimes referred by another name: TSM. However, even if this term is widely used, it usually means different things to different people. For example, the same term is also used to designate a role: an entity responsible for the administration of a smart card.
This document will not use the TSM term.

2.2. Calypso Roles

The Calypso ecosystem defines a number of roles:

- Applet Designer: Calypso Licensee who creates the Calypso Revision 3 package according to the Calypso specifications.
- Applet Certifier: Certifies that a Calypso Revision 3 package complies with Calypso.
- Applet Loader: Loads the package into Portable Objects.
- Applet Installer: Creates the Calypso application from the package and makes it accessible.
- Applet Initializer: Initializes the Calypso application, sets a unique serial number, an activation key and a file structure and activates it by loading Calypso parameters.
- Applet Personalizer: Loads the keys into the Portable Object with the application owner SAM-CPP.
- PO Holder: End user holding the Portable Object.
- Application Owner: Possesses the Calypso application operational keys.
- SE Issuer: Top manager of the SE, ultimately responsible for its security.

2.3. Security Objectives

[IN-REQ] defines a number of security objectives for the downloading phase of the Calypso application:

For the Applet Loader

- AL.AUTH.OP The authenticity of the portable object must be verified (for example by using a "Security Domain" with confidential keys).
- AL.AUTH.APP The authenticity of the Calypso applet must be verified before every loading (verification of the authenticity of the Calypso signature).
- AL.CONF.APP The applet must be kept confidential.
- AL.LOAD.APP The applet must be loaded confidentially (in a physically secure environment, or, if remotely loaded, with a protocol ensuring a confidentiality and authentication with a security level high enough).

For the Applet Installer & Initializer

- AI.CONF.SAM The SAM (Activation and CPP) must be kept confidential.
- AI.CONF.KEYS The data produced by the SAM must be kept confidential and only transmitted through a secure channel approved by Calypso.
- AI.LOAD.KEYS The operational keys loaded must be the one indicated by the owner of the SAM CPP.
- AI.LOAD.DATA The initial data loaded must be correct.

2.4. Compatibility with ISO24014

A partial mapping from the roles used in this document to the ones introduced in [24014] is summarized in the table below. ISO 24014 is a very high-level and generic standard. It can apply to any transport system that relies on products hosted on a media carried by a user. It is therefore not possible to precisely map every entity defined in [24014] to a Calypso Role.

This document	ISO24014
Applet	Media
Applet Designer	Not applicable
Applet Certifier	Security Manager (partially)
Applet Loader	Application Retailer (partially)
Applet Installer	Application Retailer (partially)
Applet Initializer	Application Retailer (partially)
Applet Personalizer	Application Retailer (partially)
PO Holder	Customer
Application Owner	Application Owner (partially)

2.5. Form Factors

This document is designed to be independent from the final form factor of the Portable Objects. Decisions or details that depend upon the actual form factor should be deferred as much as possible to the implementation phase.

Three different form factors are targeted at the moment:

- full-size smartcards,
- USB smart keys,
- SIM cards running in mobile phones.



3. FUNCTIONAL REQUIREMENTS

This chapter details the mandatory functional requirements for the downloading process.

Within this chapter, the keywords *shall*, *should* and *may* are to be interpreted as follows, when used in the italic form:

- The word *shall* is used to indicate mandatory requirements strictly to be followed in order to avoid common error situations and to avoid situations where the security or the operation of a Calypso application may be compromised. No deviation to such indications should be permitted (*shall equals is required to*).
- The word *should* is used to indicate that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is not recommended but not strictly prohibited (*should equals is recommended that*).
- The word *may* is used to indicate a course of action permissible within the limits of the platform (*may equals is permitted to*).

3.1. Java Card Support

The targeted SE *shall* support Java Card 2.2.1 or a more recent version.

3.2. GlobalPlatform Support

Non-(U)SIM SEs *should* support GlobalPlatform 2.2 with the extended `processData` method defined in [CCCM] but *may* support only GlobalPlatform 2.1.1.

(U)SIM SEs *shall* support GlobalPlatform 2.2 with the extended `processData` method defined in [CCCM].

3.3. Secure Communication

For non-(U)SIM SEs, the APDUs exchanged between the Application Provider and the Application Provider Security Domain *shall* be secured with SCP02 option "i"= '55'.

For (U)SIM SEs, the APDUs exchanged between the Application Provider and the Application Provider Security Domain *shall* be secured with SCP02 option "i"= '55'. If the SE does not support this SCP, the APDUs exchanged between the Application Provider and the Application Provider Security Domain *shall* be secured with SCP02 option "i"= '55' over SCP80.

The Application Provider *shall* own the encryption keys K_{ENC} , K_{MAC} and K_{DEK} used to secure the APDUs it exchanges with its APSD.

The security level for SCP02 option "i"= '55' *shall* be set to MAC+ENC for every command.

Every APSD in a SE *shall* use a different key set. The key set of every APSD *shall* be diversified by SE.

3.4. Security Domain Creation

A specific creation method for the Security Domain that hosts the Calypso application is not mandated since it must be tailored to the environment of the Calypso application.

3.5. Calypso Application Management

A complete management scenario for the Calypso application is not mandated since it must be tailored to the environment of the Calypso application. This section is limited to the functional requirements of the installation and pre-personalisation phases.

3.5.1. Calypso Application States

The application *shall* manage two independent life cycle states: its GlobalPlatform state and its internal state.

3.5.1.1. GlobalPlatform State

The GlobalPlatform state of the Calypso application *shall* take one of the following values:

- APPLICATION_INSTALLED ('03');
- APPLICATION_SELECTABLE ('07');
- LOCKED from APPLICATION_INSTALLED ('83');
- LOCKED from APPLICATION_SELECTABLE ('87').

The Calypso application *shall not* manage the transitions between those states. The acceptable transitions between states are defined in [GP22].

3.5.1.2. Internal State

In addition to the GlobalPlatform state, the Calypso application *shall* manage an internal state that *shall* take one of the following values:

- NOT_PERSONALIZED ('CC');
- PERSONALIZING ('5A');
- PERSONALIZED ('A5').

When the Calypso application internal state is NOT_PERSONALIZED, the SELECT [application] command returns a FCI with only the DF Name and status word '9000'.

When the Calypso application internal state is PERSONALIZING, the SELECT [application] command returns a FCI with only the DF Name and status word '6283'.

When the Calypso application internal state is PERSONALIZED, the SELECT [application] command returns a FCI with the DF Name and the FCI proprietary template, and status word '9000' (if the Transport DF is validated) or '6283' (if the Transport DF is invalidated).

The acceptable transitions between states are detailed in the following sections.

The internal life cycle of a Calypso application *shall* be independent of the internal life cycle of every other application, even applications instantiated from the same package.

3.5.2. Installation

For SEs with support for [GP22] and support for the INSTALL [for install] command, the installation *shall* be performed with the GlobalPlatform INSTALL [for install] command. In this case, the final GlobalPlatform state of the Calypso application will be APPLICATION_INSTALLED.

For SEs with support for [GP22] but without support for the INSTALL [for install] command, the installation *shall* be performed with the GlobalPlatform INSTALL [for install and make selectable] command. In this case, the final GlobalPlatform state of the Calypso application will be APPLICATION_SELECTABLE.

For the other types of SEs, the installation *shall* be performed with the GlobalPlatform INSTALL [for install and make selectable] command. In this case, the final GlobalPlatform state of the Calypso application will be APPLICATION_SELECTABLE.

The Calypso application *shall not* require any privilege. It *shall* be installed with no privileges (that is '00').

The Calypso application *shall not* require any application-specific parameters (tag 'C9' of the INSTALL [for install] command). A zero-length value *shall* be passed to the command.

The Calypso application *shall not* require any system-specific parameters (tag 'EF' of the INSTALL [for install] command). This tag *shall not* be passed to the command.

When the installation is complete, the initial Calypso application internal state *shall* be NOT_PERSONALIZED.

Initially, the value of the Calypso Serial Number of the Calypso application *shall* be set to 0 ([CLS3], R8) and the value of the transaction counter *shall* be 200.000 or more ([CLS3], R68).

3.5.3. Pre-personalisation and Activation

This section defines the standard mechanism used to implement the Pre-personalisation and Activation process defined in [IN-REQ]:

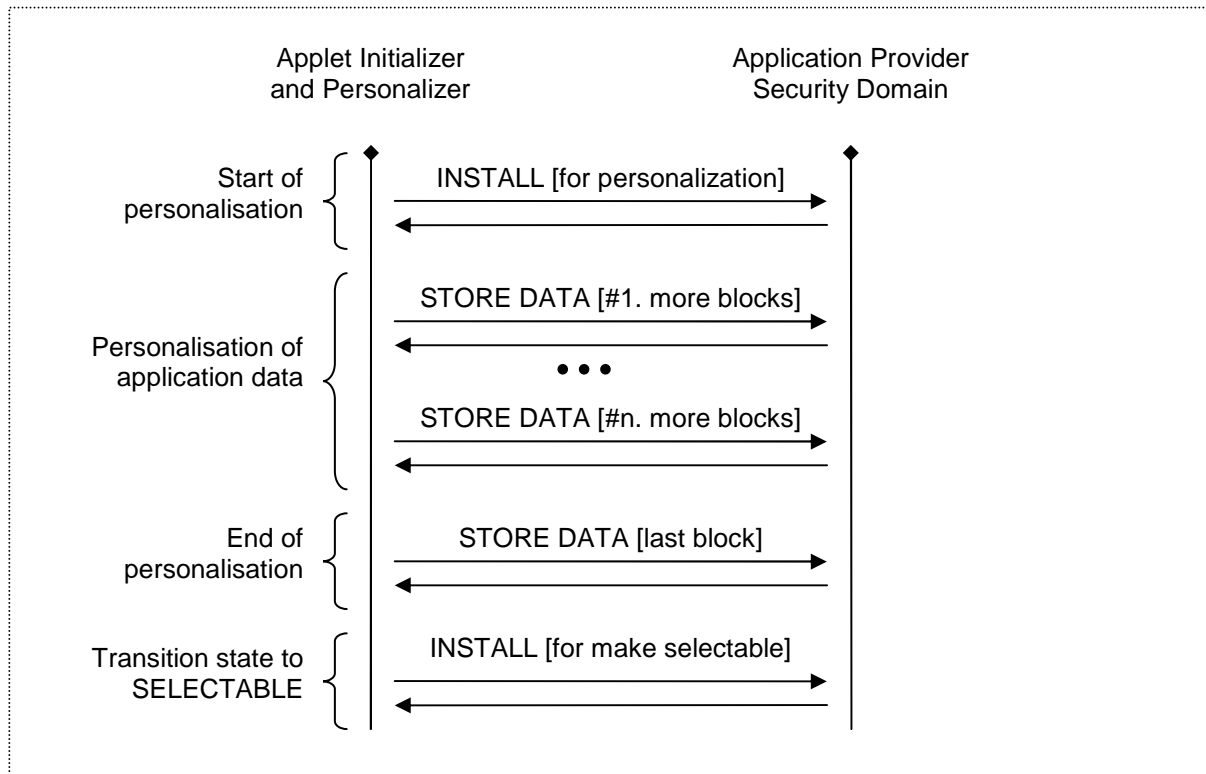
- Assigning the Calypso Serial Number,
- Loading Calypso parameters,
- Initializing the file structure,
- Loading the applicative Keys,
- Optionally, loading the initial personalisation data.

The personalisation process *shall* follow [CPS], using the DGI described in section 3.6.

Initially, the internal state of the Calypso application *shall* be NOT_PERSONALIZED. It *shall* transition to PERSONALIZING when the Calypso application receives the first DGI different from 9484, 84F2 or 8185. It *shall* transition to PERSONALIZED when the personalisation process completes successfully i.e. when the last DGI is successfully processed.

