Santander

Product Guide

Virtual POS

Redsys:
Integration Manual - Redirection

Version 2.5

January 2020

Santander

Interactive index

Santander

Interactive index

Santander

Authorisations and version control

Date: 24/01/2020		
Author: Redsys	VALIDATED BY: E-commerce	APPROVED BY: Redsys
Company: Redsys	Company: Redsys	Company: Redsys
Signature:	Signature:	Signature:

Version	Date	Affects	Short description of the change
1.0	01/06/2018	All	First Version
1.1	24/09/2018	Point 8.1	Code ASCII of the Ds_Merchant_Order
1.2	07/11/2018	Point 8.6	The option to retry the payment was added
1.3	18/12/2018	Point 8.6	The option to retry the payment was changed
1.4	10/01/2019	Point 8.1, 8.2 and 8.6	The Ds_Merchant_Paymethods field was added, a correction was added to the sending of the response code and the merchant requirements in payment retries was explained.
1.5	29/01/2019	Point 8.1	The type of authorisation cancellation was added in the Ds_Merchant_TransactionType parameter
1.6	14/03/2019	Point 8	Points 8.1 and 8.2 have been re-written and the Currencies and Languages points were deleted.
2.0	27/03/2019	Various points	Information added about EMV3DS and PSD2 GuiaErroresSIS.xlsx and TPV-Virtual Parámetros Entrada-Salida.xlsx
2.1	12/04/2019	Error code testing environment	Testing cards for EMV3DS added The spreadsheet of the SIS errors has been changed 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	All the documents	Adaptations for EVM3DS 2.2 have been added
2.4	12/11/2019		The characteristics and specifications that will be available in the future have been marked
2.5	16/12/2019	Points 6, 7, 8, 9 and 10	PSP connection added Change to exemption and tokenisation Clarification and examples for EMV3DS advance functionalities.

1 | Introduction

1.1. Scope

This document covers the technical aspects required for a merchant to perform the integration with the Virtual POS through a connection via redirection of the buyer's browser.

This form of connection makes it possible to transfer the session of the end customer to the Virtual POS, so the payment method selection and the data input is done in the secure environment of the Virtual POS environment and outside the responsibility of the merchant. In addition to the simplicity of implementation for the merchant and the peace of mind insofar as the responsibility of payment information, this connection mode offers the possibility of authenticating the card holder using the 3DS protocol, which allows the holder to be authenticated directly with the bank that issued their card at the time of making the transaction, adding greater security to purchases.

Note: *The connection requires the use of a signature system based on HMAC SHA-256, which authenticates between the merchant's server and the Virtual POS. To develop the calculation of this type of signature, the merchant can do the development itself using the standard functions of the different development environments, although to facilitate the developments, we provide libraries (PHP, JAVA and .NET), the instructions for which are detailed in this guide and which are available at the following address: http://www.redsys.es/wps/portal/redsys/publica/areadeserviciosweb/descargaDeDocumentacionY Ejecutables/ .*

Important Note: *As a result of the entry into force of the European Payments Directive PSD2 during 2020, this guide includes some new features and technical specifications that will be available in the future (in 2020) to facilitate the preparation of the work for merchant that wish to incorporate certain possibilities into their payment transactions, especially in relation to authentication management and authentication exemption covered by the PSD2. If applicable, these headings are marked as "ADVANCE".*

These new functionalities marked as "ADVANCE" are available in the Test environment.

1.2. Definitions, acronyms and abbreviations

- **SIS:** Redsys Integrated Server (Virtual POS Server).
- **3DSecure:** Security system for online payments. Hereinafter, EMV3DS.
- **EMV3DS:** Acronym to identify 3DSecure in the new version of the Virtual POS.

1.3. References

- Integration with the SIS documentation.
- SIS Guide Virtual POS.
- TPV-Virtual GuiaErroresSIS.xlsx.
- Virtual POS Parámetros Entrada-Salida.xlsx.

2 | General description of the flow

The following diagram shows the general flow of an operation performed with the Virtual POS.

1. The card holder selects the products they want to buy from the merchant.

2. The merchant redirects the customer's browser session to Redsys' URL. The customer enters their card information in this URL.

3. The Virtual POS informs the merchant of the result of the transaction and Redsys returns the customer's browser session to the merchant so as they can continue browsing the online store.

