CORE Working Group
Internet Draft
Intended status: Experimental

P. Urien
Telecom Paris

October 2 2021

Expires: April 2022

Remote APDU Call Secure (RACS)
draft-urien-core-racs-15.txt

Abstract

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grid of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm2; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements. It is designed according to the representational State Transfer (REST) architecture. RACS resources are identified by dedicated URIs. An HTTP interface is also supported.

An open implementation [OPENRACS] is available (https://github.com/purien) for various OS.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 2022.
.

Copyright Notice

Table of Contents

1 Overview

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grids of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm2; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements.

RACS is designed according to the representational State Transfer (REST) architecture [REST], which encompasses the following features:
- Client-Server architecture.
- Stateless interaction.
- Cache operation on the client side.
- Uniform interface.
- Layered system.
- Code On Demand.

1.1 What is a Secure Element

A Secure Element (SE) is a tamper resistant microcontroller equipped with host interfaces such as [ISO7816], SPI (Serial Peripheral Interface) or I2C (Inter Integrated Circuit).

The typical area size of these electronic chips is about 25mm2. They comprise CPU (8, 16, 32 bits), ROM (a few hundred KB), nonvolatile memory (EEPROM, FLASH, a few hundred KB) and RAM (a few ten KB). Security is enforced by multiple hardware and logical countermeasures.

According to the [EUROSMART] association height billion of such secure devices were shipped in 2013. Secure elements are widely deployed for electronic payment (EMV cards), telecommunication (SIM modules), identity (electronic passports), ticketing, and access control.

Most of secure elements include a Java Virtual Machine and therefore are able to execute embedded program written in the JAVACARD language. Because these devices are dedicated to security purposes they support numerous cryptographic resources such as digest functions (MD5, SHA1, SHA2...), symmetric cipher (3xDES, AES) or asymmetric procedures (RSA, ECC).

A set of Global Platform [GP] standards control the lifecycle of embedded software, i.e. application downloading, activation and

deletion.

As an illustration a typical Secure Element has the following characteristics:

- JAVACARD operating system;
- Compliant with the GP (Global Platform) standards;
- 160 KB of ROM;
- 72 KB of EEPROM;
- 4KB of RAM;
- Embedded crypto-processor;
- 3xDES, AES, RSA, ECC;
- Certification according to Common Criteria (CC) EAL5+ level;
- Security Certificates from payment operators.