3.5.3.1. Nominal Case

For SEs with support for [GP22] with the extended `processData` method defined in [CCCM], the personalisation *shall* be performed through a Security Domain. This is the only acceptable process for (U)SIM cards. This process *shall* follow the APDU commands in the figure below:

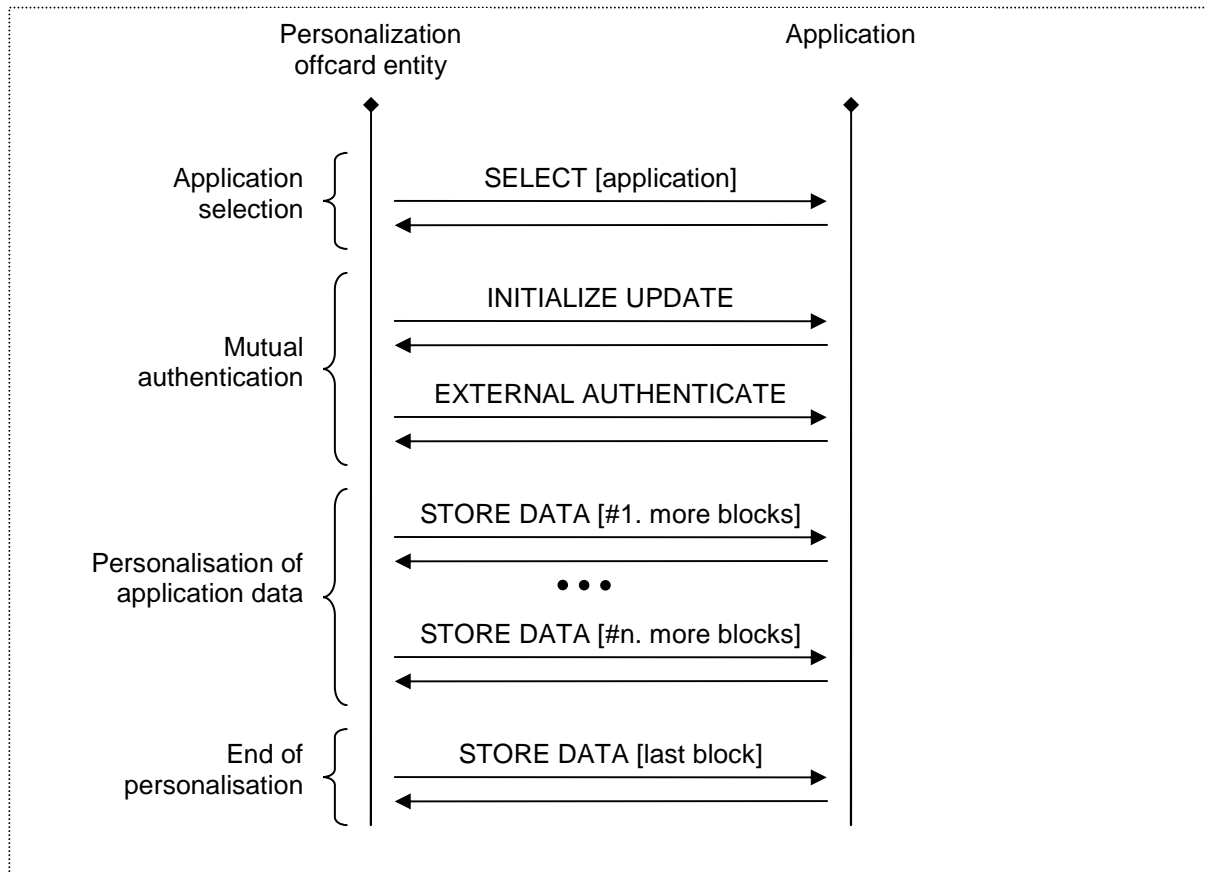


This process *should* take place after a successful installation of the Calypso application, but it *may* also take place after a mutual authentication with the APSD.

If the initial GlobalPlatform state of the Calypso application is APPLICATION_SELECTABLE, the final INSTALL [for make selectable] *shall not* be sent.

3.5.3.2. Support for older GlobalPlatform Specifications

For SEs without support for [GP22] with the extended `processData` method defined in [CCCM], the personalisation *shall* be performed through the Calypso application. The initial GlobalPlatform state of the Calypso application *shall* be APPLICATION_SELECTABLE. This process *shall* follow the APDU commands in the figure below:



3.6. Data Grouping Identifiers

This section defines the format and content of the DGI that *shall* be used to personalize the Calypso application with STORE DATA commands.

3.6.1. Definition

All DGI tags *shall* be 2-byte long.

The following DGI tags *shall* be sent only once and in the order of the table:

DGI	Description	Encrypted	Return Data	M/O	Span
'9484'	Application challenge	No	Yes	Mandatory	No
'2000'	Application Serial Number	No	No	Mandatory	No
'2100'	Startup Information	No	No	Mandatory	No
'2200'	Application Transaction Counter	No	No	Optional	No
'3100'	File Structure of the Calypso application	No	No	Mandatory	Yes
'3800'	Calypso Activation Parameters	Yes	No	Mandatory	No
'4101'	Initial value of the Issuer key	Yes	No	Mandatory	No
'4102'	Initial value of the Load key	Yes	No	Mandatory	No
'4103'	Initial value of the Debit key	Yes	No	Mandatory	No
'4104'	Initial value of the PIN	Yes	No	Optional	No
'ssnn'	Initial value of file contents	No	No	Optional	Yes

The following DGI tags *may* be sent at any time:

DGI	Description	Encrypted	Return Data	M/O	Span
'8185'	Traceability information	No	Yes	Optional	No
'84F2'	Application status	No	Yes	Optional	No

Each DGI tag *shall* be followed by a length indicator coded as follows:

- On 1 byte if the length of the data is in the range '00' to 'FE' (0 to 254),
- On 3 bytes with the first byte set to 'FF' followed by a 2-byte value in the range '0000' to 'FFFE' (0 to 65534), e.g. 'FF01AF' indicates a length of '1AF' bytes (431 bytes).

The DGI tag and the DGI length *shall* be sent in the same APDU command (they cannot span over two or more APDU commands).

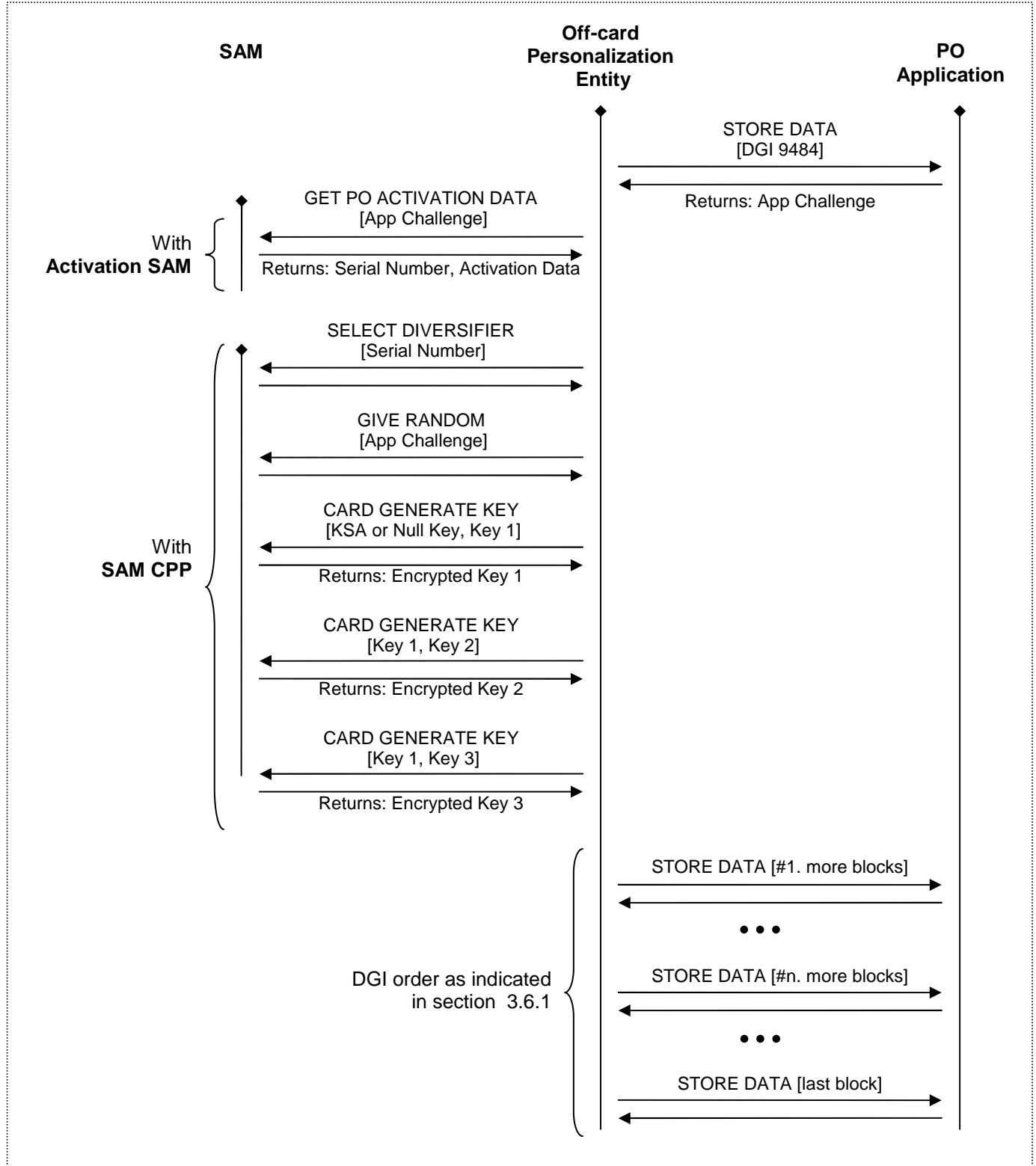
The DGI values for tags '3100' and 'ssnn' *may* span over several APDU commands (as specified in the "Span" column of the table above). The other DGI values *shall not* span over two or more APDU commands.

The last STORE DATA command *shall* have the P1.b8 bit set to 1 to signal that it is the last STORE DATA (in accordance with [GP22]).



3.6.2. Cinematic

The Pre-Personalisation and Activation cinematic is illustrated below:



3.6.3. Data content for DGI '9484'

This DGI asks the Calypso application to return a 20-byte record:

Len.	Name	Description
4	Application Reference	Identifies the applet issuer and the KSA key
16	Challenge	Random number used in subsequent DGIs

This DGI *shall* be the first one sent to the Calypso application during the pre-personalisation process.

The STORE DATA command *shall*:

- have its bit P1.b1 set to 1 (to allow the answer to be returned),
- contain only one DGI (of tag '9484') which *shall* be empty (its length must be set to '00').

DGI '9484' *shall not* transition the Calypso application to the PERSONALIZING state.

The returned challenge *shall* remain valid even if the SE being personalized is reset.

DGI '9484' *shall not* be sent once the Calypso application has reached the PERSONALIZING state.

3.6.4. Data content for DGI '2000'

This DGI contains the 8-byte long unique Calypso Serial Number that is assigned to the Calypso application, as returned by the GET ACTIVATION DATA command sent by the Activation SAM.

Off.	Len.	Name	Description
0	8	Calypso Serial Number	Calypso Serial Number

3.6.5. Data content for DGI '2100'

This DGI contains the *Startup Information*, as defined in [CLS3]:

Off.	Len.	Name	Description
0	1	Session modifications	Maximum number of modifications in a session (value between 6 and 55)
1	1	Platform (Chip type)	Type of platform
2	1	Application type	Application type
3	1	Application subtype	Application subtype
4	1	Software issuer	Software issuer reference
5	1	Software version (ROM version)	RFU (<i>shall</i> be set to '0')
6	1	Software version (EEPROM version)	RFU (<i>shall</i> be set to '0')

The contents of this DGI *shall* be used by the Calypso application to build the answer to the SELECT [application] command and to the GET DATA command.

The maximum number of modifications in a session is limited by the amount of memory made available by the SE to the Calypso application. If the amount of available memory is not large enough to hold at least 6 modifications in a session, the pre-personalisation process *shall* fail. If the amount of RAM available is large enough to hold at least 6 modifications in a session but not large enough to hold the specified number of modifications in a session, the pre-personalisation *shall* succeed. In this case the number of supported modifications in a session reported in the "Discretionary Data" field of the answer to the SELECT [application] command *shall* be reduced by the Calypso application to report the actual number of supported modifications in a session.

3.6.6. Data content for DGI '2200'

This DGI contains the initial value for the transaction counter (MSB first).

This DGI is optional. It *may* be used to supersede the default initial value of the transaction counter.

The Calypso application *shall not* verify the number given in this DGI. The pre-personalisation and activation process *shall* ensure that the value given in this DGI is 200.000 or above.

Off.	Len.	Name	Description
0	3	Transaction Counter	Initial value for the transaction counter

3.6.7. Data content for DGI '3100'

This DGI contains the file structure and access conditions of the files present in the Calypso application.

The DGI *shall* be made of a sequence of "File Descriptors", immediately one after the other:

- The first file descriptor shall describe a DF,
- The next file descriptors shall describe only EFs.

Off.	Len.	Name	Description
0	18	File Descriptor	DF file descriptor
18	18	File Descriptor	First EF file descriptor
...
n x 18	18	File Descriptor	Last EF file descriptor

The EFs *shall* initially be filled with zeroes ([CLS3], R2).