2.1. Sending request to the Virtual POS

As shown in step 2 of the previous diagram, the merchant must send the payment request information to the Virtual POS coded in UTF-8 through the card holder's browser. To do so, it must prepare a form with the following fields:

› **Ds_SignatureVersion:** Constant that indicates the version of the signature being used.

› **Ds_MerchantParameters:** String in JSON format with all the parameters of the request coded in Base 64 and without returns to the shopping cart (Annex 14.1 includes the list of parameters that can be sent in a payment request).

› **Ds_Signature:** Signature of the sent information. This is the result of the HMAC SHA256 of the JSON string coded in Base 64 sent in the previous parameter.

URL connection	Environment
https://sis-t.redsys.es:25443/sis/realizarPago	Testing
https://sis.redsys.es/sis/realizarPago	Real

2.2. Receiving the result (online notification)

Once the transaction has been managed, the Virtual POS may inform the merchant's server of the result by means of a direct connection to the merchant's server (step 3 of the flow). This notification is optional and must be configured for each terminal in the Administration Module. The online notification can consist of:

An HTTP POST (synchronous and asynchronous notification) with the information of the result coded in UTF-8. The following fields will be included in the POST:

› **Ds_SignatureVersion:** Constant that indicates the version of the signature being used.

› **Ds_MerchantParameters:** String in JSON format with all the parameters of the response coded in Base 64 and without shopping cart returns (annex 14.2 includes the list of parameters that can be included in the online notification).

› **Ds_Signature:** Signature of the sent information. Result of the HMAC SHA256 of the JSON string coded 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 needed to guarantee that the information has not been manipulated and that it actually originates from the Virtual POS.

Note: *The Virtual POS sends the online notification to the URL reported by the merchant in the Ds_Merchant_MerchantURL parameter.*

2.3. Return of the control of the card holder's browsing

In step 3 of the Virtual POS flow, the customer will be shown a receipt with the result of the transaction, provided that the configuration of the merchant specifies as such. Furthermore, if the transaction is rejected, the customer will have the option to try again with another card or another payment method, provided that the configuration of the merchant allows (the option to try and make the payment again is described in detail in Annex 14.3).

Lastly, in this step the merchant is returned the control of the card holder's browsing. As such, the merchant can see the payment flow, maintaining a natural browsing sequence for the customer/buyer.

Optionally, the Virtual POS can include the same fields as the online notification.

3 | Form for sending a request

The merchant must create a form with the parameters of the payment request that must be sent to the Virtual POS through the customer's browser. Below are different examples of the payment request form:

› Example of the payment form **without sending card information:**

```
<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="
        eyJEU19NRVJDSEFOVF9BTU9VTlQiOiI5OTkiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjEyMzQ1Njc4OTAiLCJEU19NRVJD
        SEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX01FU
        kNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDSEFOVF9
        NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiwiRFNfTUV
        SQ0hBTIRfVVJMT0siOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsT0sucGhwIiwiRFNfTUVSQ0hBTIRfVVJMS
        08iOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsS08ucGhwIn0="/>
        <input type="hidden" name="Ds_Signature" value="PqV2+SF6asdasMjXasKJRTh3UIYya1hmU/igHkzhC+R="/>
</form>.
```

› Example of the payment form **sending card information:**

```
<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="
        eyJEU19NRVJDSEFOVF9BTU9VTlQiOiI5OTkiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjEyMzQ1Njc4OTAiLCJEU19NRVJD
        SEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX01FU
        kNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDSEFOVF9
        NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiwiRFNfTUV
        SQ0hBTlRfVVJMT0siOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsT0sucGhwIiwiRFNfTUVSQ0hBTlRfVVJMS
        08iOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsS08ucGhwIn0="/>
        <input type="hidden" name="Ds_Signature" value="PqV2+SF6asdasMjXasKJRTh3UIYya1hmU/igHkzhC+R="/>
</form>
```

To facilitate the merchant integration, below is a detailed explanation of the steps to follow to create the payment request form.

3.1. Identifying the version of the signature algorithm to use

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

3.2. Assembling the request information chain

A string must be assembled with all the information from the request in JSON format. The name of each parameter must be indicated in capital letters or with the "CamelCase" structure (For example: DS_MERCHANT_AMOUNT or Ds_Merchant_Amount).

Merchants that use special operations such as the "Reference payment" (1-click-payment) must include the specific parameters of their operation as part of the JSON object.