1.2 Grid Of Secure Elements (GoSE)

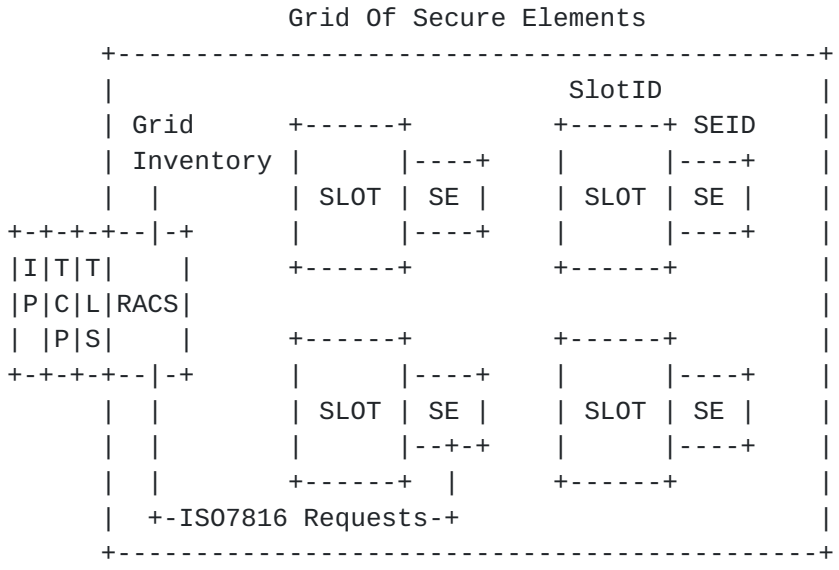

Figure 1. Architecture of a Grid of Secure Elements

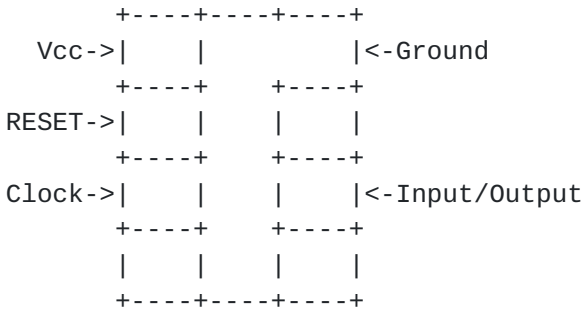

Figure 2. Illustration of an ISO7816 Secure Element

A grid of Secure Elements (GoSE) is a server hosting a set of secure elements.

The goal of these platforms is to deliver trusted services over the Internet. These services are available in two functional planes,
- The user plane, which provides trusted computing and secure storage.
- The management plane, which manages the lifecycle (downloading, activation, deletion) of applications hosted by the Secure Element.

A grid of Secure Elements offers services similar to HSM (Hardware Secure Module), but may be managed by a plurality of administrators, dealing with specific secure microcontrollers.

According to this draft all accesses to a GoSE require the TCP transport and are secured by the TLS [TLS 1.0] [TLS 1.1] [TLS 2.0] protocol.

The RACS protocol provides all the features needed for the remote use of secure elements, i.e.
- Inventory of secure elements
- Information exchange with the secure elements

1.3 Secure Element Identifier (SEID)

Every secure element needs a physical slot that provides electrical feeding and communication resources. This electrical interface is for example realized by a socket soldered on an electronic board, or a CAD (Card Acceptance Device, i.e. a reader) supporting host buses such as USB.

Within the GoSE each slot is identified by a SlotID (slot identifier) attribute, which may be a socket number or a CAD name.

The SEID (Secure Element IDentifier) is a unique identifier indicating that a given SE is hosted by a GoSE. It also implicitly refers the physical slot (SlotID) to which the SE is plugged.

The GoSE manages an internal table that establishes the relationship between SlotIDs and SEIDs.

Therefore three parameters are needed for remote communication with secure element, the IP address of the GoSE, the associated TCP port, and the SEID.

1.3.1 SlotID example

According to the PC/SC (Personal Computer/Smart Card) standard [PS/SC], a smart card reader MAY include a serial number. This attribute (VENDOR-IFD-SERIAL) is associated to the tag 0x0103 in the class VENDOR-INFO.

1.3.2 SEID for Secure Elements

According to the Global Platform standard [GP] the Issuer Security Domain (ISD) manages applications lifecycle (downloading, activation, deletion). The command 'initialize update' is used to start a mutual authentication between the administration entity and the secure element; it collects a set of data whose first ten bytes are called the 'key diversification data'. This information is used to compute symmetric keys, and according for example to [EMV] MAY comprise a serial number.

1.4 APDUs

According to the [ISO7816] standards secure element process ISO7816 request messages and return ISO7816 response messages, named APDUs (application protocol data unit).

1.4.1 ISO7816 APDU request

An APDU request comprises two parts: a header and an optional body.

The header is a set of four or five bytes noted CLA INS P1 P2 P3

- CLA indicates the class of the request, and is usually bound to standardization committee (00 for example means ISO request).
-INS indicates the type of request, for example B0 for reading or D0 for writing.
- P1 P2 gives additional information for the request (such index in a file or identifier of cryptographic procedures)
- P3 indicates the length of the request body (from P3=01 to P3=FF), or the size of the expected response body (a null value meaning 256 bytes). Short ISO7816 requests may comprise only 4 bytes
- The body may be empty. Its maximum size is 255 bytes

1.4.2 ISO7816 APDU response

An APDU response comprises two parts an optional body and a mandatory status word.