Each file descriptor *shall* be 18 bytes long, and *shall* be formatted as follows:

Off.	Len.	Name	Description	Values
0	2	LID	File identifier	
2	1	SFI	Short File Identifier	DF: Shall be '00' EF: In range '00' to '1E' ('00' means no SFI)
3	1	FileType	File type	'02': DF '04': EF
4	1	EFType	EF Type	'00': DF '04': Cyclic file '01': Binary file '08': Simulated Counter file '02': Linear file '09': Counters file
5	1	RecSize	Record size	DF: Must be '00' EF (binary): File size MSB EF (others): Must be greater than '00'
6	1	NumRec	Number of records	EF (binary): File size LSB EF (others): Must be greater than '00'
7	4	AC	Access Conditions	Each byte among: '00': NEVER '10': SESSION '01': PIN '1F': ALWAYS
11	4	NKey	Key numbers	Each byte among: '00': No Key (if AC is not SESSION) '01': Issuer key '02': Load key '03': Debit/Validation key
15	2	DataRef	DataReference	Simulated Counter: SFI of counter file and 1-based counter number Shared EF: Unique identifier DF/Other EF: '0000'
17	1	DFStatus	Status of the DF	DF (validated): '00' DF (invalidated): '01' EF: '00'

3.6.8. Data content for DGI '3800'

This DGI contains the Calypso Activation Parameters, as given by the Activation SAM.

Off.	Len.	Name	Description
0	var.	Activation Parameters	See [SAMAC]

This DGI *shall* be encrypted with K_{dek} .

3.6.9. Data content for DGI '410n'

These DGI contain the initial value of the keys (and optionally of the PIN) associated with the Calypso application.

The data field of this DGI *shall* contain the same data as the Data field of a Calypso CHANGE KEY command.



Off.	Len.	Name	Description
0	var.	Encrypted Key	Identical to the Data Field of an equivalent CHANGE KEY command

This DGI *shall* be encrypted with K_{dek} .

3.6.10. Data content for DGI 'ssnn' ('01nn' to '1Enn')

These DGI contain the initial value of file records. They are optional.

The 'ss' part of the DGI is a file SFI, where 'ss' *shall* be in the range '01' to '1E'. The 'ss' value *shall* refer to a file SFI listed in DGI '3100' of file type: binary, linear, cyclic or counters.

The 'nn' part of the DGI *shall* be:

- '00' if the file is a binary file,
- a record number between 1 and the number of records in case of a linear file, a cyclic file or a counters file.

Files without a SFI *shall not* be personalised.

Bytes are written starting from the beginning of the file record (or the beginning of the file for binary files). The number of bytes to be written shall be less than or equal to the record length (file length for binary files). If less bytes are provided, only the beginning of the record (or file) shall be written.

3.6.11. Data content for DGI '8185'

This DGI asks the Calypso application to return the Traceability Information defined [CLS3].

The STORE DATA command *shall*:

- have its bit P1.b1 set to 1 (to allow the answer to be returned),
- contain only one DGI (of tag '8185') which *shall* be empty (its length must be set to '00').

DGI '8185' *shall not* transition the Calypso application to the PERSONALIZING state.

3.6.12. Data content for DGI '84F2'

This DGI asks the Calypso application to return the GlobalPlatform and internal application state.

Tag	Len.	Name	Description
'9F70'	1	GlobalPlatform Life Cycle State	See section 3.5.1.1
'9FDC'	1	Internal Life Cycle State	See section 3.5.1.2

Knowing the GlobalPlatform and internal state enables the Application Personalizer to recover from a communication failure. Therefore, this DGI *should not* be used during the standard pre-personalisation process.

The STORE DATA command *shall*:

- have its bit P1.b1 set to 1 (to allow the answer to be returned),
- contain only one DGI (of tag '84F2') which *shall* be empty (its length must be set to '00').

DGI '84F2' *shall not* transition the Calypso application to the PERSONALIZING state.

3.7. Error Handling

This section describes the process that *shall* be followed if an error is detected during the installation and personalisation phases. This process is illustrated by *Figure 1: Error-handling Process*.

3.7.1. Rules

Once installed, the Calypso application *shall* be ready to receive STORE DATA commands or calls to the `processData` method.

If an error occurs during the installation process, the CMS *may* use the GET STATUS command to query the GlobalPlatform state of the SE and the GlobalPlatform state of the Calypso application.

If an error occurs during the personalisation process, the CMS *may* use the GET STATUS command to query the GlobalPlatform state of the SE and the GlobalPlatform state of the Calypso application. The CMS may also use the '84F2' DGI to query the internal state of the Calypso application.

From the moment when the Calypso application enters the PERSONALIZING internal state, the Calypso application *shall* be automatically locked if it receives an event different from:

- A call to the `processData` method if the personalisation is performed through the APSD,
- A STORE DATA command if the personalisation is performed through the Calypso application.

For example, the Calypso application *shall* be automatically locked if it receives one of the following events:

- a command different from STORE DATA if the personalisation is performed through the Calypso application,
- an implicit or explicit selection of any Calypso application from the same package on any logical channel,
- an explicit deselection of any Calypso application from the same package on any logical channel.

Additionally, the Calypso application *shall* be automatically locked if an error occurs while processing a STORE DATA command if the personalisation is performed through the Calypso application or a call to the `processData` method if the personalisation is performed through the APSD.

When the Calypso application is locked during the personalisation process:

- SELECT [application] *shall* return the normal FCI of the Calypso application followed by '6283'.
- Any other command *shall* return '6985'.
- It *shall not* be possible to unlock the Calypso application.

If the installation and personalisation process fails, it *shall* be recoverable by deleting the Calypso application's package and restarting the installation process, starting with the INSTALL [for load] command, possibly with a different version of the Calypso application's package and/or different installation and personalisation parameters.

3.7.2. Cinematic

This section details the decision diagram presented in *Figure 1: Error-handling Process*. The process presented in this section does not detail the commands required to establish a SCP with the targeted APSD.

3.7.2.1. INSTALL [for load]

If the execution of the INSTALL [for load] command returns a Status Word that indicates that an error occurred, the CMS *should* attempt to send the same command again, possibly with a different set of parameters. Otherwise the process has failed.

If the connection to the SE is lost during the execution of the INSTALL [for load] command, the process has failed.

3.7.2.2. **LOAD**

If the execution of one of the LOAD commands returns a Status Word that indicates that an error occurred, the process has failed.

If the connection to the SE is lost during the execution of the LOAD commands, the CMS *should* reopen the SCP channel to the SE then issue a GET STATUS command to determine whether the Calypso application's package was successfully loaded or not. Otherwise the process has failed.

If the Calypso application's package was successfully loaded, the CMS *shall* proceed to the next command. If the Calypso application's package was not successfully loaded, the process has failed.

3.7.2.3. **INSTALL [for install]**

If the execution of the INSTALL [for install] command returns a Status Word that indicates that an error occurred, the CMS *should* attempt to send the same command again, possibly with a different set of parameters. Otherwise the process has failed.

If the connection to the SE is lost during the execution of the INSTALL [for install] command, the CMS *should* reopen the SCP channel to the SE then issue a GET STATUS command to determine whether the Calypso application was successfully installed or not. Otherwise the process has failed.

If the Calypso application was successfully installed, the CMS *shall* proceed to the next command. If the Calypso application was not successfully installed, the CMS *should* send the INSTALL [for install] command again with the same set of parameters. Otherwise the process has failed.

3.7.2.4. **INSTALL [for personalization]**

If the execution of the INSTALL [for personalization] command returns a Status Word that indicates that an error occurred, the CMS *should* attempt to send the same command again, possibly with a different set of parameters. Otherwise the process has failed.

If the connection to the SE is lost during the execution of the INSTALL [for personalization] command, the CMS *should* reopen the SCP channel to the SE then resend the INSTALL [for personalization] command with the same set of parameters. Otherwise the process has failed.

3.7.2.5. **STORE DATA**

If the execution of a STORE DATA command returns a Status Word that indicates that an error occurred, the CMS *should* analyse the state of the Calypso application, possibly by sending a STORE DATA command holding a single '84F2' DGI. Otherwise the process has failed.

If the connection to the SE is lost during the execution of a STORE DATA command, the CMS *should* reopen the SCP channel to the SE then analyse the state of the Calypso application, possibly sending a new INSTALL [for personalization] command with the same AID parameter and a STORE DATA command holding a single '84F2' DGI. Otherwise the process has failed.

For both failure modes:

- If the internal state of the Calypso application is NOT_PERSONALIZED, the CMS should attempt to resend every STORE DATA command. Otherwise the process has failed.
- If the internal state of the Calypso application is PERSONALIZING, the application is locked.
- If the internal state of the Calypso application is PERSONALIZED, the CMS shall proceed to the next command.

If the Calypso application is locked, the CMS *shall* attempt to delete the Calypso application. If the DELETE command is successful, the CMS *should* restart the installation process by resending the INSTALL [for install] command. If the DELETE command returns a Status Word that indicates that an error occurred, the process has failed.

3.7.2.6. **INSTALL [for make selectable]**

If the execution of an INSTALL [for make selectable] command returns a Status Word that indicates that an error occurred, the CMS *should* attempt to send the INSTALL [for make selectable] command again, possibly with a different set of parameters. Alternatively the CMS *may* attempt to delete the Calypso application instead. Otherwise the process has failed.

If the DELETE command is successful, the CMS *should* restart the installation process by resending the INSTALL [for install] command. If the DELETE command returns a Status Word that indicates that an error occurred, the process has failed.

If the connection to the SE is lost during the execution of the INSTALL [for make selectable] command, the CMS *should* reopen the SCP channel to the SE then issue a GET STATUS command to determine whether the Calypso application was successfully transitioned to the SELECTABLE state or not. Otherwise the process has failed.

If the GlobalPlatform state of the Calypso application is INSTALLED, the CMS *should* attempt to send the INSTALL [for make selectable] command again, with the same installation parameters. Alternatively the CMS *may* attempt to delete the Calypso application instead. Otherwise the process has failed.

If the DELETE command is successful, the CMS *should* restart the installation process by resending the INSTALL [for install] command. If the DELETE command returns a Status Word that indicates that an error occurred, the process has failed.

If the internal state of the Calypso application is PERSONALIZED, the Calypso application is ready to be used.

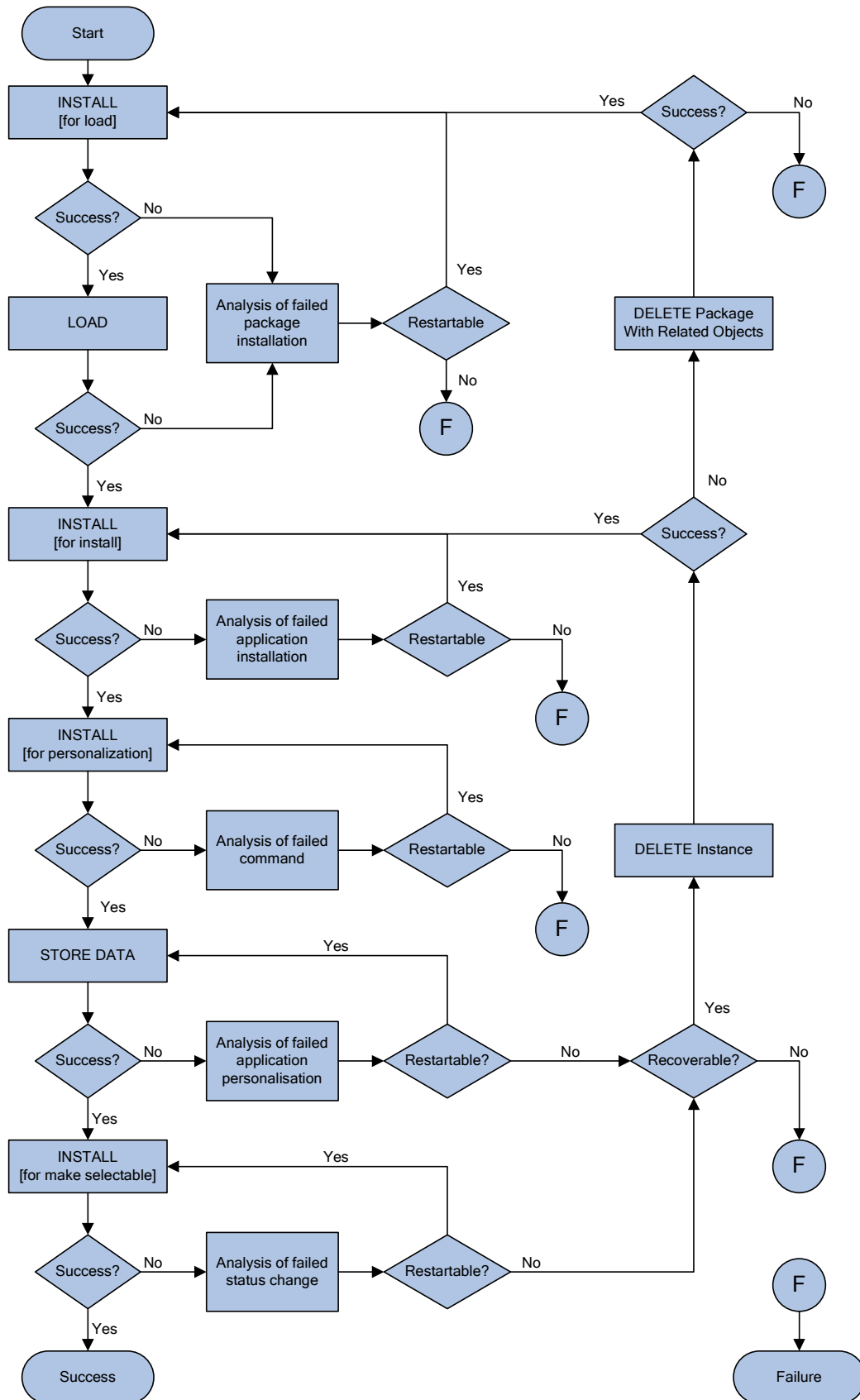


Figure 1: Error-handling Process

4. ANNEX A: SAM APPLICATION

The requirements presented in the previous sections were aimed at Portable Objects, i.e. portables devices used by customers to store Calypso applications. Still, those requirements are generic enough to be applied to other kinds of applications. This section presents a set of functional requirements for the downloading of a Calypso Revision 3 SAM application in a SE.