The list of parameters that can be included in the request are described in annex 14.1. Below are some examples of the JSON object of a request:

› Example **without sending card information:**

```
{"DS_MERCHANT_AMOUNT":"145","DS_MERCHANT_ORDER":"1446117555","DS_MERCHANT_MERCHANTC
ODE":"999008881","DS_MERCHANT_CURRENCY":"978","DS_MERCHANT_TRANSACTIONTYPE":"0","DS_MERCHA
NT_TERMINAL":"1","DS_MERCHANT_MERCHANTURL":"http:\/\/www.prueba.com\/urlNotificacion.php","DS_MERC
HANT_URLOK":"http:\/\/www.prueba.com\/urlOK.php","DS_MERCHANT_URLKO":"http:\/\/www.bancsabadell.co
m\/urlKO.php"}
```

› Example **sending card information:**

*{"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_URLOK":"http:\/\/www.prueba.com\/urlOK.php","DS_MERCHANT_URLKO":"http:\/\/www.prueba.com\/urlKO.php","DS_MERCHANT_PAN":"454881*******04","DS_MERCHANT_EXPIRYDATE":"1512","DS_MERCHANT_CVV2":"123"}*

Once the JSON string has been assembled with all the fields, it must be coded in BASE64 without shopping cart returns to ensure that it remains constant and is not changed as it moves through the customer/buyer browser.

Below are the JSON objects that were just shown **coded in BASE64:**

› Example of code JSON **without sending card information:**

eyJEU19NRVJDSEFOVF9BTU9VTlQiOiI5OTkiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjEyMzQ1Njc4OTAiLCJEU19NRVJDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiwiRFNfTUVSQ0hBTlRfVVJMT0siOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsT0sucGhwIiwiRFNfTUVSQ0hBTlRfVVJMS08iOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsS08ucGhwIn0

› Example of code JSON **sending card information:**

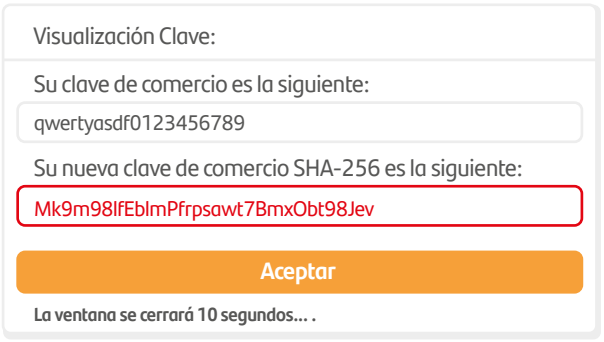
eyJEU19NRVJDSEFOVF9BTU9VTlQiOiIxNDUiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjE0NDYwNjg1ODEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiwiRFNfTUVSQ0hBTlRfVVJMT0siOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsT0sucGhwIiwiRFNfTUVSQ0hBTlRfVVJMS08iOiJodHRwOlwvXC93d3cucHJ1ZWJhLmNvbVwvdXJsS08ucGhwIiwiRFNfTUVSQ0hBTlRfUEFOIjoiNDU0ODgxMjA0OTQwMDAwNCIsIkRTX01FUkNIQU5UX0VYUElSWURBVEUiOiIxNTEyIiwiRFNfTUVSQ0hBTlRfQ1ZWMiI6IjEyMyJ9

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

Note: *The use of the help libraries provided by Redsys to generate this field are shown in section 3.5.*

3.3. Identifying the code to use for the signature

In order to calculate the signature, a specific code must be used for each terminal. The code can be obtained by accessing the administration module, "See Merchant Information" option, in the "Ver clave" ["See code"] section, as shown in the following image:



Important Note: *This code must be stored in the merchant's server as securely as possible to prevent any fraudulent use. The merchant is responsible for maintaining this code secret and its correct safekeeping.*

3.4. Signing the request information

Once the chain of information to be signed and the specific terminal code has been assembled, the signature must be calculated following the steps below:

1. A specific code will be generated for each transaction. To obtain the derived code to be used in a transaction, a 3DES coding must be carried out between the merchant's code, which must be previously decoded in BASE 64, and the value of the transaction order number (Ds_Merchant_Order).

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

3. The result obtained is coded 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 of the form shown at the beginning of section 3.

Note: *The use of the help libraries provided by Redsys to generate this field are shown in section 3.5.*

3.5. Using help libraries

The previous sections described how to access the SIS using a connection by redirection and the signature system based on HMAC SHA256. This section explains how to use the libraries available in PHP, JAVA and .NET to facilitate developments and the generation of the payment form fields. Using libraries supplied by Redsys is optional, although they do simplify the developments to be carried out by the merchant.