- The optional body is made of 256 bytes at the most.

- The response ends by a two byte status noted SW. SW1 refers the most significant byte and SW2 the less significant byte.

An error free operation is usually associated to the 9000 status word. Following are some interpretations of the tuple SW1, SW2 according to various standards:

- '61' 'xx', indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA=00, INS=C0, P1=P2=00, P3=XX)
- '9F' 'xx', indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA=00, INS=C0, P1=P2=00, P3=XX)
- '6C' 'XX', the P3 value is wrong, request must be performed again with the LE parameter value sets to 'XX'
- '6E' 'XX', wrong instruction class (CLA) given in the request
- '6D' 'XX', unknown instruction code (INS) given in the request
- '6B' 'XX', incorrect parameter P1 or P2
- '67' 'XX', incorrect parameter P3

- '6F' 'XX', technical problem, not implemented...

2 The RACS protocol

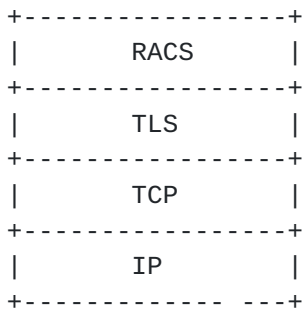

Figure 2. The RACS stack

The RACS protocol works over the TCP transport layer and is secured by the TLS protocol. The TLS client (i.e. the RACS client) MUST be authenticated by a certificate.

One of the main targets of the RACS protocol is to efficiently push a set of ISO7816 requests towards a secure element in order to perform cryptographic operations in the user's plane. In that case a RACS request typically comprises a prefix made with multiple ISO7816 requests and a suffix that collects the result of a cryptographic procedure.

The mandatory use of TLS with mutual authentication based on certificate provides a simple and elegant way to establish the credentials of a RACS client over the GoSE. It also enables an easy splitting between users' and administrators' privileges.

2.1 Structure of RACS request

A RACS request is a set of command lines, encoded according to the ASCII format. Each line ends by the Cr (carriage return) and line feed (Lf) characters. The RACS protocol is case sensitive.

Each command is a set of tokens (i.e. words) separated by space (0x20) character(s).

The first token of each line is the command to be executed.

A command line MAY comprise other tokens, which are called the command parameters.

A RACS request MUST start by a BEGIN command and MUST end by an END command.

Each command line is associated to an implicit line number. The

BEGIN line is associated to the zero line number.

The processing of a RACS request is stopped after the first error. In that case the returned response contained the error status induced by the last executed command.

2.2 Structure of a RACS response

A RACS response is a set of lines, encoded according to the ASCII format. Each line ends by the Cr (carriage return) and line feed (Lf) characters. The RACS protocol is case sensitive.

Each line is a set of tokens (i.e. words) separated by space (0x20) character(s).

The first token of each line is the header.

The second token of response each line is associated command line number

A response line MAY comprise other tokens, which are called the response parameters.

Three classes of headers are defined BEGIN, END and Status.

A RACS response MUST start by a BEGIN header and MUST end by an END header. It comprises one or several status lines.

2.2.1 BEGIN Header

This header starts a response message.

It comprises an optional parameter, an identifier associated to a previous request message.

2.2.2 END Header

This header ends a response message.

2.2.3 Status line

A status header indicates a status line.

It begins by the character '+' in case of success or '-' if an error occurred during the RACS request execution. It is followed by an ASCII encoded integer, which is the value of the status.

The second mandatory token of a status line is the command line number (starting from zero)

A status line MAY comprise other tokens, which are called the response parameters.

