# Virtual POS

Integration Manual - Redirection

**Version:** 3.2
**Date:** 31/01/2024
**Reference:** RS.TE.CEL.MAN.0039

Redsys, Servicios de Procesamiento, S.L. – c/ Francisco Sancha, 12 – 28034 Madrid (Spain)

www.redsys.es

## Version control

| Version | Date | Concerns | Brief description of the change |
|---|---|---|---|
| 1.0 | 01/06/2018 | All | First version |
| 1.1 | 24/09/2018 | Point 8.1 | ASCII del Ds_Merchant_Order code |
| 1.2 | 07/11/2018 | Point 8.6 | The option of retrying the payment is added |
| 1.3 | 18/12/2018 | Point 8.6 | The option to retry the payment is modified |
| 1.4 | 10/01/2019 | Point 8.1, 8.2 and 8.6 | The Ds_Merchant_Paymethods field is added, a correction on the submitting of the response code is added and the requirements of the merchant in the payment retries are specified |
| 1.5 | 29/01/2019 | Point 8.1 | The Ds_Merchant_TransactionType parameter is added the type of Authorization Withdrawal |
| 1.6 | 14/03/2019 | Point 8 | Points 8.1, 8.2 are redone and the Currency and Language points are eliminated |
| 2.0 | 27/03/2019 | Various points | Added information about EMV3DS and PSD2<br>Added reference to the new documents TPV-Virtual GuiaErroresSIS.xlsx and TPV-Virtual Parámetros Entrada-Salida.xlsx |
| 2.1 | 12/04/2019 | Test environment<br>Error code | Added test cards for EMV3DS<br>The SIS errors spreadsheet is modified to include it in the input-output parameters spreadsheet |
| 2.2 | 02/07/2019 | PSD2 and MIT | COF and MIT operations |

| | | | |
|---|---|---|---|
| 2.3 | 04/10/2019 | The entire document | Adaptations are added for EVM3DS 2.2 |
| 2.4 | 12/11/2019 | | The features and specifications that will be available in the future are marked as advances |
| 2.5 | 16/12/2019 | Point 6,7,8,9 and 10 | PSP connection Added<br>Modification in Exemption and tokenization<br>Clarification and examples for EMV3DS advancement functionalities.<br>Modification of test cards for EMV3DS advances. |
| 2.6 | 16/03/2020 | Point 10 | Point 10 is restructured |
| 3.0 | 01/10/2020 | All Text | The "advance" mark is removed from the functionalities affected by PSD2.<br>The integration aid libraries are included in Annex 2.<br>Point 9 is pending of brand specifications.<br>In the test section, cards of the different card brands are included. |
| 3.1 | 16/01/2024 | Point 7<br>Point 10 | Deleting 3DSecure version 1<br>3DSecure version 1 test card delete |

| | | | |
|---|---|---|---|
| 3.2 | 31/01/2024 | Point 10 | Added DCC test card in pounds |

CONTENTS

# 1. Introduction

## 1.1 Purpose

This document includes the technical aspects needed for a merchant to integrate with the Virtual POS through a redirection connection of the buyer client's browser.

This connection system allows the client session to be transferred to the Virtual POS, so that the selection of the payment method and the introduction of data are carried out in the secure environment of the Virtual POS server and outside the responsibility of the merchant. In addition to the simplicity of implementation for the merchant and the reliability regarding the responsibility of payment data, this mode of connection offers the possibility of authenticating the cardholder through the 3DS protocol, which allows the cardholder to authenticate directly with the issuing bank of his/her card at the time of making the transaction that gives greater security to purchases.

*NOTE: the connection requires the use of a signature system based on HMAC SHA-256, which authenticates each other the merchant server with the Virtual POS. To develop the calculation of this type of signature, the merchant can carry out the development by itself using the standard functions of the different development environments, although to allow the developments we offer libraries (PHP, JAVA and.NET) whose use is detailed in this guide and which are available at the following address:*

https://pagosonline.redsys.es/descargas.html

***IMPORTANT NOTE: On the occasion of the full entry into force of the European Payments Directive PSD2 in 2020, some new features and technical specifications that will be available throughout 2020 are included in this guide, to allow the preparation of works in those cases of merchants wishing to incorporate certain functionalities into their payment operations, especially in relation to the authentication and exemption management of the authentication that the PSD2 considers.***

## 1.2 Definitions, acronyms, and abbreviations

- **SIS**. Redsys Integrated Server (Virtual POS Server).
- **SCA**. Strong Customer Authentication. Cardholder's enhanced authentication.
- **Frictionless**. Authentication without cardholder participation.
- **Challenge**. Enhanced authentication of the cardholder (through OTP, static password, biometrics, etc.).
- **PSD2**. Payment Service Providers. European regulation for digital payment services.
- **3DSecure**: Security system for online payments. Hereinafter EMV3DS
- **EMV3DS**: Acronym to identify the new version of 3DSecure in the Virtual POS.

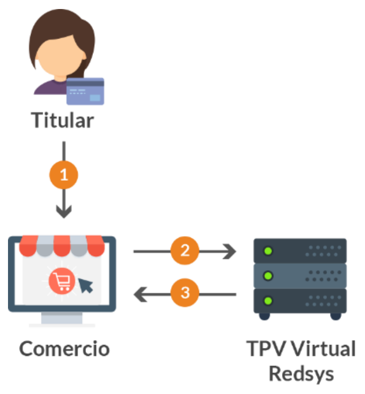
- **MIT**. Merchant Initiated Transaction. It refers to transactions initiated directly by the merchant without the cardholder being present, such as in the case of recurring payments.
- **COF**. Credentials On File. It refers to the operation in which the **card data are stored for future use.**
- **DCC.** Dynamic Currency Conversion. It allows the cardholder to make the payment in his/her own currency instead of the one defined in the terminal.

## 1.3 References

- Documentation for integration with the SIS
- TPV-Virtual Guía SIS.
- TPV-Virtual Parámetros Entrada-Salida.xlsx
- COF ECOM Specifications

# 2. Flow Overview

The following scheme presents the general flow of an operation performed with the Virtual POS.



1. The cardholder selects the products that he/she wishes to buy in the merchant.
2. The merchant redirects the client's browser session to the Redsys URL. In this URL the client enters the card details.
3. The virtual POS reports the merchant of the result of the operation and Redsys returns the client's browser session to the merchant to continue browsing its online store.

# 3. Request submission to the Virtual POS

The merchant submits the payment request data encoded in UTF-8 to the Virtual POS through the cardholder's browser. To that end, a form with the following fields must be prepared:

- Ds_SignatureVersion: Constant value indicating the signature version that is being used.
- Ds_MerchantParameters: String in JSON format with all the parameters of the request encoded in Base 64 and without carriage returns. See *Tpv Virtual Parámetros Entrada-Salida. xlsx*, which includes the list of parameters that can be sent in a payment request.
- Ds_Signature: Signature of the data sent. It is the result of the HMAC SHA256 of the JSON string encoded in Base 64 sent in the previous parameter.

This form must be sent to the following URLs depending on whether you want to make a request for real tests or operations:

| URL Connection | Environment |
|---|---|
| https://sis-t.redsys.es:25443/sis/realizarPago | Tests |
| https://sis.redsys.es/sis/realizarPago | Real |

Example of payment form **without submitting card data:**

```
<form name="from" action="https://sis-t.redsys.es:25443/sis/realizarPago" method="POST">
        <input type="hidden" name="Ds_SignatureVersion" value="HMAC_SHA256_V1"/>

        <input type="hidden" name="Ds_MerchantParameters" value="
        eyJEU19NRVJDSEFOVF9BTU9VTlQiOiI5OTkiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjEyMzQ1Njc4OTAiLCJEU19NRV
        JDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX
        01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDS
        EFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiw
        iRFNfTUVSQ0hBTlRfVVJMT0siOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsT0sucGhwIiwiRFNfTUVSQ0h
        BTlRfVVJMS08iOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsS08ucGhwIn0="/>
        <input type="hidden" name="Ds_Signature" value="PqV2+SF6asdasMjXasKJRTh3UIYya1hmU/igHkzhC+R="/>
</form>
```

Example of payment form **with submitting card data**:

```
<form name="from" action="https://sis-t.redsys.es:25443/sis/realizarPago" method="POST">
        <input type="hidden" name="Ds_SignatureVersion" value="HMAC_SHA256_V1"/>
        <input type="hidden" name="Ds_MerchantParameters" value="
        eyJEU19NRVJDSEFOVF9BTU9VTlQiOiIxNDUiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjE0NDYwNjg1ODEiLCJEU19NR
        VJDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRT
        X01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJD
        SEFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIi
        wiRFNfTUVSQ0hBTlRfVVJMT0siOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsT0sucGhwIiwiRFNfTUVSQ0
        hBTlRfVVJMS08iOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsS08ucGhwIiwiRFNfTUVSQ0hBTlRfUEFOIjoi
        NDU0ODgxMjA0OTQwMDAwNCIsIkRTX01FUkNIQU5UX0VYUElSWURBVEUiOiIxNTEyIiwiRFNfTUVSQ0hBTlRfQ
        1ZWMiI6IjEyMyJ9"/>
        <input type="hidden" name="Ds_Signature" value="PqV2+SF6asdasMjXasKJRTh3UIYya1hmU/igHkzhC+R="/>
</form>
```

*Note: It is important to note that whenever the merchant processes card data, PCI-DSS compliance is required*

To allow the integration of the merchant, the following explains in detail the steps to follow to configure the payment request form.