4.1. Data Grouping Identifiers

This section defines the format and content of the DGI used to personalize the application with STORE DATA commands.

4.1.1. Definition

All DGI tags *shall* be 2-byte long.

The following DGI tags are defined and ***shall*** be sent in the order of the table:

DGI	Description	SAM
'9484'	Application challenge	Mandatory
'2000'	Application Serial Number	Mandatory
'2100'	Startup Information	Mandatory
'3800'	Calypso Activation Parameters	Mandatory
'4110'	Initial SAM personalisation	Optional

Each DGI tag *shall* be followed by a length indicator coded as follows:

- On 1 byte if the length of the data is in the range '00' to 'FE' (0 to 254).
- On 3 bytes with the first byte set to 'FF' followed by a 2-byte value in the range '0000' to 'FFFE' (0 to 65534), e.g. 'FF01AF' indicates a length of '1AF' bytes (431 bytes).

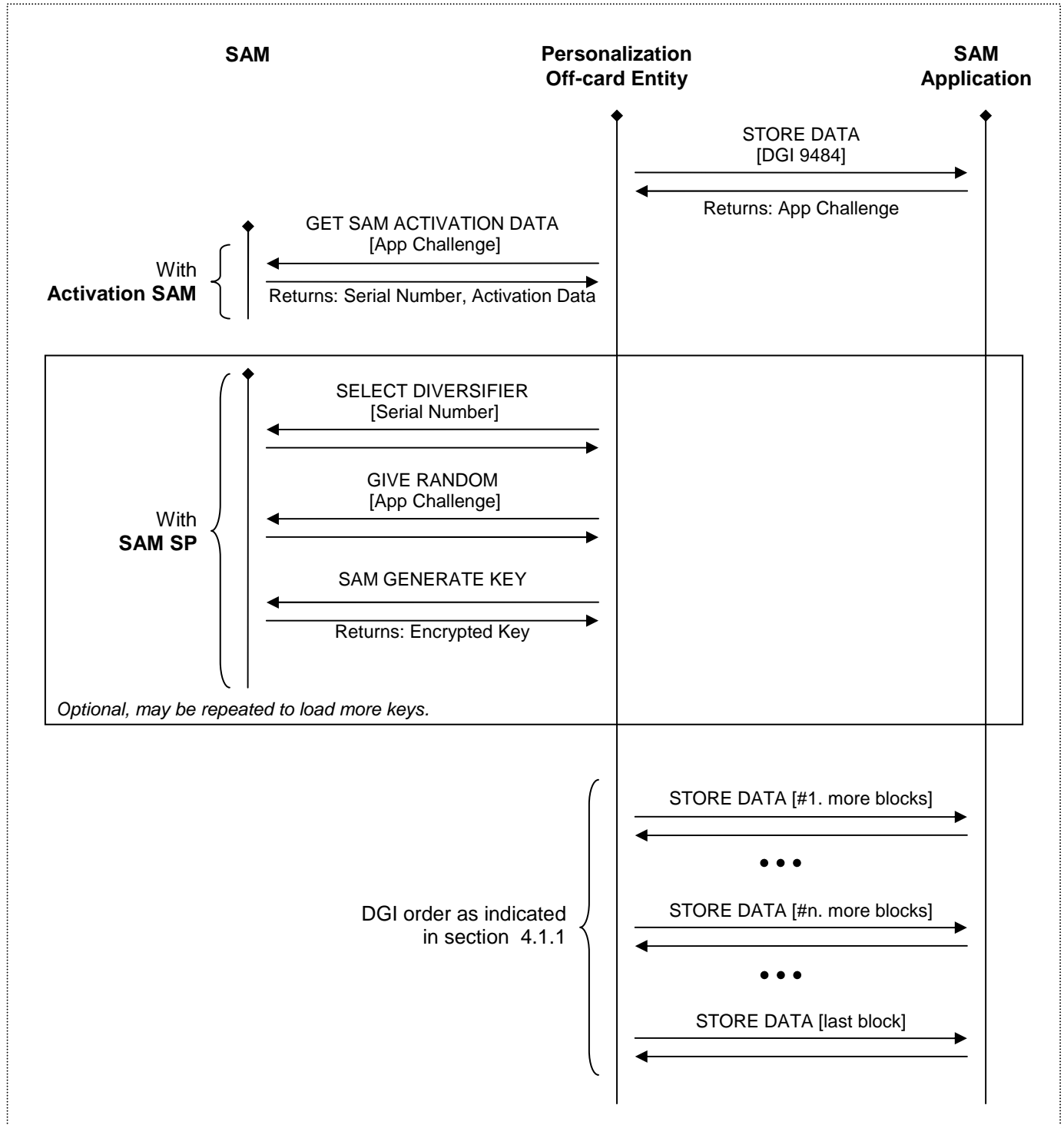
The DGI tag and the DGI length *shall* be sent in the same APDU command (they *shall not* span over two or more APDU commands).

A DGI (tag, length and value) *shall* not span over two or more APDU commands.



4.1.2. Cinematic

The personalisation cinematic is illustrated below:



4.1.3. Data content for DGI '4110'

This DGI contains a SAM application APDU (CLA, INS, P1, P2, Lc, Data In) to be processed during the personalisation.



The APDU *may* be only one of the following:

- SAM WRITE CAAD
- SAM WRITE CEILINGS
- SAM WRITE KEY
- SAM WRITE PARAMETERS

Off.	Len.	Name	Description
0	53	Personalisation APDU	APDU for the command SAM WRITE CAAD, CEILINGS, KEY or PARAMETERS.

The SAM application *shall* process the APDU as usual, with the following exceptions:

- The SAM challenge, normally obtained with Get Challenge, *shall* be the challenge previously returned by the DGI '9484' of the Calypso Revision application being personalized and *shall* not change until the end of the personalisation (the challenge remains valid after each DGI '4110' received).



5. ANNEX B: DISCUSSION & RATIONALES

The Functional Requirements chapter presented the functional requirements for the downloading process. This chapter presents and discuss the rationales that sustain those functional requirements.

This chapter also presents several mechanisms to securely install and manage an application in a Portable Object. Those mechanisms are then used by several management scenarios. The scenarios presented in this document are based on existing and planned deployments scenarios. Multiple management scenarios are required to address a variety of situations:

- In a standard smart card, in a USB key, in a mobile phone...
- Various degrees of involvement of the service provider in the management of the Portable Object.
- Various levels of support for the GlobalPlatform specifications from the Portable Object.

5.1. GlobalPlatform Support

The GlobalPlatform specification is a large and flexible standard. It defines several optional features to address the requirements of different market segments, so a single SE usually does not to implement all the features defined in the specification.

This document presents several potential deployment scenarios for this technology. For each scenario, the necessary features are spelt out. Nevertheless, those scenarios are not exhaustive and the exact requirements shall be discussed at the deployment phase according to the Application Provider use cases. For example, the requirements in terms of supported GlobalPlatform features for the deployment of the Calypso Revision 3 application in a USB key are different from the requirements for the deployment of the same application in a (U)SIM card.

5.2. GlobalPlatform Roles

Any deployment of the GlobalPlatform technology always involves two disjoint sets of roles: the GlobalPlatform roles and the industry-specific roles. GlobalPlatform roles are defined in [GP22] while roles for the Calypso ecosystem are defined in Section 2.2. The mapping between the GlobalPlatform roles and the Calypso roles may change according to the requirements of each individual deployment of the Calypso Revision 3 application.

[GP22] defines three roles:

- Issuer: the entity responsible for the security of the whole SE. It is represented in the SE by the Issuer Security Domain (ISD).
- Application Provider: an entity responsible for the behaviour of one or more applications. It is represented in the SE by an Application Provider Security Domain (APSD).
- Controlling Authority: an optional entity represented in the SE by a Controlling Authority Security Domain (CASD). It enforces the security policy on all application code loaded to the card and enable an Application Provider to manage the keys of its APSD in a manner confidential from all actors except the Application Provider itself...

The Issuer authenticates itself to the ISD through a set of keys, the Issuer Security Domain Keys. An Application Provider authenticates itself to its APSD through a set of keys, the Application Provider Security Domain Keys.

The management scenarios described in Section 5.4.2, Application Management, define several mappings between GlobalPlatform roles and Calypso roles according to the constraints and requirements of each scenario.



5.3. Secure Communication

The objective of this section is to identify the mechanisms that meet the security requirements expressed in Section 2.3. Those mechanisms must provide the means to secure the exchange of management commands between an Application Provider and its APSD. A difference is made between the nominal case and the case of the mobile phone, since this latter case presents specific challenges.

The standard mechanism to communicate securely with a GlobalPlatform SE is to use a Secure Channel Protocol (SCP). A SCP ensures the mutual authentication of both the SE and the Application Provider and protects the APDU exchanged between them (over a logical channel) by attaching cryptographic protections (signature and encryption) to each APDU. A SCP provides protection against message corruption, replaying, permutation or deletion. A SCP may optionally provide message confidentiality, if so is required.

5.3.1. Nominal Case

Every application in a GlobalPlatform SE is assigned to a Security Domain (SD). An off-card entity can securely communicate with a SD using a SCP over a logical channel.

[GP22] defines several SCP and many options for each of them. The most common configuration is known as SCP02 option "i"= '55' (*"Initiation mode explicit, C-MAC on modified APDU, ICV set to zero, ICV encryption for C-MAC session, 3 Secure Channel Keys, well-known pseudo-random algorithm (card challenge), no R-MAC"*). SCP02 option "i"= '55' satisfies the requirement placed on the security of the message exchange.

SCP02 is designed to protect APDUs exchanged over the ISO7816 or ISO14443 interface. For example, it can be used to meet the requirements of the following use cases:

- To initialise the SE during the manufacturing phase,
- To personalise the SE (to load the keys for the ISD for example),
- To communicate locally with the SE when it is physically connected to a SE reader,
- To communicate OTI with the SE when the SE is embedded in an USB key connected to a PC.

The SCP02 protocol is made of two main phases:

1. Mutual Authentication,
2. Command Exchange.

During the Mutual Authentication step, the SE and the CMS derive the three secure channel session keys and exchange two cryptograms enabling each party to authenticate the other. During this phase, the CMS also sets up the security level for further command exchange. Three possible security levels may be set up:

- AUTH: both parties are authenticated, but no cryptographic protection is attached to the APDUs sent through the secure channel.
- MAC: a Message Authentication Code computed using K_{MAC} is appended to the clear-text APDU to protect its integrity and ensure its origin.
- ENC+MAC: ENC+MAC: a Message Authentication Code (MAC) is computed using K_{MAC} over the data field of the clear-text APDU; the data field of the clear-text APDU is encrypted with K_{ENC} ; the MAC is appended to the encrypted data field of the APDU.

When a sensitive piece of data (like an encryption key) is sent to a SE through a SCP, the data field of the APDU must be encrypted with K_{DEK} as an additional security measure.