2.2.4 Examples of RACS responses:

```
BEGIN CrLf
+001 000 Success CrLf
END CrLf

BEGIN moon1969 CrLf
-301 007 Illegal command, BEGIN condition not satisfied at line 7
END CrLf

BEGIN Asterix237 CrLf
+006 001 [ISO7816-Response] CrLf
END CrLf

BEGIN CrLf
-100 002 Unknown command at line 2 CrLf
END CrLf

BEGIN CrLf
-606 001 Unauthorized command APDU command at line 1
END CrLf

BEGIN CrLf
-706 001 SEID Already in use, APDU command at line 1
END CrLf
```

2.3 RACS request commands

2.3.1 BEGIN

This command starts a request message. A response message is returned if an error is detected.

An optional parameter is the request identifier, which MUST be echoed in the parameter of the first response line (i.e. starting by the BEGIN header).

2.3.2 END

This command ends a request message. It returns the response message triggered by the last command.

Example1
========
Request:
BEGIN CrLf
END CrLf

Response:
BEGIN CrLf
+001 000 Success CrLf
END CrLF

Example2
========
Request:
BEGIN Marignan1515 CrLf
APDU ASTERIX-CRYPTO-MODULE [ISO7816-Request] CrLf
END CrLf

Response:
BEGIN Marignan1515 CrLf
+006 001 [ISO7816-Response] CrLf
END CrLf

2.3.3 The APPEND parameter

The APPEND parameter MAY be used in all command lines, excepted BEGIN and END. The APPEND parameter MUST be the last parameter of a command line.
By default a response message returns only the last status line.
When APPEND is inserted, the command line, if executed, MUST produce a status line.

Example

Request:
BEGIN SanchoPanza CrLf
APDU 100 [ISO7816-Request-1] CrLf
APDU 100 [ISO7816-Request-2] CrLf
END CrLf

Response:
BEGIN SanchoPanza CrLf
+006 002 [ISO7816-Response-2] CrLf
END CrLf

Request:
BEGIN DonQuichotte CrLf
APDU 100 [ISO7816-Request-1] APPEND CrLf
APDU 100 [ISO7816-Request-2] APPEND CrLf
END CrLf

```
Response:
BEGIN DonQuichotte CrLf
+006 001 [ISO7816-Response-1] CrLf
+006 002 [ISO7816-Response-2] CrLf
END CrLf
```

2.3.4 GET-VERSION

This command requests the current version of the RACS protocol. The returned response is the current version encoded by two integer separated by the '.' character. The first integer indicates the major version and the second integer gives the minor version.

This draft version is 0.2

```
Example
=======
Request:
BEGIN CrLf
GET-VERSION CrLf
END CrLf

Response:
BEGIN CrLf
+002 001 1.0 CrLf
END CrLf
```

2.3.5 SET-VERSION

This command sets the version to be used for the RACS request. An error status is returned by the response if an error occurred.

```
Example 1
=========
Request:
BEGIN CrLf
SET-VERSION 2.0 CrLf
END CrLf

Response:
BEGIN CrLf
-403 001 Error line 1 RACS 2.0 is not supported CrLf
END CrLf

Example 2
=========
Request:
BEGIN CrLf
SET-VERSION 1.0 CrLf
```

END CrLf

Response:
BEGIN CrLf
+003 001 RACS 1.0 has been activated CrLf
END CrLf

2.3.6 LIST

This command requests the list of SEID plugged in the GoSE.

It returns a list of SEIDs separated by space (0x20) character(s).

Some SEID attributes MAY be built from a prefix and an integer suffix (such as SE#100 in which SE# is the suffix and 100 is the integer suffix. A list of non-consecutive SEID MAY be encoded as prefix[i1;i2;..;ip] where i1,i2,ip indicates the integer suffix. A list of consecutive SEID could be encoded as prefix[i1-ip] where i1,i2,ip indicates the integer suffix.

Example 1
=========
Request:
BEGIN CrLf
LIST CrLf
END CrLf

Response:
BEGIN CrLf
+004 001 SEID1 SEID2 CR LF
END CrLf

Example 2
=========
Request:
BEGIN CrLf
LIST CrLf
END CrLf

Response:
BEGIN CrLf
+004 001 Device[1000-2000] SerialNumber[567;789;243] CrLf
END CrLf

2.3.7 RESET

This command resets a secure element. The first parameter gives the secure element identifier (SEID). An optional second parameter specifies a warm reset. The default behavior is a cold reset.
The response status indicates the success or the failure of this

operation.