## 3.1 Identify the signature algorithm version to use

The request must identify the specific version of the algorithm that is being used for the signature. Currently the value **HMAC_SHA256_V1** is used to identify the version of all requests, so this will be the value of the **Ds_SignatureVersion** parameter, as can be seen in the example form shown at the beginning of section 3.

## 3.2 Configure the request data string

A string must be configured with all the request data in JSON format. The name of each parameter must be indicated in uppercase or with "CamelCase" structure (For example: DS_MERCHANT_AMOUNT or Ds_Merchant_Amount).

Merchants that use special operations such as "Payment by reference" (1-Click Payment) must include the specific parameters of their operation as part of the JSON object.

The complete list of parameters that can be included in the request are available in the document *"TPV-Virtual Parámetros Entrada-Salida.xlsx"*.

Below are some examples of the JSON object of a request:

Example without submitting card data:

```
{"DS_MERCHANT_AMOUNT":"145","DS_MERCHANT_ORDER":"1446117555","DS_MERCHANT_MERCHANTC
ODE":"999008881","DS_MERCHANT_CURRENCY":"978","DS_MERCHANT_TRANSACTIONTYPE":"0","DS_MER
```

CHANT_TERMINAL":"1","DS_MERCHANT_MERCHANTURL":"http:\/\/www.prueba.com\/urlNotificacion.php"
,"DS_MERCHANT_URLOK":"http:\/\/www.prueba.com\/urlOK.php","DS_MERCHANT_URLKO":"http:\/\/www
.bancsabadell.com\/urlKO.php"}

Example with submitting card data:

{"DS_MERCHANT_AMOUNT":"145","DS_MERCHANT_ORDER":"1446068581","DS_MERCHANT_MERCHANTC
ODE":"999008881","DS_MERCHANT_CURRENCY":"978","DS_MERCHANT_TRANSACTIONTYPE":"0","DS_MER
CHANT_TERMINAL":"1","DS_MERCHANT_MERCHANTURL":"http:\/\/www.prueba.com\/urlNotificacion.php"
,"DS_MERCHANT_URLOK":"http:\/\/www.prueba.com\/urlOK.php","DS_MERCHANT_URLKO":"http:\/\/www
.prueba.com\/urlKO.php","DS_MERCHANT_PAN":"454881********04","DS_MERCHANT_EXPIRYDATE":"151
2","DS_MERCHANT_CVV2":"123"}

Once the JSON string is configured with all the fields, it is necessary to code it in BASE64 without carriage returns to ensure that it remains constant and is not altered by the client / buyer's browser.

The JSON objects that have just been displayed encoded in BASE64 are shown below:

JSON example coded without submitting card data:

eyJEU19NRVJDSEFOVF9BTU9VTlQiOiI5OTkiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjEyMzQ1Njc4OTAiLCJEU19NRV
JDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX
01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDS
EFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiw
iRFNfTUVSQ0hBTlRfVVJMT0siOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsT0sucGhwIiwiRFNfTUVSQ0h
BTlRfVVJMS08iOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsS08ucGhwIn0

JSON example with submitting card data:

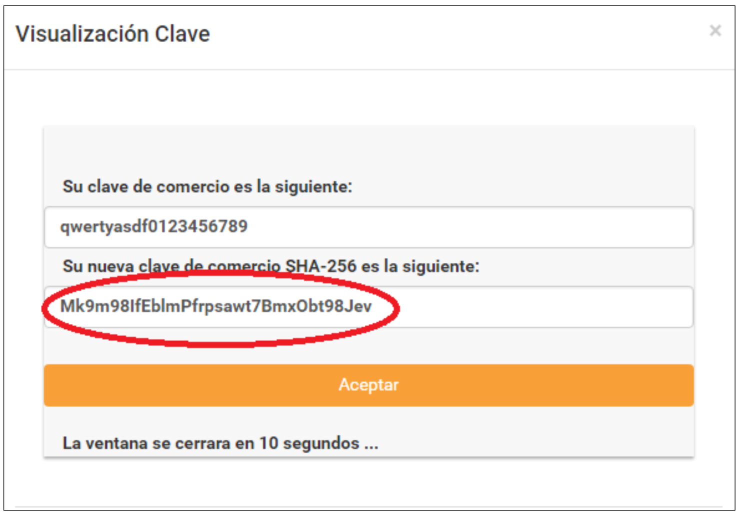
eyJEU19NRVJDSEFOVF9BTU9VTlQiOiIxNDUiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjE0NDYwNjg1ODEiLCJEU19NR
VJDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRT
X01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJD
SEFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIi
wiRFNfTUVSQ0hBTlRfVVJMT0siOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsT0sucGhwIiwiRFNfTUVSQ0
hBTlRfVVJMS08iOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsS08ucGhwIiwiRFNfTUVSQ0hBTlRfUEFOIjoi
NDU0ODgxMjA0OTQwMDAwNCIsIkRTX01FUkNIQU5UX0VYUElSWURBVEUiOiIxNTEyIiwiRFNfTUVSQ0hBTlRfQ
1ZWMiI6IjEyMyJ9

The string resulting from coding in BASE64 will be the value of the **Ds_MerchantParameters** parameter, as can be seen in the example form shown at the beginning of section 3.

*NOTE: You can use help libraries to generate this field, see Annex 2- point 2.1*

## 3.3 Identify the key to be used for signing

To calculate the signature, it is necessary to use a specific key for each terminal. The key can be obtained by accessing the Management Portal of the Virtual POS, option Check Merchant data, in the "See key" section, as shown in the following picture:



Visualización Clave

Su clave de comercio es la siguiente:

qwertyasdf0123456789

Su nueva clave de comercio SHA-256 es la siguiente:

Mk9m98IfEblmPfrpsawt7BmxObt98Jev

La ventana se cerrara en 10 segundos ...

*IMPORTANT NOTE: This key must be stored on the merchant's server in the safest way possible to avoid its fraudulent use. The merchant is responsible for the proper custody and secret maintenance of this key.*

## 3.4 Sign the request details

Once the data string to be signed and the specific terminal key are configured, the signature must be calculated following the next steps:

1. A specific key is generated per operation. To obtain the derived key to be used in an operation, a 3DES encryption must be performed between the merchant key, which must be previously decoded in BASE 64, and the value of the order number of the operation (Ds_Merchant_Order).

2. The HMAC SHA256 of the value of the **Ds_MerchantParameters** parameter **and** the key obtained in the previous step is calculated.

3. The result obtained is encoded in BASE 64, and the result of the coding will be the value of the Ds_Signature parameter, as can be seen in the example form, shown at the beginning of section 3.

*NOTE: You can use help libraries to generate this field, see Annex 2- point 2.1*

# 4. Receipt of online notification

Once the transaction is done, the Virtual POS may report the merchant's server of the result of the transaction through a direct connection to the merchant's server (step 3 of the described flow in point 2). This online notification is an optional function allowing the web store to receive the result of an online transaction in real time, once the client has completed the payment process in the Virtual POS. The Virtual POS submits the online notification with the result of the transaction to the URL provided by the merchant in the payment request, in the *Ds_Merchant_MerchantURL* parameter.

For the merchant to receive this notification online it must be configured it in the Virtual POS Management portal.

The merchant that is configured to receive this notification must capture **and validate all the parameters together with the signature** of the online notification, prior to any execution on its server.

The online notification consists of an HTTP POST (Synchronous or Asynchronous Notification) with the information of the payment result, encoded in UTF-8. The following fields will be included in the POST:

- Ds_SignatureVersion: Constant value indicating the signature version that is being used.
- Ds_MerchantParameters: String in JSON format with all the parameters of the response encoded in Base 64 and without carriage returns. See *Guía Tpv Virtual Parámetros Entrada-Salida. xlsx*, this includes the list of parameters that can be used in the on-line notification.
- Ds_Signature: Signature of the data sent. Result of the HMAC SHA256 of the JSON string encoded in Base 64 sent in the previous parameter. The merchant is responsible for validating the HMAC sent by the Virtual POS to ensure the validity of the response. This validation is necessary to ensure that the data have not been altered and that the actual origin is the Virtual POS.

The Virtual POS has different types of notification:

1. **Synchronous.** It implies that the result of the purchase is first sent to the merchant and then shown to the client. Even if the merchant does not process the notification correctly, the result of the operation is kept.

2. **Asynchronous.** It implies that the result of the authorization is communicated at the same time to the merchant and to the client. Even if the merchant does not process the notification correctly, the result of the operation is kept.

*NOTE: You can use help libraries to allow the developments for the receipt of the parameters of the online notification and validation of the signature, see Annex 2 - point 2.2*

# 5. Return of the browsing control to the cardholder

Once the client has completed the payment process at the Virtual POS, browsing is redirected to the web store (step 3 of the flow described in point 2):

- At this time, a receipt will be shown to the client with the result of the operation, if the configuration of the merchant so specifies it. In addition, if the operation is denied, the client will have the option of retrying the payment with another card or other payment method, as long as the merchant's configuration allows it (the option of retrying the payment is described in detail in Annex 1 - point 1-3).
- In this step the browsing control of the cardholder is returned to the merchant. Thus, the merchant can complete the payment flow by maintaining a natural browsing sequence for the client / buyer.