SCP02 option "i"= '55' ensures all the security requirements expected for the SCP. Mutual authentication is ensured though the exchange of two cryptograms enabling each party to authenticate the other. APDU corruption is prevented by the MAC protection attached to each command. Replaying APDU commands

taken from former sessions is prevented by the use of session keys. Replaying APDU commands from the same session is prevented by including an Initialization Chaining Vector (ICV) as part of the MAC computation. In option "i"= '55', the ICV is the MAC attached to the previous command encrypted with K_{ENC} . APDU command permutation or deletion is also prevented by the fact that the attached MAC depends on the previously sent commands, and therefore chains all the commands of the session. Finally, application code confidentiality is ensured by the encryption with K_{ENC} . If the MAC security level is being used to populate cryptographic keys after application loading, key confidentiality is preserved by encryption with K_{DEK} .

When sensitive pieces of data are encrypted with K_{DEK} , encrypting the data field of each personalisation command will not improve the "security" of the process, from a fraud-prevention point of view: it is not possible to recover the value of the sensitive pieces of data. The other pieces of data sent to the SE are not sensitive because they are accessible to anyone with a physical access to the card. However, encrypting the data field with K_{ENC} protects the privacy of the user: it makes it impossible to record the Calypso Serial Number or the initial value of the files attributed to a certain user. Using the ENC+MAC mode instead of the MAC mode does not incur a performance penalty because the encryption and authentication process is carried out in native code by the OS of the SE.

5.3.2. Mobile Phones

The case of a SE embedded in a mobile phone is interesting because a mobile phone is permanently connected to a telecommunication network that can be leveraged to deploy the Calypso Revision 3 application. However, it presents specific challenges to the implementation of this downloading process.

Note: This section only deals with the case of SIM SEs. It does not deal with the case of a non-SIM SEs embedded or hardwired in mobile phones. In this case, the mobile phone manufacturer is likely to provide a proprietary mechanism to send APDUs to this SE over an OTA channel. Such a SE should be handled like a regular SE: communications between an Application Provider and its APSD present in the SE shall be secured with SCP02 option "i"= '55'.

At this time, there is no specification that would define a standard mechanism to communicate OTA with non-SIM SEs embedded in mobile phones. Were embedded SEs to be deployed in large numbers, a standard would likely emerge, potentially from the ETSI or the NFC Forum. In this case, this document would have to be updated.

5.3.2.1. A Tiny Bit of History

For historical reasons, it is not possible to simply send an APDU command OTA to a mobile phone with instructions to deliver it to a GP application running in a SIM or (U)SIM card. Traditionally, applications running in SIM or (U)SIM cards use the SIM Toolkit API to communicate with the outside world. Since the SIM Toolkit API is only available in a SIM or (U)SIM card, applications designed to accommodate GlobalPlatform cannot use this API.

Furthermore, SIM cards did not support the GlobalPlatform specification since they had not use for it. The OTA communication mechanisms enabled the MNO to remotely adjust the SIM functions (the GSM keys and other kinds of telecom data) but loading new applications remotely was unheard of.

Today, SIM cards are being replaced by (U)SIM cards (also called UICC cards), which are fully multi-applicative Java Card environments with OTA capability for Remote Application Management (RAM), and applicative OTA dialog. However this does not automatically translate into support for the GlobalPlatform specification: while applications running in (U)SIM cards are similar to traditional Java Card applications, they are loaded and managed according to the specifications published by ETSI, which are not compatible with the GlobalPlatform specification.

Furthermore, (U)SIM applications usually rely on the SIM Toolkit API to communicate with the user and with remote servers. This API is not available to applications running in standard GlobalPlatform SEs so applications designed to run in such a SE (such as the Calypso Revision 3 application) do not use the SIM Toolkit API.

For several years, the ETSI and the GlobalPlatform Consortium have been working on bringing the whole GlobalPlatform specification to (U)SIM cards in order to enable GP-CMS to manage standard GP application in (U)SIM cards. Today, (U)SIM cards with full support for the GlobalPlatform specification are available from many vendors. In this case, every Java Card application can be loaded in one of those (U)SIM cards through the GlobalPlatform mechanisms.

Within such a (U)SIM card, a special SD with the OTA capability (called the OTASD) acts as a bridge between the "ETSI world" and the "GP world." When the OTASD receives an OTA command from the MNO, it is capable to translating the OTA command into a format adequate to an application in the "GP world". Similarly, when an application in the "GP world" returns a completion status, the OTASD can translate this status in a format suitable for OTA delivery. This bridge is mostly used to exchange standard GlobalPlatform management commands. This mechanism is being standardized by the Mobile Task Force within the GlobalPlatform Consortium.

Those (U)SIM cards still support the SIM Toolkit API since the SIM functions still rely on it. A (U)SIM card is therefore a mixed environment, as exposed in Figure 2: (U)SIM Card Architecture.



Figure 2: (U)SIM Card Architecture

5.3.2.2. OTA Communication Protocols

The secure channel protocol co-defined by ETSI and GlobalPlatform to exchange card and application management commands OTA between an off-card entity and the SD with OTA capability is called



SCP80. It is similar in function to SCP02 but includes additional capabilities to deal with the constraints of OTA communication. It is only supported in (U)SIM cards.

SCP80 was designed for traditional telecommunication scenarios: it is mostly used to communicate with the Security Domain that belongs to the MNO (i.e. the ISD). However it can also be used to communicate with any APSD if and only if the APSD is granted the Delegated Management privilege (or the Authorized Management privilege) and the OTA capability.

SCP80 was designed by ETSI as [102225] and [102226]. It is explicitly supported in [GP22].

Two solutions have been proposed to deliver management APDU to an APSD:

- **SCP80 only:** Grant the Delegated Management privilege (or the Authorized Management privilege) and the OTA capability to every APSD. Each APSD would have two key sets, one for SCP80, one for SCP02. In this case, the Application Provider would have to support a specific OTA CMS to manage the (U)SIM cards.
- **SCP02 over SCP80:** Encapsulate SCP02 commands within the SCP80 commands, illustrated in Figure 3: SCP02 Encapsulation for SIM Cards. The Application Provider would be able to communicate with its APSD with SCP02 but the actual message delivery between the MNO and the (U)SIM card would be carried out by the SCP80 protocol. The SCP02 are protected with the keys of the APSD so the commands exchanged over this channel are confidential from the point of view of the Issuer (see Section 5.4.1.2, Post-issuance Creation by the Issuer) This mechanism is under standardization at GP and at ETSI.

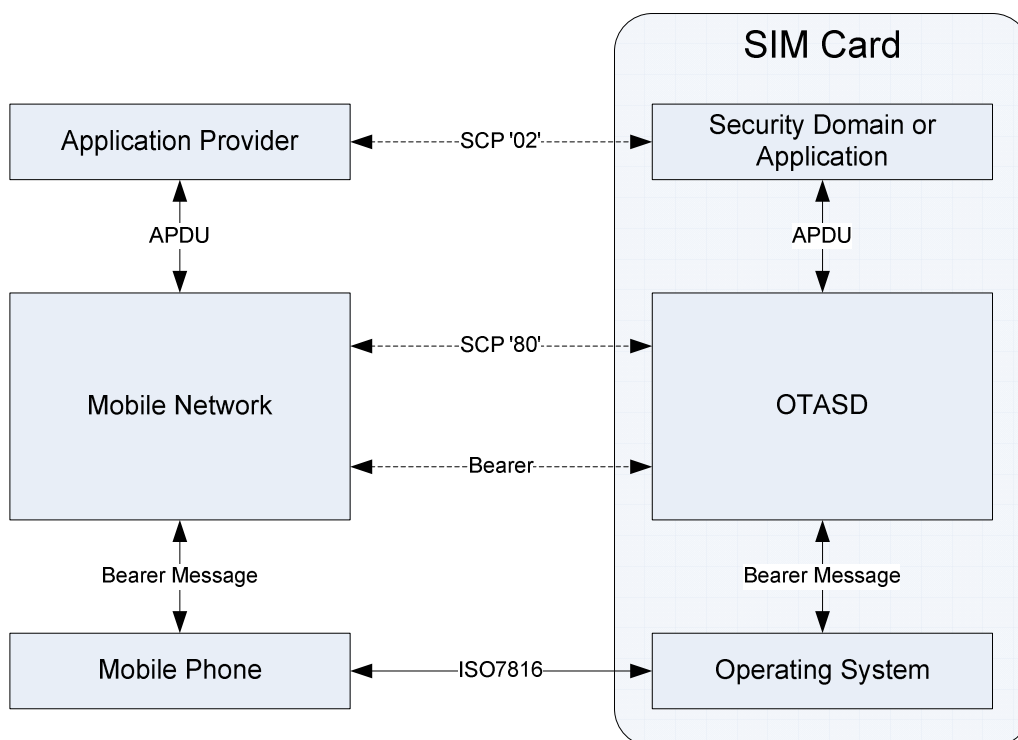


Figure 3: SCP02 Encapsulation for SIM Cards

Those two solutions are almost equally capable:

- Both protect the authenticity and confidentiality of the management commands using standard and public algorithms and protocols,
- Both support the installation and personalisation mechanisms presented in this document,
- Both are already supported by several leading (U)SIM card manufacturers.



Still, neither of those two solutions enables the delivery of an arbitrary APDU to a standard Java Card application (i.e. an application that does not use the SIM Toolkit API). APDUs can only be delivered to an APSD. In the context of the Calypso Revision 3 application, this means that it will not be possible to open a Calypso Secure Session remotely.

However, the SCP80 solution has a number of drawbacks compared to the “SCP02 over SCP80” solution.

First, SCP80 is not supported outside of (U)SIM cards, so using the “SCP80 only” method would force Application Providers to support two different CMS: a SCP02 CMS and a SCP80 CMS. With the “SCP02 over SCP80” solution, an Application Provider can focus on supporting a single CMS based on SCP02 and delegate the encapsulation over SCP80 to the manager of the OTASD. This role may be played by the MNO or by another actor like a TTP if the (U)SIM card supports multiple APSDs with OTA capability.

Second, at this time there is uncertainty as to whether SCP80 supports K_{DEK} : [102226] is unclear on this subject. K_{DEK} is used to protect the confidentiality of the Calypso keys when they are delivered during the personalisation phase (see Section 3.6.8, 3.6.9, and 5.3.1): this feature is mandatory.

Some mobile phones support a simple but ultimately unsatisfying workaround to this communication issue called a "Proxy Midlet." If a phone supports the JSR-177 API or the JSR-257 API, an authorized Java Midlet can exchange APDUs with an application or a Security Domain running in the (U)SIM card, through the ISO7816 interface. Furthermore, a Midlet can communicate over HTTP with a remote network if the mobile phone supports this kind of connections. Therefore a Midlet can be used to relay secured APDUs back and forth between a remote Application Provider and a GlobalPlatform application.

This workaround presents several drawbacks:

- Not every phone supports MIDP.
- Not every MIDP phone supports the JSR-177 API or the JSR-257 API.
- Historically, MIDP has been a very unstable specification: the Midlet Proxy would have to be tested and potentially modified for every mobile phone's model so it is not possible to design a single, generic Midlet Proxy.
- The Midlet has to be signed, which usually requires a certification phase with the MNO and the manufacturer of the mobile phone.
- The management system would have to support Java Card applications and Midlets.
- There is no standard provisioning system for Midlets.

Furthermore, this solution introduces complex and non-intuitive interactions with the end-user, for an action which is done just once in the lifecycle of the application:

- Accepting a WAP PUSH SMS,
- Opening a WAP connection to load the Proxy Midlet,
- Installing and launching the Proxy Midlet.

On the other hand, using a standard OTA mechanism (“SCP80 only” or “SCP02 over SCP80”) provides a transparent and easy process for the end-user to load and to personalise the Calypso Revision 3 application since the loading, installation and personalisation phases are invisible to the end-user.



For all those reasons, this document strongly advises against using a Proxy Midlet for the purpose of loading, installing and personalizing the Calypso Revision 3 application.

5.4. GlobalPlatform Management

The management framework defined in [GP22] addresses all the smartcard market segments and therefore includes many optional features in order to deal with a variety of scenarios. Using best practices and existing recommendations from potential stakeholders like MNOs and Application Owners, this document defines:

- Four scenarios for the management of the keys of the Security Domain of an Application Provider,
- Four scenarios for the management of the Calypso application in a Security Domain.

Those scenarios are realistic examples of potential applications of the GlobalPlatform technology to the Calypso ecosystem but do not preclude the existence of alternative scenarios. This document does not mandate specific scenarios.

5.4.1. Security Domain Management

The management commands sent by an Application Provider to its APSD are secured using a set of 3 keys: the SD keys. Those keys protect the origin, integrity and confidentiality of the management commands exchanged between an Application Provider and its APSD.