Syntax: RESET SEID [WARM] CrLf

Example 1
=========
Request:
BEGIN CrLf
RESET device#45 CrLf
END CrLf

Response:
BEGIN CrLf
+005 001 device#45 Reset Done
END CrLf

Example 2
=========
Request:
BEGIN CrLf
RESET device#45 CrLf
END CrLf

Response:
BEGIN CrLf
-705 001 error device#45 is already in use
END CrLf

Example 3
=========
Request:
BEGIN CrLf
RESET device#45 WARM CrLf
END CrLf

Response:
BEGIN CrLf
+005 001 device#45 Warm Reset Done CrLf
END CrLf

2.3.8 APDU

This command sends an ISO7816 request to a secure element or a set of ISO7816 commands.

The first parameter specifies the SEID.
The second parameter is an ISO7816 request.
Three optional parameters are available; they MUST be located after the second parameter.

- CONTINUE=value, indicates that the next RACS command will be executed only if the ISO7816 status word (SW) is equal to a given value. Otherwise an error status is returned.
- MORE=value, indicates that a FETCH request will be performed (i.e. a new ISO7816 request will be sent) if the first byte of the ISO7816 status word (SW1) is equal to a given value.
- FETCH=value fixes the four bytes of the ISO7816 FETCH request (i.e. CLA INS P1 P2). The default value (when FETCH is omitted) is 00C00000 (CLA=00, INS=C0, P1=00, P2=00)

When the options CONTINUE and MORE are simultaneously set the SW1 byte is first checked. If there is no match then the SW word is afterwards checked.

The ISO7816 6Cxx status MUST be autonomously processed by the GoSE.

SYNTAX
APDU SEID ISO7816-REQUEST [CONTINUE=SW] [MORE=SW1] [FETCH=CMD] CrLf

The returned response is the ISO7816 response. If multiple ISO7816 requests are executed (due to the MORE option), the bodies are concatenated in the response, which ends by the last ISO7816 status word.

The pseudo code of the APDU command is the following :

```
1. BODY = empty;
2. SW   = empty;
3. DoIt = true;
3. Do
4. { iso7816-response = send(iso7816-request);
5.   body || sw1 || sw2 = iso7816-response;
6.   If ( (first request) && (iso7816-request.size==5) &&
          (body==empty) && (sw1==6C) )
8.   { iso7816-request.P3 = sw2 ; }
6.   Else
7.   { SW = sw1 || sw2
8.     BODY = BODY || body;
9.     If (sw1 == MORE)
10.    { iso7816-request = FETCH || sw2 ; }
11.    Else
12.    { DoIt=false;}
13.   }
14. }
15. While (DoIt == true)

16. iso7816-response = BODY || SW ;
17. If (SW != CONTINUE) Error   ;
```

18. Else No Error;

Example 1
=========
Request:
BEGIN CrLf
APDU SEID ISO7816-REQUEST CrLf
END CrLf

Response:
BEGIN CrLf
+006 001 ISO7816-RESPONSE CrLf
END CrLf

Example 2
=========
Request:
BEGIN CrLf
APDU SEID ISO7816-REQUEST CrLf
END CrLf

Response:
BEGIN CrLf
-706 001 error SEID is already used CrLf
END CrLf

Example 3
=========
Request:
BEGIN CrLf
APDU SEID ISO7816-REQUEST CrLf
END CrLf

Response:
BEGIN CrLf
-606 001 error access unauthorized access CrLf
END CrLf

Example 4
=========
BEGIN CrLf
APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQUEST-2 CrLf
END CrLf

Response:
BEGIN CrLf
+006 002 ISO7816-RESPONSE-2 CrLf
END CrLf

Example 5
=========
BEGIN CrLf
APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQUEST-2 CrLf
END CrLf