This return to the web store is made to the URL communicated as a parameter in the initial call to the Virtual POS or, failing that, it is obtained from the terminal configuration in the Virtual POS management module. Different return URLs may be available depending on the outcome of the transaction (URL OK and URL KO).

Optionally, the Virtual POS may include the same fields of the online notification in the return URLs. In this case, the merchant must capture and validate said parameters prior to any execution on its server. To activate this option, the *Parameters field in the URLs = YES in the* Management portal must be configured.

*NOTE: You can use help libraries to allow the capture and validation of the browsing control return, see Annex 2 -point 2.3.*

# 6. Integration for PSPs

If you are a merchant aggregator or a PSPs, there is a specific integration so that, in this way, with a secret key for PSP you can operate on behalf of the merchants at the terminal level.

For these requests, the parameters to be sent are the same as in the usual merchant request, but these will have a signature version and signature in ANSI X9.19.

No specific flows are defined for PSP. The input and output parameters as well as the error codes can be seen in the guide *"TPV-Virtual Parámetros Entrada-Salida.xlsx.*

*NOTE: to carry out this integration, an activation by the Bank is required.*

## Settings

The merchant-terminal must be associated and configured to submit requests from a PSP. This configuration should be requested by the merchant to its Bank.

## Request and receipt of keys

Two private keys can be received with the protocol set forth by Redsys.

- Key to perform 3DES encryption of the DS_MERCHANT_PAN field (if necessary).
- Key to sign the (DS_MERCHANTPARAMETERS) request according to the ordinary X9.19

## Request submission to the Virtual POS

As in the payment request sent by a merchant, the PSP has the following fields which may vary as we have mentioned before the signature and its signature version.

- Ds_SignatureVersion: Constant value indicating the signature version that is being used. For this PSP integration the value to be used will be **T25V1.**
- Ds_MerchantParameters: String in JSON format with all the parameters of the request encoded in Base 64 and without carriage returns (the list of parameters that can be sent in a payment request is included in the Annex section).
- Ds_Signature: Signature of the data sent. It is the result of the Mac X9.19 of the JSON string encoded in Base 64 sent in the previous parameter. The value to be sent in the Ds_Signature field is obtained by converting the Mac obtained in the previous step to hexadecimal and taking the first 4 bytes (8 characters) starting from the left of the result.

## Receipt of the result

The reception of the result will be signed in the same way as the request for submission, according to the ANSI x9.19 standard.

## Example of requests

### JSON string

*{"DS_MERCHANT_AMOUNT":"145","DS_MERCHANT_ORDER":"1446068581","DS_MERCHANT_MERCHANTCODE":"999008881","DS_MERCHANT_CURRENCY":"978","DS_MERCHANT_TRANSACTIONTYPE":"0","DS_MERCHANT_TERMINAL":"1","DS_MERCHANT_MERCHANTURL":"http:\/\/www.prueba.com\/urlNotificacion.php","DS_MERCHANT_PAN":"454881********04","DS_MERCHANT_EXPIRYDATE":"1512","DS_MERCHANT_CVV2":"123"}*

### We encrypt the DS_MERCHANT_PAN parameter in 3DES (if card data are included)

| | |
|---|---|
| *Card number* | *454881********04* |
| *3DES encrypted hexadecimal card number CBC mode without vector, key F180E06B7A89A88F4A2A52C8EC1C5D1C* | *377f34989cf0ae79857a70aa45f3e4c3* |

### JSON string with encrypted PAN:

*{"DS_MERCHANT_AMOUNT":"145","DS_MERCHANT_ORDER":"1446068581","DS_MERCHANT_MERCHANTCODE":"999008881","DS_MERCHANT_CURRENCY":"978","DS_MERCHANT_TRANSACTIONTYPE":"0","DS_MERCHANT_TERMINAL":"1","DS_MERCHANT_MERCHANTURL":"http:\/\/www.prueba.com\/urlNotificacion.php","DS_MERCHANT_PAN":"377f34989cf0ae79857a70aa45f3e4c3","DS_MERCHANT_EXPIRYDATE":"1512","DS_MERCHANT_CVV2":"123"}*

The JSON object encoded in BASE64, DS_MERCHANT_PARAMETERS field, is shown below:

*eyJEU19NRVJDSEFOVF9BTU9VTlQiOiIxNDUiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjE0NDYwNjg1ODEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiwiRFNfTUVSQ0hBTlRfUEFOIjoiMzc3ZjM0OTg5Y2YwYWU3OTg1N2E3MGFhNDVmM2U0YzMiLCJEU19NRVJDSEFOVF9FWFBJUllEQVRFIjoiMTUxMiIsIkRTX01FUkNIQU5UX0NWVjIiOiIxMjMifQ==*

### Sign the request details

On the whole string obtained in the previous step (DS_MERCHANTPARAMETERS) the complete signature is calculated based on the x9.19 standard.

We obtain the MAC 5D823A402DE70705 with the encryption key 269289DA6EAD0B20928C8F2F2F6BC752 and DS_MERCHANTPARAMETERS.

The DS_SIGNATURE field will be obtained by taking the first 4 bytes (8 characters) starting from the left of the MAC result of the previous step ***5D823A40***

### Configure the request message

- *Ds_SignatureVersion: T25V1*
- *Ds_MerchantParameters:*
  *eyJEU19NRVJDSEFOVF9BTU9VTlQiOiIxNDUiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjE0NDYwNjg1ODEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxO*

*b3RpZmljYWNpb24ucGhwIiwiRFNfTUVSQ0hBTlRfUEFOIjoiMzc3ZjM0OTg5Y2YwYWU3OTg1N2E3MGFhNDVmM2U0YzMiLCJEU19NRVJDSEFOVF9FWFBJUllEQVRFIjoiMTUxMiIsIkRTX01FUkNIQU5UX0NWVjIiOiIxMjMifQ==*

- *Ds_Signature: 5D823A40*

# 7. EMV3DS Authenticated Requests

This type of integration allows payments to be made with authentication of the cardholder through the EMV3DS protocol defined by the card brands.

In order to improve the user experience in the authentication process, the parameters that have been defined by the card brands for the EMV3DS v2 version can be added to the payment request, which provide additional data on the cardholder and the device being used. In this case, the merchant must add the input parameter DS_MERCHANT_EMV3DS type JSON in the payment or preauthorization request. For more information on this input parameter, please refer to the document "*TPV-Virtual Parámetros Entrada-Salida.xlsx*".

## Payment example with additional data EMV3DS

The following is an example of the Ds_Merchant_Parameters parameter, prior to being encoded in Base 64:

```
{"DS_MERCHANT_ORDER":"1552565870",
"DS_MERCHANT_MERCHANTCODE":"999008881",
"DS_MERCHANT_TERMINAL":"999",
"DS_MERCHANT_CURRENCY":"978",
"DS_MERCHANT_TRANSACTIONTYPE":"0",
"DS_MERCHANT_AMOUNT":"1000",
"DS_MERCHANT_MERCHANTURL":"http:\/\/www.prueba.com\/urlNotificacion.php",
"DS_MERCHANT_URLOK":"http:\/\/www.prueba.com\/urlOK.php",
"DS_MERCHANT_URLKO":"http:\/\/www.bancsabadell.com\/urlKO.php"",
"DS_MERCHANT_EMV3DS": {
            "shipAddrCountry": "840",
            "shipAddrCity": "Ship City Name",
            "shipAddrState": "CO",
            "shipAddrLine3": "Ship Address Line 3",
            "shipAddrLine2": "Ship Address Line 2",
            "shipAddrLine1": "Ship Address Line 1",
            "shipAddrPostCode": "Ship Post Code",
            "cardholderName": "Cardholder Name",
            "email": "example@example.com",
            "mobilePhone": {"cc": "123","subscriber": "123456789"}
 }
}
```

# 8. PSD2 adaptations

According to the PSD2 regulation (entered into force on September 14, 2019), a European directive that aims to improve security and reinforce client authentication in electronic commerce operations. As a basic rule, authentication of the cardholder is required in all operations (SCA), however, the possibility that the merchant in the payment request asks for an exemption to avoid said authentication is also defined. To request an exemption, the merchant must include the following parameter in their requests:

| PARAMETER | POSSIBLE VALUES |
|---|---|
| DS_MERCHANT_EXCEP_SCA | LWV, TRA, MIT, COR, ATD |

- LWV (Low value transaction): low amount exemption (up to € 30, with a maximum of 5 operations or € 100 accumulated per card, these counters are controlled at the card issuer level).
- TRA (Transaction Risk Analysis): This exemption is based on a transaction risk analysis (and be considered as low risk) by the acquirer / merchant.
- MIT (Merchant Initiated Transaction): operation initiated by the merchant (without being associated with an action or event of the client, that is, without any possible interaction with the client), they are outside the scope of PSD2. This is the case of subscription payment operations, recurring operations, etc. requiring the storage of the client payment credentials (COF) or its equivalent through tokenized scheduled payment operations (use of “payment by reference” functionality in payments initiated by the merchant). All payment operations initiated by the merchant (MIT) require that the initial operation, when the client grants permission to the merchant to use his/her payment credentials, be done through an operation authenticated with SCA.