Those keys are usually diversified by SE which means that every SD of every SE uses a different set of keys. A sequence of APDUs computed with the keys of the "Calypso" APSD of SE A cannot be replayed on the "Calypso" APSD of SE B. Similarly, the keys of every APSD are diversified by SE, which means that every APSD of the every SE uses a different set of keys. A sequence of APDUs computed with the keys of APSD X of SE A cannot be replayed on the APSD X of SE B. Similarly, a sequence of APDUs computed with the keys of APSD X of SE A cannot be replayed on the APSD Y of SE A. Please note that those propositions are true only if the sequence of APDU includes the initialisation of the Secure Channel.

However, each APSD has to be created by another SD, and the keys of the newly created APSD must be forwarded to the Application Provider in a secure fashion. This phase is necessary prior to the downloading process. This chapter described the different options available to create an APSD in a SE and to transfer those keys.

5.4.1.1. Pre-issuance Creation

When a SE is designed to support multiple Application Providers, a set of default APSDs is usually instantiated during the manufacturing phase. Each default APSD is initialized with a different set of keys. Along with the SEs, the SE manufacturer transfers the keys for the ISD to the Issuer while the keys for each APSD are transferred to a trusted third-party (TTP). Therefore the Issuer does not know the keys for the APSDs available in its SEs. At the request of the Issuer, the TTP securely forwards the key set for an APSD to an Application Provider. The actual delivery means are outside of the scope of this document: physical delivery, secure communication links... In this case, the TTP simply acts as safe repository of keys and is not actually involved in the actual management of the SEs.

This solution has one main drawback: evaluating the right number of pre-instantiated APSDs is difficult. If the manufacturer creates too many APSDs, it wastes precious memory in its SE. The main result is that manufacturers usually create a limited number of APSDs, if any. If the number of pre-created

APSDs is exhausted during the life of the SE, it is necessary to create new APSDs post-issuance. This presents a set of challenges that will be tackled by the remaining scenarios.

Note: When the SE is going to support a single Application Provider which is also the Issuer, the SE may be created with a single SD, namely the ISD. In this case, the SE manufacturer simply transfers the keys for the ISD to the Issuer. In turn, the Issuer updates the keys within its own, secure facilities, prior to customer delivery.

5.4.1.2. Post-issuance Creation by the Issuer

If the manufacturer does not create enough APSDs, the Issuer has to create new APSDs by itself. This scenario may replace or complement the creation of APSDs during the manufacturing phase.

Creating a new APSD post-issuance is a three-step process:

1. The Issuer creates the APSD and loads an initial set of keys.
2. The Issuer securely forwards this initial set of keys to the targeted Application Provider.
3. The Application Provider updates the APSD with a new set of keys.

According to this process, the Issuer cannot know the value of the final set of keys loaded in the APSD: the APSD creation is confidential from the point of view of the Issuer.

However this property does not hold when the role of the Issuer is played by an entity which is also in charge of delivering the APDU commands that update the key set of the newly created APSD. When the Application Provider sends the PUT KEY command to replace the default keys of the newly-created APSD, the content of this PUT KEY command is encrypted with the default key set provided by the Issuer to the Application Provider. If this command is carried over a network controlled by the Issuer, the Issuer is theoretically able to decrypt the content of this PUT KEY command and to recover the value of the new key set.

This is the case for (U)SIM cards: the MNO is both the Issuer and in charge of the network infrastructure that delivers the APDU to the targeted APSD. This scenario requires the MNO to actively listen and record such commands, but it is possible according to [GP22].

In this case, a simple solution is to let the Application Provider update the key set of its APSD locally, in an environment controlled by the Application Provider. For example, the Application Provider may update the key set when the user subscribes to the service at the point of sales. This solution is not always acceptable since it requires the user to visit a real point of sales managed by the Application Provider and to take the (U)SIM out of her mobile phone. The next sections describe additional solutions to this confidentiality issue.

5.4.1.3. Post-issuance Creation by a Trusted Third-Party

This scenario presents a solution to the confidentiality issue raised by the previous scenario. This scenario expands upon the "Pre-issuance Creation" scenario by allowing one of the pre-created SD to create additional SDs post-issuance. This special SD is managed by a specific TTP and is called the SD_{TTP}. The creation of this special SD_{TTP} during the manufacturing phase does not preclude the creation of standard APSDs during this phase.

The TTP obtains the keys for the SD_{TTP} from the SE manufacturer. With those keys, the TTP can create APSDs post-issuance. In turn, the TTP must forward the keys for the new APSD to the Application Provider.

Since the Issuer does not know the keys for the SD_{TTP} , it cannot decrypt the APDU commands sent to change the keys of the SD_{TTP} or the keys of the APSDs created by the SD_{TTP} . Therefore the Issuer cannot access the value of the new key sets even if it can record every single exchange between the TTP and the SD_{TTP} . For (U)SIM cards this means that the MNO cannot recover the values of the keys used by the APSDs.

Initially, the keys of the APSD are known to the TTP since the TTP is responsible for the creation of the APSD. However, the Application Provider can update the APSD's keys with new keys that are not known to the TTP. If this process is carried out over a communication channel which is not available to the TTP, the TTP cannot recover the new keys. In this case, the Application Provider is assured that the keys to the APSD cannot be recovered by the TTP or by the Issuer. For (U)SIM cards, the TTP cannot listen to the commands exchanged between the Application Provider and its APSD so the keys of the APSD are confidential from the MNO and from the TTP.

5.4.1.4. Post-issuance Creation with CCCM

This scenario presents another solution to the confidentiality issue raised in the "Post-issuance Creation by a Trusted Third-Party" scenario. The solution proposed in [CCCM] is to use public key cryptography in order to cipher the keys delivery, and so to perform keys management keeping such keys confidential from the Issuer.

[CCCM] proposes two keys creation modes:

- On board key generation: the keys are generated in the SE and are sent back to the server in a confidential way,
- Off board key generation: the keys are generated on the server side and are sent back to the SE in a confidential way.

Confidentiality is ensured by using either certificates (public and private keys) or shared keys for both the Controlling Authority (CA) and the Application Provider.

For instance, if certificates are used, the public and private keys of the CA are loaded in the SE, in a dedicated SD_{CA} , at manufacturing time. Then the public key of the Application Provider is loaded in its own APSD.

In case of on board key generation, the generated keys are ciphered using this Application Provider public key (to ensure confidentiality: only the private key of the Application Provider can un-cipher them), and the message is signed using the private key of the CA (to ensure authentication of the message: the sender is a certified CA).

In case of off board key generation, the generated keys are ciphered using the CA public key (to ensure confidentiality: only the private key of the CA can un-cipher them, done by the SD_{CA}), and the message is signed using the private key of the Application Provider (to ensure authentication of the message: the sender is the Application Provider itself: verified by the APSD using the loaded public key)

5.4.1.5. Summary

The scenarios are summarized in the following table:

Scenario	1	2	3	4
Main Feature	Pre-issuance creation	Post-issuance creation by the ISD	Post-issuance creation by a TTP	Post-issuance creation secured by CCCM
Availability	Now	Now	Now	Planned
Use case	Simplicity	Issuer is not the Application Provider	Issuer, TTP and Application Provider are distinct.	Issuer is not the Application Provider
Required GP Optional features				CCCM

5.4.2. Application Management

The previous section defined the mechanism to create an APSD in a GlobalPlatform SE. This section describes the GlobalPlatform mechanisms required to load and install an application in a GlobalPlatform SE through an APSD. Just like the previous section, this section describes four scenarios in order to expose several potential deployments of the GlobalPlatform technologies and to deal with multiple business cases.

Each Application Provider may represent either one or several Application Owners. An Application Owner may be represented by several Application Providers. The role of the Issuer may be played by an Application Owner or by another entity.

Each scenario is concluded by a simple, realistic example. This example is for illustrative purpose only: it is not claimed that a scenario is only acceptable in the context of the example.

5.4.2.1. Scenario 1

This scenario represents a straight transcription of the scheme used for existing Calypso deployments to native SEs: The Application Owner is the Issuer.

For this scenario the SE holds a single Security Domain, which is the Issuer Security Domain. Every package loaded in the SE is assigned to the Issuer Security Domain. The Issuer is the only entity that is allowed to load a package in the SE.

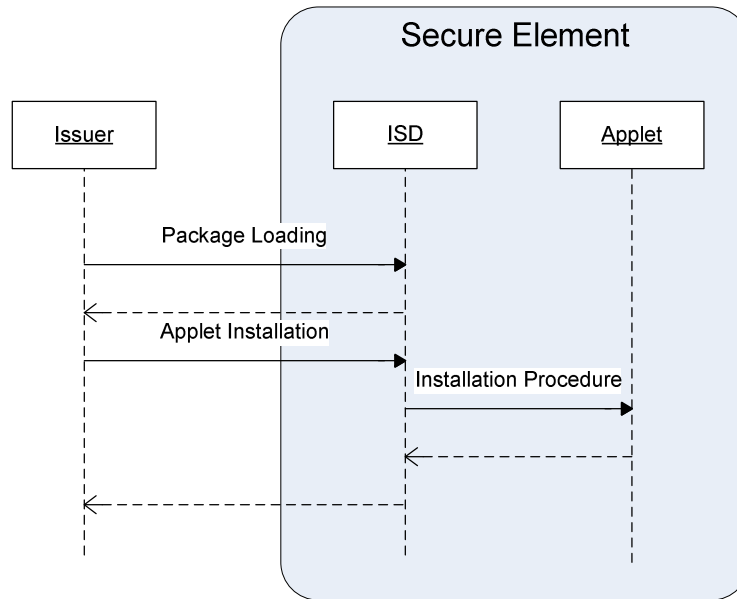


Figure 4: Scenario 1 - Issuer Management

This scenario matches the current state of most of the Calypso uses cases that are presently deployed, where the Application Provider is the Issuer as well. It may be acceptable for USB keys dedicated to a single Application Provider.

Such a scenario supports a SE with multiple applications. This scenario is perfectly adequate to support a USB key with several Calypso applications (one per transport authority) and several Calypso Stored Value applications.

This scenario is technically possible for (U)SIM cards but is unlikely to be used: MNOs usually assume the role of the Issuer and normally will not load an additional package in their Security Domains.

Example:

A certain city runs its own transportation network. Like most cities, it provides its inhabitants with free libraries. Instead of deploying a different mono-application card for every city service, the city deploys a single, multi-applicative card that hosts the Calypso Revision 3 and a custom application to authenticate its inhabitants when they borrow books. In this case, the city plays the role of the Issuer and the role of the Application Provider.

5.4.2.2. Scenario 2 – DAP Verification

This scenario represents an update of Scenario 1 where the Application Owner is not the Issuer but still retains full control of the application. This scenario can be used when the Application Owner cannot or does not want to play the role of the Issuer.

In this scenario, the SE hosts two types of Security Domains: the ISD managed by the Issuer, and one or more APSDs, each one managed by an Application Provider.

In this scenario, the APSDs cannot load packages nor install applications by themselves. The following process is required to install an application in one of the APSDs:

- The package of the application is loaded by the Issuer under the ISD,
- The application is installed by the Issuer under the ISD,
- The application is extradited to the targeted APSD.

In order to control the content of the application received from the ISD, the DAP Verification privilege is granted to each APSD. During the execution of the LOAD command, the APSD associated with the package being loaded may verify that its Application Provider has approved the loading operation through the verification of the DAP signature attached to the package.