Response:
BEGIN CrLf
-006 001 Request Error line 1 wrong SW CrLf
END CrLf

Example 6
=========
BEGIN CrLf
APDU SEID ISO7816-REQ-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQ-2 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQ-3 CONTINUE=9000 MORE=61 FETCH=00C00000 CrLf
END CrLf

Response:
BEGIN CrLf
+006 003 ISO7816-RESP-3 CrLf
END CrLf

Multiple ISO7816 requests have been performed by the third APDU command according to the following scenario :
- the ISO7816-REQ-3 request has been forwarded to the secure element (SEID)
- the ISO 7816 response comprises a body (body-0) and a status word (SW-0) whose first byte is 0x61, and the second byte is SW2-0
- the FETCH command CLA=00, INS=00, P1=00, P2=00, P3=SW2-0 is sent to the secure element
- the ISO 7816 response comprises a body (body-1) and a status word (SW-1) set to 9000

The RACS response is set to
+006 003 body-0 || body-1 || SW-1 CrLf
where ||indicates a concatenation operation.

2.3.9 SHUTDOWN

This command powers down a secure element. The first parameter gives the secure element identifier (SEID).

Syntax: SHUTDOWN SEID CrLf

Example
=========
Request:
BEGIN Goodbye CrLf
SHUTDOWN device#45 CrLf
END CrLf

Response:
BEGIN Goodbye CrLf
+007 001 device#45 has been powered down CrLf
END CrLf

2.3.10 POWERON

This command powers up a secure element. The first parameter gives the secure element identifier (SEID).

Syntax: POWERON SEID CrLf

Example 1
=========
Request:
BEGIN CrLf
POWERON device#45 CrLf
END CrLf

Response:
BEGIN CrLf
+008 001 device#45 Has been powered up CrLf
END CrLf

Example 2
=========
Request:
BEGIN CrLf
POWERON device#45 CrLf
END CrLf

Response:
BEGIN CrLf
-708 001 error device#45 is already in use CrLf
END CrLf

Example 3
=========
Request:
BEGIN CrLf
POWERON device#45 CrLf

END CrLf

```
Response:
BEGIN CrLf
-608 001 error unauthorized access CrLf
END CrLf
```

2.3.11 ECHO

This command echoes a token. The first parameter is the token (word) to be echoed by the response.

Syntax: ECHO SEID CrLf

Example 1

```
Request:
BEGIN TestEcho CrLf
ECHO Hello CrLf
END CrLf

Response:
BEGIN TestEcho CrLf
+009 001 Hello CrLf
END CrLf
```

Example 2

```
Request:
BEGIN ResetSEID CrLf
POWERON device#45 CrLf
ECHO Done CrLf
END CrLf

Response:
BEGIN ResetSEID CrLf
+009 001 Done CrLf
END CrLf
```

2.3.12 SEN

This command associates Secure Element Name (SEN) to SEID.
Secure Element Name are defined in [IOSE]

The first parameter (mandatory) is the SEID. By default the SEN is found in the ISO7816 ATR, and the TLS-SE application is the secure element default application.

The second parameter (optional) is the SEN. This option sets the

SEN, and discards the ATR content.

The third parameter (optional) is the TLS-SE Application IDentifier (AID).

Syntax: SEN SEID [SEN] [AID] CrLf

Example 1
=========

Request:
BEGIN CrLf
SEN mySEID CrLf
END CrLf

Response:
BEGIN CrLf
+010 001 SEN= key1.com AID= default
END CrLf

Example 2
=========

Request:
BEGIN CrLf
SEN mySEID key1.com CrLf
END CrLf

Response:
BEGIN CrLf
+010 001 SEN= key1.com AID= default CrLf
END CrLf

Example 3
=========

Request:
BEGIN CrLf
SEN mySEID key1.com 010203040500 CrLf
END CrLf

Response:
BEGIN CrLf
+010 001 SEN= key1.com AID= 010203040500 CrLf
END CrLf

Example 4
=========

```
Request:
BEGIN CrLf
SEN wrongSEID key1.com CrLf
END CrLf

Response:
BEGIN CrLf
-410 001 SEN invalid SEID (wrongSEID) CrLf
END CrLf
```