  **NOTE: To submit MIT-type operations, it is recommended to use REST integration.**
- COR (Secure Corporate Payment): exemption restricted to cases of use of a secure corporate payment protocol.
- ATD: Delegated authentication exemption. Delegated Authentication is a brand-specific program (for more information, refer the brand's documentation on this subject)

*NOTE: It should be noted that for the LWV, TRA and COR exemptions the first option will be to mark the exemption in the authentication step, to improve the user experience. This allows that if the issuer does not accept the exemption proposal and requires SCA, it can request authentication at the same time without having to reject the operation (challenge required EMV3DS).*

## 8.1 Operation of exemption requests to SCA (Strong Customer Authentication)

Although a merchant through the payment service offered by its bank may request that a payment invokes one of the SCA exemptions provided, the definite acceptance of said request to "not authenticate" the cardholder depends, ultimately, on the card issuer. This request for exemption in a transaction does not guarantee the acceptance of it by the bank issuer of the card. In case of non-acceptance, the transaction will be required to be authenticated.

For this reason, the Virtual POS, under the *best practices* of the sector and to ensure the best user payment experience, will always prioritize starting the authentication flow with the issuer (via EMV3DS, if the issuer supports it) by indicating the issuer in said request merchant's preference not to authenticate based on the requested exemption. This improves usability and if the issuer does not accept the exemption, the user shall be authenticated at the same time. If the issuer agrees to apply the requested exemption, the authentication flow is closed without requiring or displaying any screen or action to the cardholder, being a transparent process for the user.

### Request with submission of exemption

Once the exemptions that can be used are known, the merchant must add the **DS_MERCHANT_EXCEP_SCA** parameter to the transaction with one of the allowed exemptions, as indicated below:

Example:

*{"DS_MERCHANT_AMOUNT":"145","DS_MERCHANT_ORDER":"1446117555","DS_MERCHANT_MERCHANTCODE":"999008881","DS_MERCHANT_CURRENCY":"978","DS_MERCHANT_TRANSACTIONTYPE":"0","DS_MERCHANT_TERMINAL":"1","DS_MERCHANT_MERCHANTURL":"http:\/\/www.prueba.com\/urlNotificacion.php","DS_MERCHANT_URLOK":"http:\/\/www.prueba.com\/urlOK.php","DS_MERCHANT_URLKO":"http:\/\/www.bancsabadell.com\/urlKO.php",**"DS_MERCHANT_EXCEP_SCA":"TRA"}***

## 8.2 MIT- Merchant Initiated Transaction

A MIT transaction is the one initiated by the merchant itself without any possible interaction with the client. For example, monthly payment of a receipt or subscription fee. This type of operation, since the client is not present and his/her authentication is not possible, will not require authentication by the cardholder (SCA).

To correctly identify this type of transaction, the merchant must include in the payment request, the **DS_MERCHANT_EXCEP_SCA** parameter with the value **MIT** and, in addition, submit the **DS_MERCHANT_DIRECTPAYMENT** parameter with the value **true**.

These MIT transactions may be associated with an initial payment request (**Initial COF Operation**) in which the cardholder is present and grants permission to the merchant to use his/her payment data in subsequent charges, according to a service provided continuously over time. This initial operation must be authenticated with SCA and must be marked following the COF specifications. The MIT operation also requires that the COF indicator be correctly marked, according to the specific use that is being made of the stored credentials.

It must be considered that not all operations in which stored card data / credentials (COF) are used can be considered MIT. For example, the **1-click payment** operation, where the client's credentials are stored or tokenized (payment by reference), payment requests that are made using the stored credentials **CAN'T** be considered transactions initiated by the merchant since the cardholder is present and thus he/she can be authenticated. In this case, according to PSD2, and if no other exemption is applied, the use of enhanced authentication (SCA) is required.

*NOTE 1: It is recommended that the merchant use REST integration for these types of transactions since in MIT transactions there is no direct interaction with the cardholder.*

*NOTE 2: For more information on Credentials on File (COF) specifications, please, refer to COF Ecom Specifications Guide.*

*NOTE 3: The complete list of all SIS input parameters is available in the document "TPV-Virtual Parámetros Entrada-Salida.xlsx".*

# 9. Advanced EMV3DS Features (3RI- OTA)

As for this point of R3I operations, the specifications are not yet closed by the card brands. This point is temporarily deleted until we have the requirements of the card brands that allow us to implement the definitive solution for this type of transaction.

# 10. Test environment

The merchant can use the test environment to carry out the tests that it needs to verify the correct operation of its integration before implementing in the real environment.

This guide provides generic test data that can be used by any client. If the merchant is interested in carrying out these tests with its establishment data, it should contact its bank to provide access data.

The URLs to access the test environment are:

| URL Payment request |
|---|
| https://sis-t.redsys.es:25443/sis/realizarPago |
| **URL access to the Management portal** |
| https://sis-t.redsys.es:25443/canales/portal |

*NOTE: The test environment will be identical to the real environment, with the only difference that payments made in it will not be valid in accounting terms.*

**GENERIC TEST DATA**

- Merchant number (Ds_Merchant_MerchantCode): Here the number provided by the Bank must be entered (example 999008881)
- Terminal (Ds_Merchant_Terminal): Here the number provided by the bank must be provided (example 01)
- Signing key: sq7HjrUOBfKmC576ILgskD5srU870gJ7

| Test card | Description |
|---|---|
| 4918019160034602<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.1 with *threeDSMethodURL and* Frictionless authentication |
| 4548814479727229<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.1 without *threeDSMethodURL and* Frictionless authentication |

| | |
|---|---|
| 4918019199883839<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.1 with *threeDSMethodURL and* Challenge authentication |
| 4548817212493017<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.1 without *threeDSMethodURL and* Challenge authentication |
| **Denied operations** | |
| Any operation with a value of CVV2 = 999 or amount ended in 96<br>To perform a denial of operation test by response code 172, 173, or 174, simply use the code value as CVV or set an amount ending in 72, 73, or 74. This will only work in test environments. | |
| **PSD2 tests** | |
| 4548816134581156<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.2 *without threeDSMethodURL with Frictionless authentication* |
| 4548816131164386<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.2 *without threeDSMethodURL with Challenge authentication* |
| 4548815324058868<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.2 *without threeDSMethodURL. If an exemption is sent, Frictionless authentication will be performed. If no exemptions are sent it will ask for Challenge* |
| 4548815374025114<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.2 *without threeDSMethodURL. If MIT is sent it will return Frictionless, in any other case it will ask for Challenge* |
| 5576441563045037<br>Expiration: not validated<br>CVV2: other than 999 | EMV3DS 2.2 *without threeDSMethodUR. 3RI-OTA payments are accepted* |
| 4548817212493017<br>Expiration: not validated<br>CVV2: other than 999 | Card with soft decline. Denial 195, if the authorization is not authenticated. |
| **DCC tests** | |
| 5424180805648190<br>Expiration: not validated<br>CVV2: other than 999 | Master card with DCC and EMV3DS 2 Frictionless |
| 4117731234567891<br>Expiration: not validated<br>CVV2: other than 999 | Visa card with DCC and EMV3DS 2 Challenge |
| 5409960031405146<br>Expiration: not validated<br>CVV2: other than 999 | Master card with Norwegian Korona currency and EMV3DS 2 Challenge |

| | |
|---|---|
| 5171471234567894<br>Expiration: not validated<br>CVV2: other than 999 | Master card with Pound currency and EMV3DS 2 Challenge |
| 4001151234567891<br>Expiration: not validated<br>CVV2: other than 999 | Visa card with Pound currency and EMV3DS 2 Challenge |
| **Other card brands** | |
| 36849800000018<br>Expiration: not validated<br>CVV2: other than 999 | Diners Club card |
| 36849800000000<br>Expiration: not validated<br>CVV2: other than 999 | Diners Club card |
| 376674000000008<br>Expiration: not validated<br>CVV2: other than 999 | Amex card |
| 376674000000016<br>Expiration: not validated<br>CVV2: other than 999 | Amex card |
| 3587870000000001<br>Expiration: not validated<br>CVV2: other than 999 | JCB card |
| 3587870000000019<br>Expiration: not validated<br>CVV2: other than 999 | JCB card |