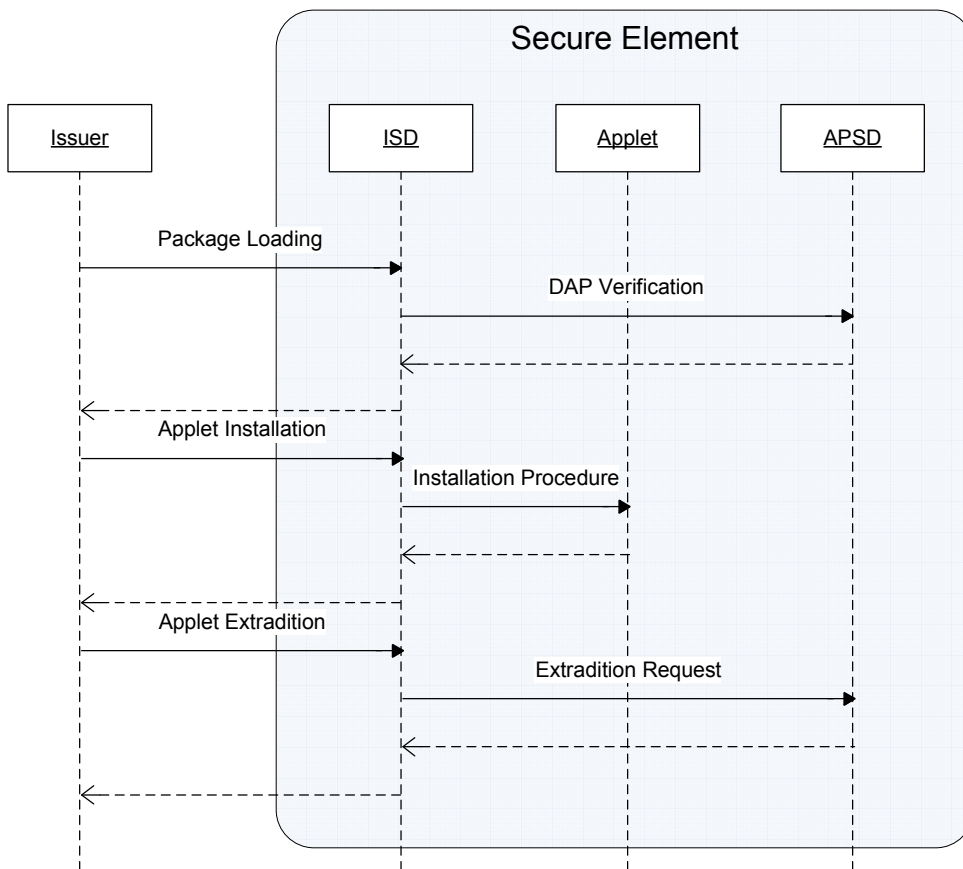


Figure 5: Scenario 2 - Issuer Management and Multiple Application Providers

In this scenario, the package holding the Calypso Revision 3 is not confidential from the point of view of the Issuer.

This scenario is adequate for every form-factor, including (U)SIM cards.



Example:

A bank deploys a contactless, multi-applicative GlobalPlatform card. The bank strikes a deal with a transport operator to host and manage a Calypso application in a separate security domain. The bank issues the card with a single EMV application. Customers can “activate” the transport application through one of the ticketing machines run by the transport operator. In this case, the bank plays the role of the Issuer and the transport operator plays the role of an Application Provider.

5.4.2.3. Scenario 3 – Delegated Management

This scenario increases the amount of control granted to the Application Owner. In this case, the Application Owner would not play the role of the Issuer, but it would share some of its privileges. More precisely, the Application Owner would be able to manage applications on its own, provided that the management commands have been validated by the Issuer.

In this scenario, the SE hosts two types of the Security Domain: the Issuer Security Domain (ISD) managed by the Issuer, and one or more Application Provider Security Domains (APSD), each one managed by a different Application Provider.

In this scenario, the APSDs are granted the Delegated Management privilege so they can perform any kind of SE management operations on their own. However, the ISD is granted the Token Verification privilege, so during every management operation, the ISD is able to verify that every command (including the loading commands) has been signed by the Issuer.

The APSD is not granted the DAP Verification privilege and shall refuse extradition request from the ISD.

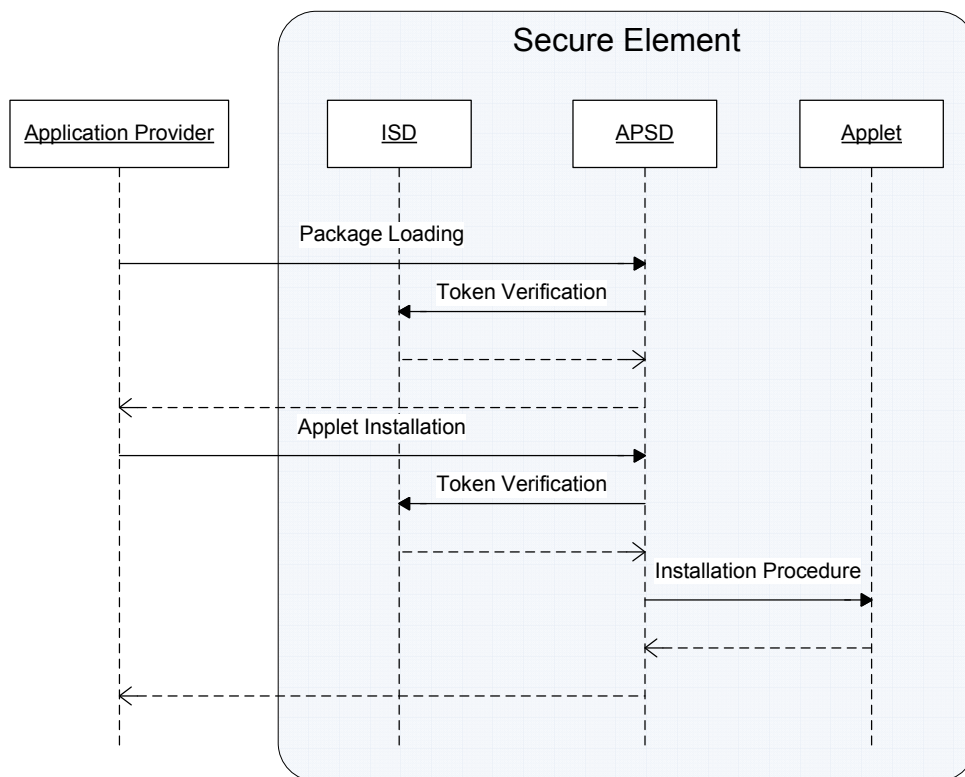


Figure 6: Scenario 3 - Delegated Management

This scenario is adequate for every form-factor.



This scenario uses a feature (Delegated Management) that has not been largely deployed and therefore is not available on every GlobalPlatform SE. However, the availability of this feature is set to rise in the near future in order to match the requirements of emerging applications, like NFC services. This is especially true for (U)SIM cards.

This scenario grants the Issuer a fine level of control over the application management process while reducing its own involvement:

- The involvement of the Issuer in this process is reduced because it does not have to manage the connectivity to the SE: the Issuer only has to provide a Token when the Application Provider requests one. The Application Provider has to provide the mechanism that delivers the commands to the SE. For example, applications running in (U)SIM cards are usually delivered OTA, but according to this scenario an Application Provider may use the ISO14443 interface of the SE instead. This would enable a transport authority to deliver new applications and updates through its existing network of vending machines instead of going through the mobile network of the MNO.
- The level of control granted to the Issuer is finer because it can use the Token mechanism to control the commands sent to the SE by the Application Provider. Tokens management is based on issuer policy and can be deployed with unique or reusable Tokens.

Reusable Tokens are valid for every SE issued by the Issuer. This means that the Application Provider only has to obtain the Token once per command: the same Token can be reused over and over for every SE.

Unique Tokens are tied to a single SE. This means that in order to load an application in a certain SE, the Application Provider has to request a Token computed specifically for this SE. Another SE would not be able to validate the returned Token. This can be accomplished by diversifying the key used to validate each Token in the SE.

Unique Tokens enable an Issuer to track the commands sent to a specific SE. During the creation of the Token, if the Issuer determines that according to the current state of the SE, the submitted command should not be executed, the Issuer may deny the Application Provider's request for a Token. For example, [GP22] does not define a way to allocate a fixed amount of in-card memory to a specific APSD. This means that an uncooperative Application Provider may use enough memory to prevent other Application Providers from loading their applications in the SE. In this case the Issuer can prevent this problem from occurring by limiting the numbers of applications installed by each Application Provider by denying requests for new Tokens.

This specific quota-management issue is presently being discussed within the GlobalPlatform Consortium and will possibly be fixed with the release of an update to [GP22].

In this scenario, the package holding the Calypso Revision 3 is confidential from the point of view of the Issuer if the keys of the APSD that receives the package were loaded using a mechanism confidential from the point of view of the Issuer.

The Issuer may track each SE individually so it can deal with SEs of varying functionalities and capacities.

Example:

A MNO diversifies its offering and produces an USB key that embeds an NFC interface and a SE compatible with [GP22]. Once plugged in a PC, the user can make VOIP calls over the Internet thanks to the (U)SIM application embedded in the SE.

A transport operator enters into an agreement with the MNO to rent an APSD in the SE. At this point, the user can use the website of the transport operator to request the installation of a Calypso application. In this case, the transport operator has a direct, unmediated connection to the SE embedded in the USB key, but the SE will verify the token of each command in order to make sure that the MNO (i.e. the Issuer) accepts the modifications.

In this case the MNO plays the role of the Issuer, and the transport operator plays the role of an Application Provider.

5.4.2.4. Scenario 4 – Authorized Management

This scenario gives a total control to the Application Owner over its own applications. In this case, the Application Owner would not play the role of the Issuer, but it would have full privileges over its own applications. More precisely, the Application Owner would be able to manage applications on its own, without any validation of the Issuer.

Authorized Management is a new feature of [GP22]. It enables the Issuer to give a SD full control over the card content loaded under this SD. If a SD is granted the Authorized Management privilege, the ISD will not control the commands exchanged between the SD and its owner.

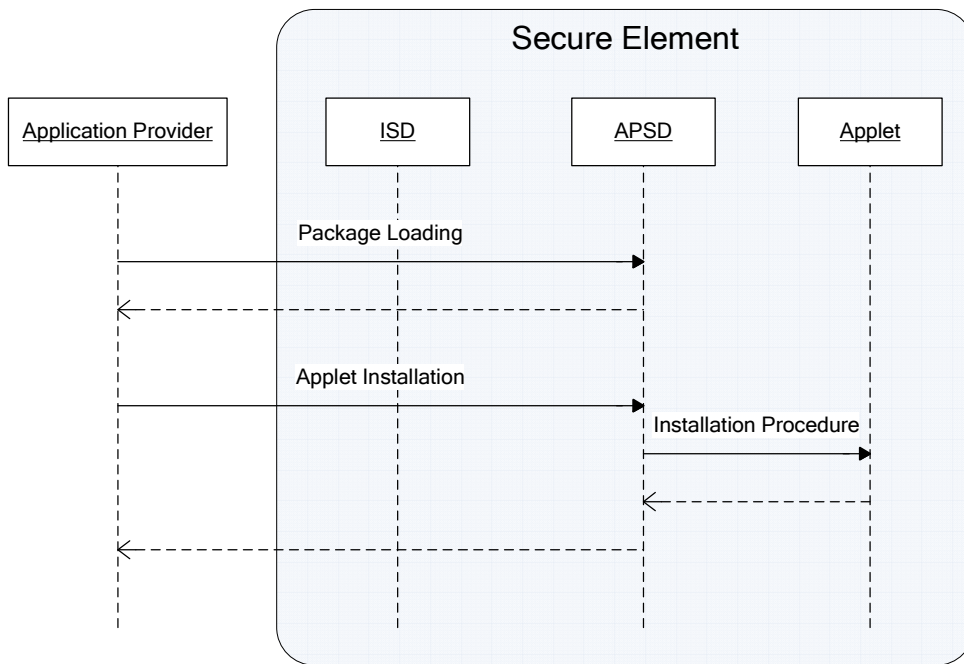


Figure 7: Scenario 4 - Authorized Management

Technically, the SE is virtually split in at least two areas under responsibility of the same number of entities:

- The area of the ISD: under the responsibility of the Issuer, it includes the ISD, all the Security Domains linked with this ISD, and all the classical applications of the Issuer linked with these SDs. The Issuer has the full control on all these applications.
- The area of the SD with the Authorized Management privilege: under the responsibility of the owner, it includes all the Security Domains and applications linked with this SD. The owner of this SD has the full control on all this area, making any application management commands without the need of any interaction with the Issuer.

This scenario enables the Issuer to eliminate its own involvement while maintaining a certain amount of control over the application management process:

- The Issuer does not have to manage a DAP Verification process or the creation of Tokens for a Delegated Management process.
- The APSD with the Authorized Management privilege cannot perform any action outside of its area (and in particular in the Issuer area).
- The Issuer may retain the right to delete certain items in the area of the APSD with the Authorized Management privilege: this enables the Issuer to maintain control over its SEs in case of an emergency. For example, the Issuer may lock the APSD and every application managed by this SD.

The owner of the APSD with Authorized Management privilege may be either a regular Application Provider (like CNA) or a TTP. In this last case, the TTP would create APSDs on behalf of Application Provider.

In this scenario, the package holding the Calypso Revision 3 is confidential from the point of view of the Issuer if the keys of the APSD that receives the package were loaded using a mechanism confidential from the point of view of the Issuer.

This scenario uses a feature (Authorized Management) that has not been largely deployed and therefore is not available on every GlobalPlatform SE. However, the availability of the feature is set to rise in the near future in order to match the requirements of emerging applications, like NFC services. This is especially true for (U)SIM cards.