2.3.13 GET-SEN

This command gets Secure Element Name (SEN) associated to SEID. Secure Element Name are defined in [IOSE]

Syntax: GET-SEN SEID CrLf

Example 1
=========

```
Request:
BEGIN CrLf
GET-SEN mySEID CrLf
END CrLf

Response:
BEGIN CrLf
+011 001 key1.com [AID= default]
END CrLf
```

Example 2
=========

```
Request:
BEGIN CrLf
GET-SEN mySEID CrLf
END CrLf

Response:
BEGIN CrLf
+011 001 key1.com [AID= 010203040500]
END CrLf
```

Example 3
=========

```
Request:
```

BEGIN CrLf

```
GET-SEN wrongSEID CrLf
END CrLf

Response:
BEGIN CrLf
-511 001 GET-SEN invalid SEID (wrongSEID)
END CrLf
```

2.4 Status header encoding

The first token of a response line is the status header. It begins by a '+' or a '-' character, and comprises three decimal digits (xyz).

The first digit (x) MUST indicates an event class.
The second and third digits (yz) MAY indicate a command class.

2.4.1 Event class

This draft only defines the meaning of the first digit located at the left most side.

```
+0yz: No error
-0yz: Command execution error
-1yz: Unknown command, the command is not defined by this draft
-2yz: Not implemented command
-3yz: Illegal command, the command can't be executed
-4yz: Not supported parameter or parameter illegal value
-5yz: Parameter syntax error or parameter missing
-6yz: Unauthorized command
-7yz: Already in use, a session with this SE is already opened
-8yz: Hardware error
-9yz: System error
```

2.4.2 Command class

The second and third digits (yz) MAY indicates the command that trigged the current line status

```
01 BEGIN
02 GET-VERSION
03 SET-VERSION
04 LIST
05 RESET
06 APDU
07 SHUTDOWN
08 POWERON
09 ECHO
10 SEN
```

3 URI for the GoSE

The URI addressing the resources hosted by the GoSE is represented by the string:

RACS://GoSE-Name:port/?request

where request is the RACS request to be forwarded to a the GoSE.

RACS command lines are encoded in a way similar to the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters, is written according to the URL encoding rules. End of line characters, i.e. carriage return (Cr) and line feed (Lf) are omitted.

As a consequence a request is written to the following syntax cmd1=cmd1-parameters&cmd2=cmd2-parameters

Example:
RACS://GoSE-Name:port/?BEGIN=&APDU=SEID%20[ISO7816-REQUEST]&END=

4 HTTP interface

A GoSE SHOULD support an HTTP interface. RACS requests/responses are transported by HTTP messages. The use of TLS is mandatory.

4.1 HTTPS Request

https://GoSE-Name:port/RACS?request

where request is the RACS request to be forwarded to a secure element (SEID)

The RACS request is associated to an HTML form whose name is "RACS". The request command lines are encoded as the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters is written according to the URL encoding rules. End of line characters, i.e. carriage return (Cr) and line feed (Lf) are omitted.

As a consequence a RACS request is written as https://GoSE-Name/RACS?cmd1=cmd1-parameters&cmd2=cmd2-parameters

Example:

https://GoSE-Name/RACS?BEGIN=&APDU=SEID%20[ISO7816-REQUEST]&END=

4.2 HTTPS response

The RACS response is returned in an XML document.

The root element of the document is <RACS-Response>

The optional parameter of the BEGIN header, is the content of the <begin> element.

Each status line is the content of the <Cmd-Response> element, which includes the following information :

- The status header is the content of the <status> element.