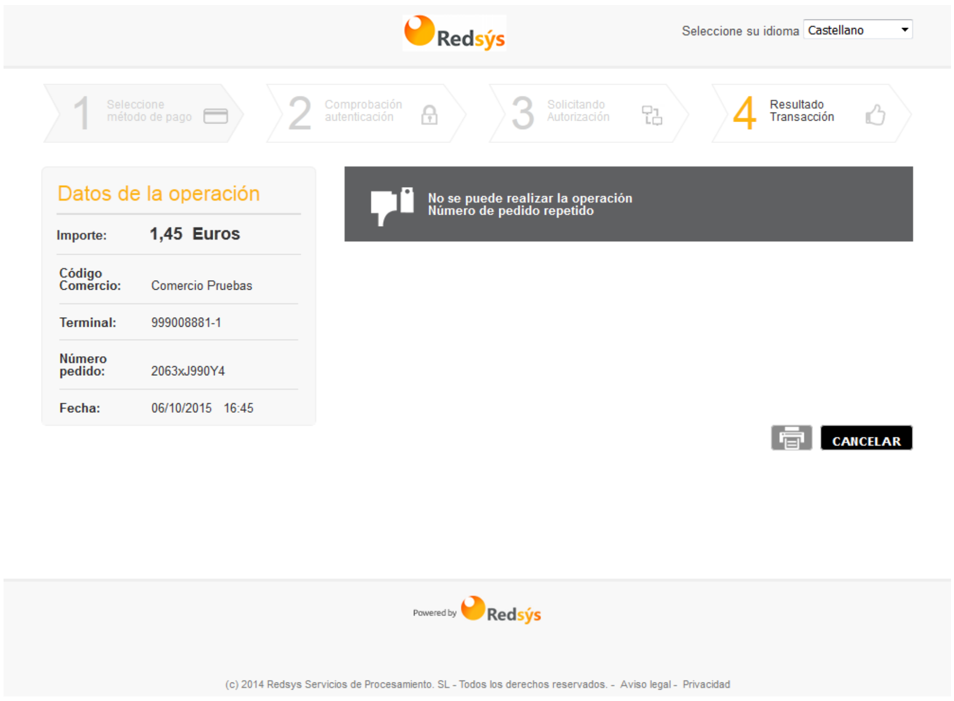
# 11. Error codes

This section shows how to report possible errors that may occur in the integration process.

*NOTE: The complete list of all SIS error codes is available in the document "TPV-Virtual Parámetros Entrada-Salida.xlsx".*

The error that has occurred can be obtained by consulting the source code of the result page of the operation, as shown below:

**Operation Result Page:**


Redsýs
Seleccione su idioma Castellano
1 Seleccione método de pago
2 Comprobación autenticación
3 Solicitando Autorización
4 Resultado Transacción
CANCELAR
Powered by Redsýs

Datos de la operación

| | |
|---|---|
| Importe: | 1,45 Euros |
| Código Comercio: | Comercio Pruebas |
| Terminal: | 999008881-1 |
| Número pedido: | 2063xJ990Y4 |
| Fecha: | 06/10/2015 16:45 |

No se puede realizar la operación
Número de pedido repetido

(c) 2014 Redsys Servicios de Procesamiento. SL - Todos los derechos reservados. - Aviso legal - Privacidad

**Operation result page (source code)**

```
<div class="result-code error">
<p>
<text lngid="noSePuedeRealizarOperacion">No se puede
<br>Número de pedido repetido<br>
<!--SIS0051:-->
</p>
</div>
</div>
<div class="result-info">
<table class="tableresults">
<tbody></tbody>
```

# 12. Frequent errors

**I do not receive the "online" response of purchases with the Virtual POS**

The Virtual POS will submit the notification to the address that the merchant has sent in the Ds_Merchant_MerchantURL field of the payment request.

The merchant can also check the result of the notification in the Virtual POS Management portal, in the "Notifications" menu option, or in the details of the transaction on the operations query screen of this Management portal.

Since the response of the servers follows an international protocol, the specific meaning of the generic errors may be consulted on many websites.

For example:

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

If the merchant is not able to install a server allowing it to receive HTTP notifications and it wants to request direct and personalized consulting, it should contact its Bank.

**I do not receive the confirmation email of purchases with the Virtual POS.**

It happens frequently that the notifications sent by the Virtual POS arrive to the merchant's email account, but they enter the spam folder. This folder should be checked.

**I do not receive the parameters with the result of the purchase operations in my OK and KO URLs.**

The OK and KO URLs should never be used to receive the parameters with the result of the operations. These are URLs that should only be used to redirect from the cardholder to the merchant's website for commercial purposes or for the merchant to show the receipt to the cardholder, if configured for it.

**Signature error (error SIS0042)**

When there is a signature error, the merchant must verify:

- That the data that have been used to make the signature are the same as those sent in the form, taking into account that any modification of the value or format of a subsequent field to the signature calculation makes it incorrect.
- That the secret key used by the merchant matches with the key that the merchant has loaded in the Administration module (merchant section).
- It should be verified that blank spaces are not being sent in the signature. If the request is made through cURL or through the Safari browser, the "+" symbols might become blank

spaces. To avoid this, the "+" symbols of the signature must be replaced by "% 2B" (URL encoded value).

- If the merchant fails to locate the wrong parameter, it should contact the Redsys Client Service Centre, or the Redsys Integration Support department (if its Bank has provided the contact) or contact its Bank support service.

**I have in my merchant refusals by repeated number (SIS0051), though I am not aware of having repeated them.**

This usually occurs because the merchant platform is generating repeated order numbers only when it receives denials or authorizations, but it is repeating them when the transactions are incomplete. Given this, there are two options:

- Request the support service to configure the POS so that you can repeat order numbers. Maximum of one authorized operation per day and no limit for those denied.
- Always generate different order numbers, not only for authorized and denied operations, but for those that have not finished after some time.

**I need to return an operation, but the return option does not appear in the management module.**

It is because the user used to access the Management Portal does not have permission to make returns. To activate this permission to the user, you must request it to your bank.

# 13. Frequently Asked Questions

**I am a merchant and I need to know the encryption key of my Virtual POS**

To see the Virtual POS key, follow these steps:

Access your virtual POS management module.

Select the "merchant" option and press "view key"

Enter your Virtual POS user password and press Accept. -You will have access to see the merchant key for 10 seconds.

**My access merchant user to the Management Portal is blocked. How can I unlock it?**

Under the username and password boxes there is a "I forgot my password" link. After pressing it, you must enter your username and confirm the email address for the new password.

**I want to modify the OK and KO URLs of my merchant.**

The OK and KO URLs are Internet addresses defined by the merchant, to which it is possible to redirect the cardholder once the purchase receipt is displayed. There are two ways to define these URLs:

- If they are configured in the Management portal.
- If the merchant submits them on the payment form in the Ds_Merchant_UrlOK and Ds_Merchant_UrlKO fields, it will be the merchant itself that must modify them on the payment form being submitted.

# ANNEX 1

## 1.1 Request parameters

In the payment request to the SIS Virtual POS, a series of mandatory and other optional data will have to be sent, which will be based on the type of operation and operative to perform.

*NOTE: The complete list of all SIS parameters is available in the document "TPV-Virtual Parámetros Entrada-Salida.xlsx".*

### 1.1.1 Payment / preauthorization example

The following is an example of the Ds_Merchant_Parameters parameter, prior to being encoded in Base 64:

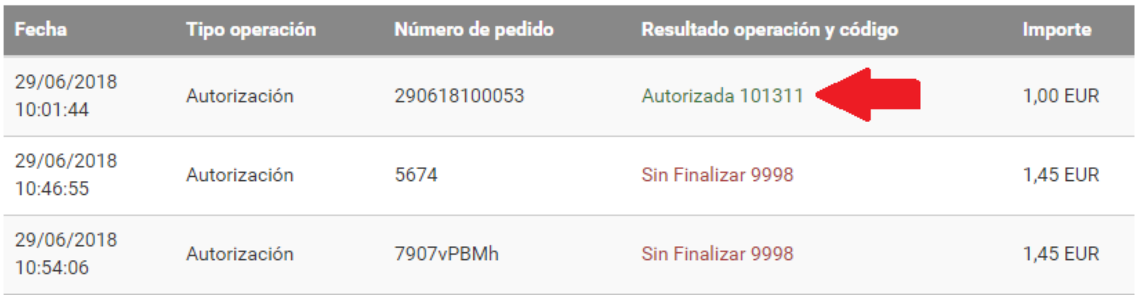
```
{"DS_MERCHANT_ORDER":"1552565870",
"DS_MERCHANT_MERCHANTCODE":"999008881",
"DS_MERCHANT_TERMINAL":"999",
"DS_MERCHANT_CURRENCY":"978",
"DS_MERCHANT_TRANSACTIONTYPE":"0",
"DS_MERCHANT_AMOUNT":"1000",
"DS_MERCHANT_MERCHANTURL":"http://www.prueba.com/urlNotificacion.php",
"DS_MERCHANT_URLOK":"http://www.prueba.com/urlOK.php",
"DS_MERCHANT_URLKO":"http://www.bancsabadell.com/urlKO.php"}
```

* DS_MERCHANT_TRANSACTIONTYPE":"0" for PAYMENT
* DS_MERCHANT_TRANSACTIONTYPE":"1" for PRE-AUTHORIZATION

## 1.2 Online notification parameters

The online notification is a parallel communication and independent of the client's browsing process through the Virtual POS, through which a POST with the data resulted are sent to the merchant.