Example:

A MNO deploys a (U)SIM card and rents a SD with the Delegated Management privilege to a transport operator. This transport operator loads and installs a Calypso application in the card.

The setup is running smoothly for several years and the number of users is growing regularly. The MNO decides that the value of the resources spent on validating the commands of the transport operator outweigh the risk associated with this specific transport operator. The MNO and the transport operator negotiates a new agreement and from this moment on newly created SDs for this transport operator are granted the Authorised Management privilege: the commands sent by the transport operator to load, install and personalize the Calypso application are not validated by the MNO anymore. If a serious breach of contract occurs, the MNO retains the possibility of deleting the SD of the transport operator and every associated application.

5.4.2.5. Summary

The scenarios are summarized in the following table:



Scenario	1	2	3	4
Main Feature		DAP Verification	Delegated Management	Authorized Management
Availability	Now	Now	Planned	Planned
Suitable for Use Case	Card, USB Simple	Card, USB, SIM Available	Card, USB, SIM Limited involvement of the Issuer	Card, USB, SIM No involvement of the Issuer
Required GP Optional Features		Application Provider Security Domains, DAP Verification, Application Extradition	Application Provider Security Domains, Delegated Management	Application Provider Security Domains, Authorized Management
Package Confidential from the Issuer	N/A	No	Yes	Yes

The scenarios 2, 3 and 4 rely on the pre-existence of an APSD. Therefore those 3 scenarios must be used in conjunction with one of the "Security Domain Creation" scenario. The scenario 1 relies on the ISD so it does not require the creation of an additional Security Domain.

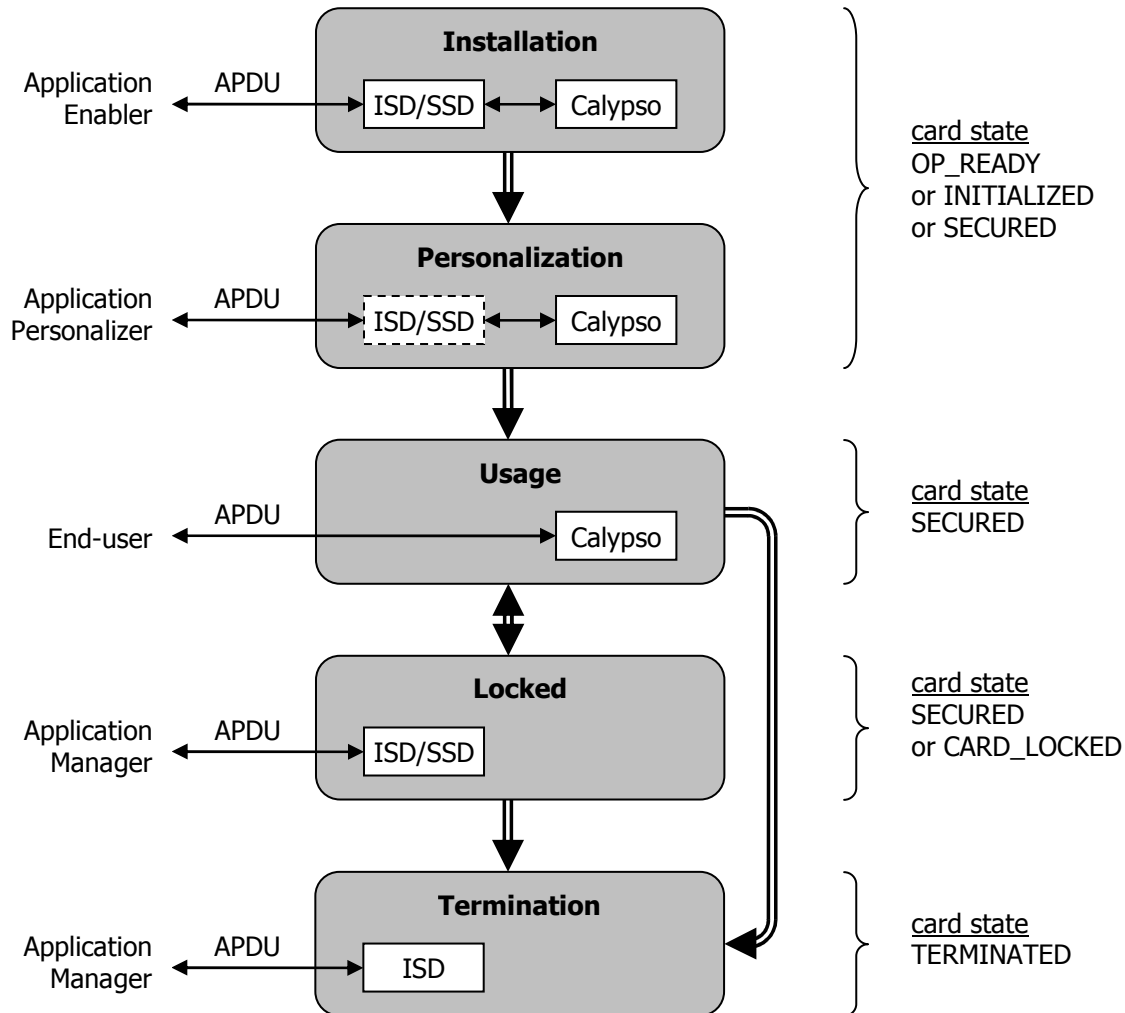
5.5. GlobalPlatform Life Cycle

At any time, the ISD may be in the OP_READY, INITIALIZED, SECURED, CARD_LOCKED or TERMINATED states. Transitions between those states may be requested by the application or initiated by the ISD.

The life cycle of a GP application consists of five phases: Installation, Personalisation, Usage, Locked and Termination. Transitions between those states may be requested by the application or forced by the ISD. In the figure below, grey boxes represent phases and double-lined arrows between the grey boxes represent allowed phase transitions. Some transitions are irreversible.

The GlobalPlatform life cycle of every application is independent from the life cycle of the ISD and from the life cycle of the Security Domain. It is also independent of the GlobalPlatform life cycle of every other application, even applications instantiated from the same package. However, the state of the ISD takes precedence over the state of every application. For example, if the state of the ISD is CARD_LOCKED, it is impossible to send commands to the application.

The figure below describes the lifecycle applied to a standard GlobalPlatform application:



5.6. Application Personalisation

5.6.1. Nominal Case

In most cases an application has to be personalized after its installation phases, to initialize some values or to load a set of keys. This document presents a standard approach to personalisation based on [GP22]:

- Selecting the targeted Security Domain (ISD or APSD),
- Opening a secured logical channel,
- Sending an INSTALL [for personalization] to tell the targeted SD that the following STORE DATA commands should be forwarded to a specific application managed by the SD,
- Sending the personalisation data in one or more STORE DATA commands,
- Transitioning the application's GlobalPlatform state to APPLICATION_SELECTABLE.

When the APSD receives the STORE DATA commands, it is decrypted with the APSD keys. If the content of the commands is successfully decrypted, it is forwarded to the targeted application as a buffer. In turn, the application decodes this buffer and updates its internal state accordingly.

The APSD does not interpret the content of each STORE DATA command so this content may be encrypted if it is extremely sensitive, typically with K_{dek} .

This process is not impacted by the management scenario selected from Section 5.4, GlobalPlatform Management: every scenario supports this personalisation process.

[GP22] does not mandate a format for the content of the STORE DATA commands: this document proposes to follow the format defined in [CPS]. According to this document, personalisation data should be packaged as a set of Data Grouping Identifiers (DGIs). [CPS] is proposed by EMV and has been largely adopted for the personalisation of banking cards.

[CPS] applies only to the communications between the application and the off-card entity in charge of preparing the personalisation APDUs. It does not cover the mechanisms required to collect the personalisation data.

With [CPS], application data is encoded into discrete DGIs. Since DGIs are processed by the application, they may be encoded using a variety of schemes. However it is desirable to reduce the amount of computation required from the application so DGIs are usually encoded using a format as close as possible to the final storage format in the application. In the best case, the byte array contained in a DGI is simply copied over to an internal buffer.

This proposal has many advantages over proprietary personalisation schemes:

- It reuses existing commands,
- Commands are protected with standard and validated cryptographic protocols,
- Commands are protected with keys that are already present in the SE,
- DGIs can be used to pack multiple records into one APDU,
- Personalisation tools and machines are already compatible.

In order to use the existing personalisation SAMs, the Calypso application must be able to return 20 bytes at the beginning of the Activation process. [CCCM] updates the `processData` method defined in [GP22] to enable this method to return such a buffer of data. [CCCM] is proposed by the GlobalPlatform Consortium and is based on [GP22].

5.6.2. Support for older GlobalPlatform Specifications

Presently, most SEs support either GP 2.1.1 or GP 2.2 but without support for [CCCM]. GP 2.2 is a superset of GP 2.1.1 and therefore most of the features used for the management scenarios are already present in GP 2.1.1. The personalisation mechanism defined in the previous section can be reused to personalize GP2.1.1 SEs or GP 2.2 SEs without support for [CCCM] with only minor modifications.

Since those SEs do not support the extended `processData` method defined in [CCCM], the main issue is that they cannot return the Calypso Challenge to initiate the personalisation process. The solution is to open a SCP channel with the Calypso application itself instead of opening it with the SD. The downloading process is turned into a multi-step process :

- Opening a SCP with the APSD to load and install the Calypso application,
- Closing the SCP,
- Selecting the Calypso application,
- Opening a SCP with the Calypso application with the keys of the APSD,
- Sending the STORE DATA commands to the Calypso application,
- Closing the SCP.

Commands sent in the SCP to the Calypso application are decoded by the APSD on behalf of the Calypso application. The security properties of the process are the same as the security properties of the

method used for SEs that comply with [GP22]. The Calypso application has to support this process explicitly.

However, this process is not acceptable for (U)SIM cards since there is no standard mechanism to deliver GP commands OTA to a GP application running in a (U)SIM card: the STORE DATA commands must be channelled through the APSD.

5.7. Error Handling

This section discusses the behaviour of a Calypso application when an error occurs during the downloading process.

Since this process is performed OTA, failures and errors are bound to occur. The overall requirement is that an error at any point during the downloading process shall not let the SE, a SD or a Calypso application in an incoherent external state.

The guidelines provided in this section consider the case of a single instantiation of a Calypso application.

During the downloading process, most errors belong to one of those two types:

- The SE or the Calypso application returns a Status Word indicating an error. This kind of problem should be extremely rare since it indicates an error in the code of SE, in the code of the Calypso application or in the command sent to the SE. It is not recoverable: any attempt to send the same data again will lead to a similar result. This kind of error must be flagged for manual processing.
- The connection to the SE is lost. The actual cause of the loss is usually unknown: a congestion problem, a weak battery, a software error in a component of a network... This kind of error may be transient and recoverable in some cases.

Errors of the first type must be handled on a case-by-case basis since they may have a huge variety of causes. The remaining of this section covers only the second error type.

When the connectivity to the SE is restored, the CMS should attempt to continue the downloading process as if nothing had happen: there is always a possibility that the SE was not the cause of the error and that the SCP is still available. Therefore, the CMS may attempt to resend the last command. However it should be noted that since the SE is not actually affected by this error, it is not involved in the recovery process. Therefore the mechanisms required to implement this behaviour are outside of the scope of this document.

If the SE returns an error indicating that the SCP is not available anymore or if the connection to the SE has to be rebuild from the ground up, the CMS must remotely identify the actual state of the SE and restart the process at the point of failure.

The atomicity of the Loading and Installation phases is guaranteed by the GlobalPlatform implementation. If an error is detected during one of those phases, the CMS can obtain the status of every package and every application from the ISD or from an APSD. Using this information, it must decide upon a course of action to rectify the error:

- Restart the Loading phase if the package holding the application is not loaded,
- Restart the Installation phase if the application is not installed.

The atomicity of the Pre-personalisation phase is not guaranteed by the GlobalPlatform implementation. Furthermore, unlike most GlobalPlatform applications, the personalisation of a Calypso application involves the allocation of a large amount of in-memory structures since the file system is defined during

this step. Since the creation of those structures may span several APDUs, a connectivity error during the Activation phase may let the Calypso application with an incorrectly initialized internal structure.

An implementation of the Calypso Revision 3 specification must be designed to accommodate this possibility: if an error is detected during the Pre-personalisation phase, the Calypso application must permanently block itself and refuse to process any subsequent APDUs. This will also prevent attackers from exploiting the incoherent internal state of the Calypso application.

The Pre-personalisation phase can be restarted by deleting the Calypso application (using the DELETE command) then instantiating it again (using the INSTALL [for install] command). In most cases, the package does not have to be reloaded so this process does not incur a significant performance penalty.

The functional requirements for this process were presented in Section 3.7, Error Handling.