- The line number is the content of the <line> element.

- The other parameters of the status line are the content of the <parameters> element.

The END header is associated to the element <end>

End of line, i.e. carriage return (Cr) and line feed (Lf) characters are omitted.

As a consequence a RACS response is written as :

```
<RACS-Response>
<begin>Optionnal-ID</begin>
<Cmd-Response
<status>+000</status>
<line>001</line>
<parameters>other parameters of the RACS response</parameters>
</Cmd-Response>
<end></end>
</RACS-Response>
```

5 Security Considerations

5.1 Authorization

A RACS client MUST be authenticated by an X509 certificate.

The GoSE software MUST provide a mean to establish a list of SEIDs that can be accessed from a client whose identity is the CommonName (CN) attribute of its certificate. It MAY allocate a UserID (UID), i.e. an integer index from the certficate common name.

5.2 Secure Element access

The GoSE MUST manage a unique session identifier (SID) for each TLS session. The SID is bound to the client's certificate CommonName

(SID(CN))

A secure element has two states, unlocked and locked. In the locked state the secure element may be only used by the SID that previously locked it.

The first authorized command that successfully accesses to a SEID (either POWERON ,RESET, APDU) locks a secure element (SEID) with the current session (SID).

The SHUTDOWN command MUST unlock a secure element (SEID).

The end of a TLS session MUST unlock all the secure elements locked by the session.

5.3 Applications security policy

According to the [ISO7816] standards each Application embedded within a secure element (associated to a SEID) is identified by an AID parameter (16 bytes at the most)

The RACS server SHOULD support the following facilities

5.3.1 Users-Table

Each CN (the Users-Table primary key) is associated to a list of SEIDs whose access is authorized.

5.3.2 SEID-Table

Each AID (the SEID-Table primary key) is associated to a list of CNs whose access is authorized.

5.3.3 APDU-Table

For a given AID and an authorized CN, an APDU-Table MAY be available. This table acts as a firewall, which defined a set of forbidden ISO7816 commands.

For example this filter could be expressed as a set of the four first bytes of an APDU-Prefix (CLA INS P1 P2) and a four bytes Mask An ISO7816-Request is firewall if:

ISO7816-Request AND Mask IsEQUAL to APDU-Prefix

5.4 Overview of the security policy

The summary of the security policy is illustrated by the figure 3.

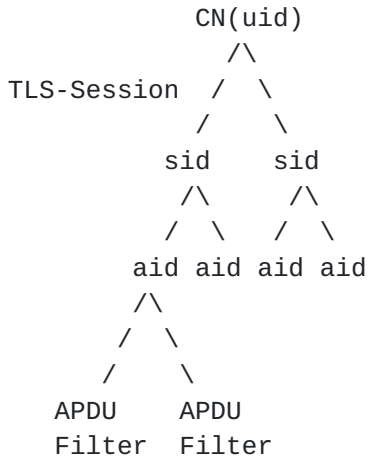

Figure 3. Summary of the security policy

6 IANA Considerations

This draft does not require any action from IANA.

7 References

7.1 Normative References

[TLS 1.0] Dierks, T., C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999

[TLS 1.1] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006

[TLS 1.2] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5746, August 2008

[TLS 1.3] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO)

7.2 Informative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[GP] Global Platform Standards, http://www.globalplatform.org

[EUROSMART] The EUROSMART association, http://www.eurosmart.com

[PC/SC] The PC/SC workgroup, http://www.pcscworkgroup.com

[EMV] EMV Card Personalization Specification, Version 1.1, July 2007

[OPENRACS] https://github.com/purien, open RACS implementation for Win32, Ubuntu, Raspberrypi

[IOSE] Internet of Secure Elements, draft-urien-coinrg-iose-03.txt, September 2021

8 Authors' Addresses

Pascal Urien
Telecom Paris
19 place Marguerite Perey
91120 Palaiseau Phone: NA
France Email: Pascal.Urien@telecom-paris.fr