The result of the operation will be reported by means of the Ds_Response or "Response code" parameter. In addition, said response code will be reported in the operations query, if the operation is not authorized, as shown in the following picture:



| Fecha | Tipo operación | Número de pedido | Resultado operación y código | Importe |
|---|---|---|---|---|
| 29/06/2018 10:01:44 | Autorización | 290618100053 | Autorizada 101311 | 1,00 EUR |
| 29/06/2018 10:46:55 | Autorización | 5674 | Sin Finalizar 9998 | 1,45 EUR |
| 29/06/2018 10:54:06 | Autorización | 7907vPBMh | Sin Finalizar 9998 | 1,45 EUR |

Obviously, from the merchant server, there must be a process that collects this response and performs the necessary tasks for the order management. To do this, you will have to provide, as a parameter, a URL to receive these responses on the web form sent when making the authorization request (see the field Ds_Merchant_MerchantURL in “Payment form data”). This URL will be a CGI, Servlet, etc. developed in the language that the merchant considers appropriate to integrate into its Server (C, Java, Perl, PHP, ASP, etc.), able of interpreting the response sent by the Virtual POS.

*NOTE: The notified data will also be incorporated in the URL OK (Ds_Merchant_UrlOK) or URL KO (Ds_Merchant_UrlKO) if the merchant has activated the submitting of parameters in the response redirection.*

*NOTE 2: The complete list of all SIS input parameters is available in the document “TPV-Virtual Parámetros Entrada-Salida.xlsx”.*

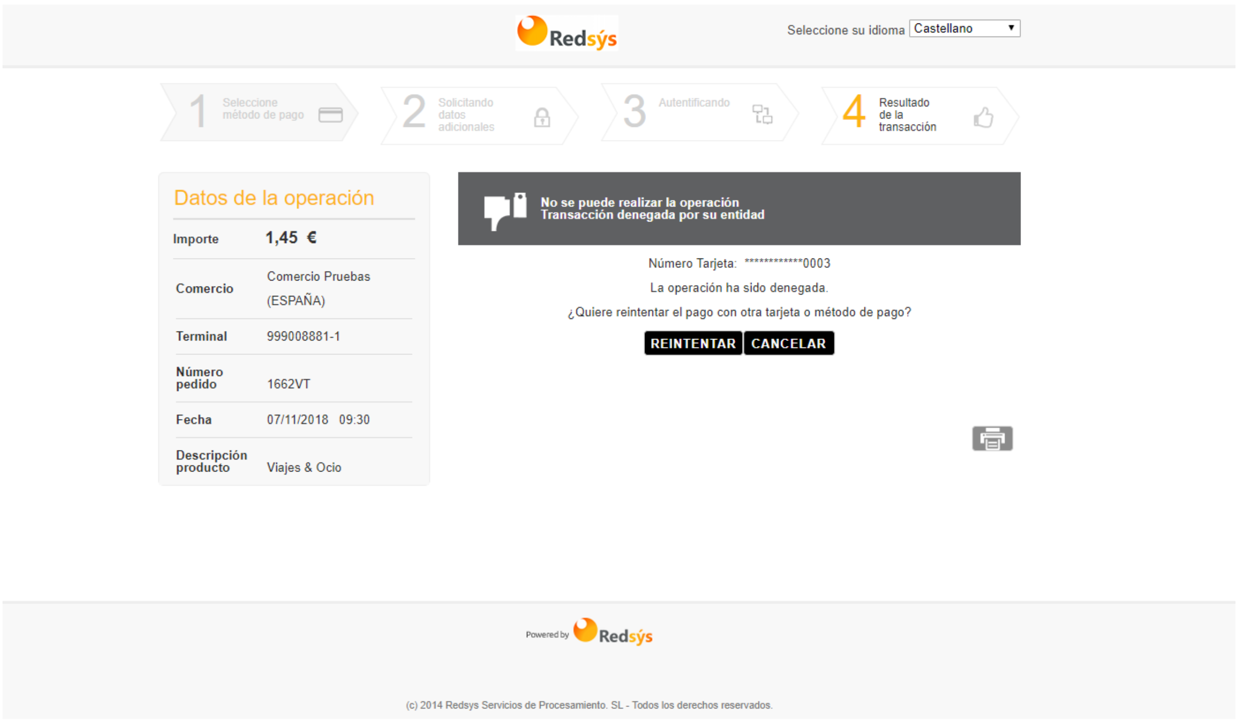
## 1.3 Payment retries

The option of retrying the payment offers the cardholder the possibility of trying to make the payment, with another card or with another payment method, when the operation has been denied by the issuer or by an error in the cardholder authentication (Error = 184). The payment retry option will not be offered to the cardholder if the operation is denied by the application of fraud rules, by a technical error in the authentication process or by any other type of error.

For the client to see this option, the merchant must have a specific configuration in the Virtual-POS Management Module (Allow Repeat Order = YES, with retries) and meet the following requirements:

- The merchant must be able to receive several notifications associated with the same order number and only take into account the first notification that identifies that the operation is authorized, or in case of not receiving any notification of authorized operation, it must take into account the last notification received about the specific order number.
- Currently, the retry option is not compatible with the payment modules provided by Redsys for the virtual shops of Prestashop, Magento, etc. In payment modules external to Redsys this option may not work properly, causing an error in the update of orders in the virtual shop.

The following picture shows an example of the receipt shown to the client, in which he/she is offered the option of retrying:


Seleccione su idioma
Castellano
1 Seleccione método de pago
2 Solicitando datos adicionales
3 Autentificando
4 Resultado de la transacción
REINTENTAR
CANCELAR
Powered by Redsýs

Datos de la operación

| | |
|---|---|
| Importe | 1,45 € |
| Comercio | Comercio Pruebas (ESPAÑA) |
| Terminal | 999008881-1 |
| Número pedido | 1662VT |
| Fecha | 07/11/2018 09:30 |
| Descripción producto | Viajes & Ocio |

No se puede realizar la operación
Transacción denegada por su entidad

Número Tarjeta: ************0003

La operación ha sido denegada.

¿Quiere reintentar el pago con otra tarjeta o método de pago?

(c) 2014 Redsys Servicios de Procesamiento. SL - Todos los derechos reservados.

# ANNEX 2

## 2.1 Help libraries. Request Submission Form

In the previous sections the way of accessing the SIS using Redirection connection and the signature system based on HMAC SHA256 has been described. This section explains how to use the libraries available in PHP, JAVA and.NET to allow the development and generation of payment form fields. The use of the libraries provided by Redsys is optional, although they simplify the developments to be made by the merchant.

### 2.1.1 PHP Library

Below are the steps that a merchant must follow to use the PHP library provided by Redsys:

1. Import the main library file, as shown below:

```
include_once 'redsysHMAC256_API_PHP_4.0.2/apiRedsys.php';
```

   The merchant must decide if the import wishes to do it with the "include" or "required" function, depending on the developments made.

2. Define an object of the main class of the library, as shown below:

```
$miObj = new RedsysAPI;
```

3. Calculate the Ds_MerchantParameters parameter. To carry out the calculation of this parameter, initially all the parameters of the payment request to be sent must be added, as shown below:

```
$miObj->setParameter("DS_MERCHANT_AMOUNT",$amount);
$miObj->setParameter("DS_MERCHANT_ORDER",$id);
$miObj->setParameter("DS_MERCHANT_MERCHANTCODE",$fuc);
$miObj->setParameter("DS_MERCHANT_CURRENCY",$moneda);
$miObj->setParameter("DS_MERCHANT_TRANSACTIONTYPE",$trans);
$miObj->setParameter("DS_MERCHANT_TERMINAL",$terminal);
$miObj->setParameter("DS_MERCHANT_MERCHANTURL",$url);
$miObj->setParameter("DS_MERCHANT_URLOK",$urlOK);
$miObj->setParameter("DS_MERCHANT_URLKO",$urlKO);
```

   Finally, to calculate the Ds_MerchantParameters parameter, the library function "createMerchantParameters ()" must be named, as shown below:

```
$params = $miObj->createMerchantParameters();
```

4. Calculate the Ds_Signature parameter. To carry out the calculation of this parameter, the library function "createMerchantSignature ()" must be named with the key obtained from the management module, as shown below:

```
$claveModuloAdmin = 'Mk9m98IfEblmPfrpsawt7BmxObt98Jev';
$signature = $miObj->createMerchantSignature($claveModuloAdmin);
```

5. Once the values of the Ds_MerchantParameters and Ds_Signature parameters have been obtained, the payment form must be filled in with these values, as shown below:

```
<form action="https://sis.redsys.es/sis/realizarPago"
    method="POST" target="_blank">

    <input type="text" name="Ds_SignatureVersion"
        value="HMAC_SHA256_V1"/>
    <input type="text" name="Ds_MerchantParameters"
        value="<?php echo $params; ?>"/>
    <input type="text" name="Ds_Signature"
        value="<?php echo $signature; ?>"/>
    <input type="submit" value="Realizar Pago"/>

</form>
```

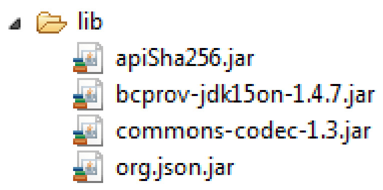
### 2.1.2 JAVA Library

Below, the steps that a merchant must follow to use the JAVA library provided by Redsys are described:

1. Import the library, as shown below:

```
<%@page import="sis.redsys.api.ApiMacSha256"%>
```

The merchant must include in the project construction route all the libraries (JARs) that are provided:



```
lib
    apiSha256.jar
    bcprov-jdk15on-1.4.7.jar
    commons-codec-1.3.jar
    org.json.jar
```

2. Define an object of the main class of the library, as shown below:

```
ApiMacSha256 apiMacSha256 = new ApiMacSha256();
```

3. Calculate the Ds_MerchantParameters parameter. To carry out the calculation of this parameter, initially all the parameters of the payment request to be sent must be added, as shown below:

```
apiMacSha256.setParameter("DS_MERCHANT_AMOUNT", amount);
apiMacSha256.setParameter("DS_MERCHANT_ORDER", id);
apiMacSha256.setParameter("DS_MERCHANT_MERCHANTCODE", fuc);
apiMacSha256.setParameter("DS_MERCHANT_CURRENCY", moneda);
apiMacSha256.setParameter("DS_MERCHANT_TRANSACTIONTYPE", trans);
apiMacSha256.setParameter("DS_MERCHANT_TERMINAL", terminal);
apiMacSha256.setParameter("DS_MERCHANT_MERCHANTURL", url);
apiMacSha256.setParameter("DS_MERCHANT_URLOK", urlOK);
apiMacSha256.setParameter("DS_MERCHANT_URLKO", urlKO);
```

Finally, the library function "createMerchantParameters ()" must be named, as shown below:

```
String params = apiMacSha256.createMerchantParameters();
```

4. Calculate the Ds_Signature parameter. To carry out the calculation of this parameter, the library function "createMerchantSignature ()" must be named with the key obtained from the management module, as shown below:

```
String claveModuloAdmin = "Mk9m98IfEblmPfrpsawt7BmxObt98Jev";
String signature = apiMacSha256.createMerchantSignature(claveModuloAdmin);
```

5. Once the values of the Ds_MerchantParameters and Ds_Signature parameters have been obtained, the payment form must be filled in with these values, as shown below:

```
<form action="https://sis.redsys.es/sis/realizarPago"
    method="POST" target="_blank">

    <input type="text" name="Ds_SignatureVersion"
        value="HMAC_SHA256_V1" />
    <input type="text" name="Ds_MerchantParameters"
        value="<%= params %>" />
    <input type="text" name="Ds_Signature"
    value="<%= signature %>" />
    <input type="submit" value="Realizar Pago" />

</form>
```

### 2.1.3 .NET Library

Below, the steps that a merchant must follow to use the library are described. NET provided by Redsys:

1. Import the RedsysAPI and Newronsoft.Json library in the project.
2. Calculate the **Ds_MerchantParameters parameter**. To carry out the calculation of this parameter, initially all the parameters of the payment request to be sent must be added, as shown below:

```
// New instance of RedysAPI
RedsysAPI r = new RedsysAPI();

// Fill Ds_MerchantParameters parameters
r.SetParameter("DS_MERCHANT_AMOUNT", amount);
r.SetParameter("DS_MERCHANT_ORDER", id);
r.SetParameter("DS_MERCHANT_MERCHANTCODE", fuc);
r.SetParameter("DS_MERCHANT_CURRENCY", currency);
r.SetParameter("DS_MERCHANT_TRANSACTIONTYPE", trans);
r.SetParameter("DS_MERCHANT_TERMINAL", terminal);
r.SetParameter("DS_MERCHANT_MERCHANTURL", url);
r.SetParameter("DS_MERCHANT_URLOK", urlOK);
r.SetParameter("DS_MERCHANT_URLKO", urlKO);
```

   Finally, the library function “createMerchantParameters ()” must be named, as shown below:

```
string parms = r.createMerchantParameters();
Ds_MerchantParameters.Value = parms;
```

3. Calculate the **Ds_Signature** parameter. To carry out the calculation of this parameter, the library function “createMerchantSignature ()” must be named with the key obtained from the management module, as shown below:

```
string sig = r.createMerchantSignature(kc);
Ds_Signature.Value = sig;
```

4. Once the values of the **Ds_MerchantParameters and Ds _Signature** parameters have been obtained, the payment form must be filled in with these values, as shown below:

```
<form action="https://sis-d.redsys.es:25443/sis/realizarPago"
method="post">
    <input runat="server" type="text" id="Ds_SignatureVersion"
        name="Ds_SignatureVersion" value="" />
    <input runat="server" size="100" type="text" id="Ds_MerchantParameters"
        name="Ds_MerchantParameters" value="" />
    <input runat="server" type="text" size="50" id="Ds_Signature"
        name="Ds_Signature" value="" />
    <input runat="server" type="submit" value="Realizar Pago" />
</form>
```

## 2.2 Help libraries. On-line reception

### 2.2.1 PHP Library

Below are the steps that a merchant must follow to use the PHP library provided by Redsys:

1. Import the main library file, as shown below:

```
include_once 'redsysHMAC256_API_PHP_4.0.2/apiRedsys.php';
```

The merchant must decide if the import wishes to do it with the “include” or “required” function, depending on the developments made.

2. Define an object of the main class of the library, as shown below:

```
$miObj = new RedsysAPI;
```

3. Capture the parameters of the online notification:

```
$version = $_POST["Ds_SignatureVersion"];
$params = $_POST["Ds_MerchantParameters"];
$signatureRecibida = $_POST["Ds_Signature"];
```

4. Decode the **Ds_MerchantParameters parameter**. To perform the decoding of this parameter, the library function “decodeMerchantParameters ()” must be named, as shown below:

```
$decodec = $miObj->decodeMerchantParameters($params);
```

Once the “decodeMerchantParameters ()” function call has been executed, the value of any parameter that is likely to be included in the online notification (Annex 1.2) can be obtained. To achieve the value of a parameter, the “getParameter ()” function of the library must be named with the parameter name to obtain the response code, as shown below:

```
$codigoRespuesta = $miObj->getParameter("Ds_Response");
```

*IMPORTANT NOTE: To guarantee the security and the origin of the notifications, the merchant must carry out the validation of the received signature and of all the parameters sent in the notification.*

5. Validate the **Ds_Signature** parameter. To carry out the validation of this parameter, the signature must be calculated and compared with the captured **Ds_Signature** parameter. To do this, the library function "createMerchantSignatureNotif ()" must be named with the key obtained from the management module and the **captured Ds_ MerchantParameters parameter,** as shown below:

```
$claveModuloAdmin = 'Mk9m98IfEblmPfrpsawt7BmxObt98Jev';
$signatureCalculada = $miObj->createMerchantSignatureNotif($claveModuloAdmin,
                                                           $params);
```

Once this is done, it can already be validated if the value of the sent signature matches with the value of the calculated signature, as shown below:

```
if ($signatureCalculada === $signatureRecibida){
    echo "FIRMA OK. Realizar tareas en el servidor";
} else {
    echo "FIRMA KO. Error, firma inválida";
}
```

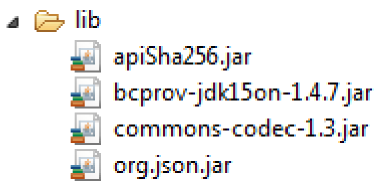
### 2.2.2 JAVA Library

Below, the steps that a merchant must follow to use the JAVA library provided by Redsys are described:

1. Import the library, as shown below:

```
<%@page import="sis.redsys.api.ApiMacSha256"%>
```

The merchant must include in the project construction route all the libraries (JARs) that are provided:



```
lib
    apiSha256.jar
    bcprov-jdk15on-1.4.7.jar
    commons-codec-1.3.jar
    org.json.jar
```

2. Define an object of the main class of the library, as shown below:

```
ApiMacSha256 apiMacSha256 = new ApiMacSha256();
```

3. Capture the parameters of the online notification:

```
String version = request.getParameter("Ds_SignatureVersion");
String params = request.getParameter("Ds_MerchantParameters");
String signatureRecibida = request.getParameter("Ds_Signature");
```

4. Decode the **Ds_MerchantParameters parameter**. To perform the decoding of this parameter, the library function "decodeMerchantParameters ()" must be named, as shown below:

```
String decodec = apiMacSha256.decodeMerchantParameters(params);
```

Once the "decodeMerchantParameters ()" function call has been executed, the value of any parameter that is likely to be included in the online notification (Annex 1.2) can be obtained. To achieve the value of a parameter, the "getParameter ()" function of the library must be named with the parameter name to obtain the response code, as shown below:

```
String codigoRespuesta = apiMacSha256.getParameter("Ds_Response");
```

*IMPORTANT NOTE: To guarantee the security and the origin of the notifications, the merchant must carry out the validation of the received signature and of all the parameters that are sent in the notification.*

5. Validate the **Ds_Signature** parameter. To carry out the validation of this parameter, the signature must be calculated and compared with the captured **Ds_Signature** parameter. To do this, the library function "createMerchantSignatureNotif ()" must be named with the key obtained from the management module and the **captured Ds_ MerchantParameters parameter,** as shown below:

```
String claveModuloAdmin = "Mk9m98IfEblmPfrpsawt7BmxObt98Jev";
String signatureCalculada = apiMacSha256.createMerchantSignatureNotif(claveModuloAdmin,
                                                                        params);
```

Once this is done, it can already be validated if the value of the sent signature matches with the value of the calculated signature, as shown below:

```
if (signatureCalculada.equals(signatureRecibida)) {
    System.out.println("FIRMA OK. Realizar tareas en el servidor");
} else {
    System.out.println("FIRMA KO. Error, firma inválida");
}
```

### 2.2.3 .NET library

Below, the steps that a merchant must follow to use the library are described.

NET provided by Redsys:

1. Import the RedsysAPI and Newronsoft.Json library in the project.
2. Capture the parameters of the online notification:

```
// New instance of RedsysAPI
RedsysAPI r = new RedsysAPI();

// Obtain Ds_SignatureVersion using post
if (Request.Form["Ds_SignatureVersion"] != null)
{
    version = Request.Form["Ds_SignatureVersion"];
}

// Obtain Ds_MerchantParameters using post
if (Request.Form["Ds_MerchantParameters"] != null)
{
    data = Request.Form["Ds_MerchantParameters"];
}

// Obtan Ds_Signature using post
if (Request.Form["Ds_Signature"] != null)
{
    signatureReceived = Request.Form["Ds_Signature"];
}
```

3. Decode the **Ds_MerchantParameters parameter**. To perform the decoding of this parameter, the library function "decodeMerchantParameters ()" generating the JSON string (type) of the response must be named, as shown below:

```
string deco = r.decodeMerchantParameters(data);
```

*IMPORTANT NOTE: To guarantee the security and the origin of the notifications, the merchant must carry out the validation of the received signature and of all the parameters that are sent in the notification.*

4. Validate the **Ds_Signature** parameter. To carry out the validation of this parameter, the signature must be calculated and compared with the captured **Ds_Signature** parameter. To do this, the library function "createMerchantSignatureNotif ()" must be named with the key obtained from the management module and the **captured Ds_ MerchantParameters parameter,** as shown below:

```
var kc = "Mk9m98IfEblmPfrpsawt7BmxObt98Jev";
string notif = r.createMerchantSignatureNotif(kc, data);
```

Once this is done, it can already be validated if the value of the sent signature matches with the value of the calculated signature, as shown below:

```
string text = "";
if (notif.Equals(signatureReceived) && notif != "")
{
    text = "SIGNATURE OK";
}
else
{
    text = "SIGNATURE KO";
}
```

## 2.3 Help libraries. Browsing control return.

In the previous sections the way of accessing the SIS using connection by redirection has been described. This section explains how to use the libraries available in PHP, JAVA and.NET to allow the developments for the reception of the parameters for the browsing control return parameters. The use of the libraries provided by Redsys is optional, although they simplify the developments to be made by the merchant.

### 2.3.1 PHP Library

Below are the steps that a merchant must follow to use the PHP library provided by Redsys:

1. Import the main library file, as shown below:

```
include_once 'redsysHMAC256_API_PHP_4.0.2/apiRedsys.php';
```

   The merchant must decide if the import wishes to do it with the “include” or “required” function, depending on the developments made.

   Define an object of the main class of the library, as shown below:

```
$miObj = new RedsysAPI;
```

2. Capture the parameters of the online notification:

```
$version = $_GET["Ds_SignatureVersion"];
$params = $_GET["Ds_MerchantParameters"];
$signatureRecibida = $_GET["Ds_Signature"];
```

3. Decode the **Ds_MerchantParameters parameter**. To perform the decoding of this parameter, the library function “decodeMerchantParameters ()” must be named, as shown below:

```
$decodec = $miObj->decodeMerchantParameters($params);
```

4. Once the “decodeMerchantParameters ()” function call has been executed, the value of any parameter that is likely to be included in the online notification (Annex 1.2) can be obtained. To achieve the value of a parameter, the “getParameter ()” function of the library must be named with the parameter name to obtain the response code, as shown below:

```
$codigoRespuesta = $miObj->getParameter("Ds_Response");
```

   *IMPORTANT NOTE: It is important to carry out the validation of all the parameters that are sent in the communication. To update the status of the order online, this communication should NOT be used, but the online notification described in the other sections, since the return of the browsing depends on the actions of the client in his/her browser.*

5. Validate the **Ds_Signature** parameter. To carry out the validation of this parameter, the signature must be calculated and compared with the captured **Ds_Signature** parameter. To do this, the library function “createMerchantSignatureNotif ()” must be named with the key obtained from the management module and the **captured Ds_ MerchantParameters parameter,** as shown below:

```
$claveModuloAdmin = 'Mk9m98IfEblmPfrpsawt7BmxObt98Jev';
$signatureCalculada = $miObj->createMerchantSignatureNotif($claveModuloAdmin,
                                                            $params);
```

6. Once this is done, it can already be validated if the value of the sent signature matches with the value of the calculated signature, as shown below:

```
if ($signatureCalculada === $signatureRecibida){
    echo "FIRMA OK. Realizar tareas en el servidor";
} else {
    echo "FIRMA KO. Error, firma inválida";
}
```

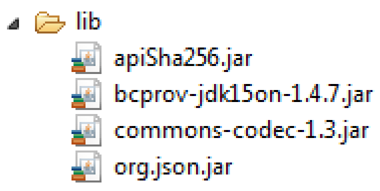
### 2.3.2 JAVA Library

Below, the steps that a merchant must follow to use the JAVA library provided by Redsys are described:

1. Import the library, as shown below:

```
<%@page import="sis.redsys.api.ApiMacSha256"%>
```

The merchant must include in the project construction route all the libraries (JARs) that are provided:



```
◢ lib
    apiSha256.jar
    bcprov-jdk15on-1.4.7.jar
    commons-codec-1.3.jar
    org.json.jar
```

2. Define an object of the main class of the library, as shown below:

```
ApiMacSha256 apiMacSha256 = new ApiMacSha256();
```

3. Capture the browsing control return parameters:

```
String version = request.getParameter("Ds_SignatureVersion");
String params = request.getParameter("Ds_MerchantParameters");
String signatureRecibida = request.getParameter("Ds_Signature");
```

Decode the **Ds_MerchantParameters parameter**. To perform the decoding of this parameter, the library function “decodeMerchantParameters ()” must be named, as shown below:

```
String decodec = apiMacSha256.decodeMerchantParameters(params);
```

Once the call to the “decodeMerchantParameters ()” function has been executed, the value of any parameter that is likely to be included in the browsing control return (Annex 1.2). To obtain the value of a parameter, the “getParameter ()” function of the library must be named with the parameter name to obtain the response code, as shown below:

```
String codigoRespuesta = apiMacSha256.getParameter("Ds_Response");
```

*IMPORTANT NOTE: It is important to carry out the validation of all the parameters that are sent in the communication. To update the status of the order online, this communication should NOT be used, but the online notification described in the other sections, since the return of the browsing depends on the actions of the client in his/her browser.*

4. Validate the **Ds_Signature** parameter. To carry out the validation of this parameter, the signature must be calculated and compared with the captured **Ds_Signature** parameter. To do this, the library function “createMerchantSignatureNotif ()” must be named with the key obtained from the management module and the captured **Ds_** MerchantParameters parameter, as shown below:

```
String claveModuloAdmin = "Mk9m98IfEblmPfrpsawt7BmxObt98Jev";
String signatureCalculada = apiMacSha256.createMerchantSignatureNotif(claveModuloAdmin,
                                                                         params);
```

Once this is done, it can already be validated if the value of the sent signature matches with the value of the calculated signature, as shown below:

```
if (signatureCalculada.equals(signatureRecibida)) {
    System.out.println("FIRMA OK. Realizar tareas en el servidor");
} else {
    System.out.println("FIRMA KO. Error, firma inválida");
}
```

### 2.3.3 .NET library

Below, the steps that a merchant must follow to use the library are described. NET provided by Redsys:

1. Import the library, as shown below:

```
using RedsysAPIPrj;
```

2. Define an object of the main class of the library, as shown below:

```
RedsysAPI r = new RedsysAPI();
```

3. Capture the browsing control return parameters:

```
string version = Request.QueryString["Ds_SignatureVersion"];

string parms = Request.QueryString["Ds_MerchantParameters"];

string signatureRecibida = Request.QueryString["Ds_Signature"];
```

*IMPORTANT NOTE: It is important to carry out the validation of all the parameters that are sent in the communication. To update the status of the order online, this communication should NOT be used, but the online notification described in the other sections, since the return of the browsing depends on the actions of the client in his/her browser.*

4. Validate the **Ds_Signature** parameter. To carry out the validation of this parameter, the signature must be calculated and compared with the captured **Ds_Signature** parameter. To do this, the library function "createMerchantSignatureNotif ()" must be named with the key obtained from the management module and the **captured Ds_ MerchantParameters parameter,** as shown below:

```
var kc = "sq7HjrUOBfKmC576ILgskD5srU870gJ7";

string signatureCalculada =  r.createMerchantSignatureNotif(kc, parms);
```

Once this is done, it can already be validated if the value of the sent signature matches with the value of the calculated signature, as shown below:

```
if (signatureRecibida == signatureCalculada)
{
    result.InnerHtml = "FIRMA OK. Realizar tareas en el servidor";
}
else
{
    result.InnerHtml = "FIRMA KO. Error, firma invalida";
}
```