3.5.1 PHP L ibrary

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

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

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

The merchant must decide whether to import with the "include" or "required" function, according to the developments carried out.

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

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

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

Lastly, to calculate the Ds_MerchantParameters parameter, the

```
$miObj->setParameter("DS_MERCHANT_AMOUNT",$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);
```

"createMerchantParameters()" library function must be called, as shown below:

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

4. Calculate the Ds_Signature parameter. To calculate this parameter, the "createMerchantSignature()" library function must be called with the code obtained from the administration module, as shown below:

```
$claveModuloAdmin = `Mk9m98IffEblmPfrpsawt7BmxObt98Jev´;
$signature = $miObj->createMerchantParameters($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>
```

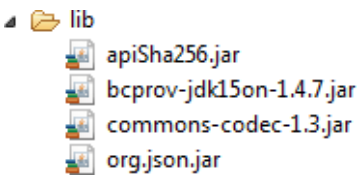
3.5.2 JAVA Library

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

1. Import the library, as shown below:

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

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



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

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

3. Calculate the Ds_MerchantParameters parameter. To calculate 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);
```

Lastly, the "createMerchantParameters()" library function must be called, as shown below:

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

4. Calculate the Ds_Signature parameter. To calculate this parameter, the "createMerchantSignature()" library function must be called with the code obtained from the administration 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>
```

3.5.3 .NET Library

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

1. Import the RedsysAPI library and Newronsoft.Json library to the project.

2. Calculate the **Ds_MerchantParameters** parameter. To calculate this parameter, initially all the parameters of the payment request to be sent must be added, as shown below:

```
// New instance of RedsysAPI
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);
```

Lastly, the "createMerchantParameters()" library function must be called, as shown below:

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

3. Calculate the **Ds_Signature** parameter. To calculate this parameter, the "createMerchantSignature()" library function must be called with the code obtained from the administration 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_SigntureVersion" 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>
```

4 Receiving the online notification

The online notification is an optional function that allows the online store to receive the result of an online transaction in real time, once the customer has completed to the process in the Virtual POS.

The merchant must **capture and validate all the parameters together** with the signature of the online notification before any execution on its server.

The Virtual POS has different types of notifications, as follows:

1. **Synchronous.** Implies that the result of the first purchase is sent to the merchant and then to the customer with the value. The transaction cannot be cancelled, even if the notification is incorrect.
2. **Asynchronous.** Implies that the result of the authorization is communicated to the merchant and the customer at the same time.

The following sub-sections explain how to use the help libraries provided by Redsys, which will depend on the type of notification configured:

4.1. Synchronous and Asynchronous notification

The previous sections described how to access the SIS using a connection by redirection and the signature system based on HMAC SHA256. This section described how to use the available PHP, .NET and JAVA libraries to facilitate developments for the receipt of the online notification and signature validation parameters. Using libraries supplied by Redsys is optional, although they do simplify the developments to be carried out by the merchant.

4.1.1 PHP Library

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

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

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

The merchant must decide whether to import with the "include" or "required" function, according to the developments carried out.

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

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

3. Capture the online notification parameters:

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

4. Decode the **Ds_MerchantParameters** parameter. To decode this parameter, the “decodeMerchantParameters()” library function must be called, as shown below:

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

Once the call has been made to the “decodeMerchantParameters()” function, the value of any parameter susceptible to being included in the online notification can be obtained (Annex 14.2). To obtain the value of a parameter, the "getParameter()" library function must be called with the name of the parameter, as shown below to obtain the response code:

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

Important Note: *To guarantee the security and the origin of the notifications, the merchant must validate the signature received and all the parameters that are sent in the notification.*

5. Validate the **Ds_Signature** parameter. To validate this parameter, the signature must be calculated and compared with the **Ds_Signature** parameter captured. To do so, the “createMerchantSignatureNotif()” library function must be called with the code obtained from the administration module, and the **Ds_MerchantParameters** parameter captured, as shown below:

```
$claveModuloAdmin = `Mk9m8BIffEblmPffrpsawt7BmxObt98Jev´;
$signatureCalculada = $miObj->createMerchantSignatureNotif($claveModuloAdmin,$params);
```

Once done, it is possible to check whether the value of the signature sent matches the value of the signature calculated, as shown below:

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

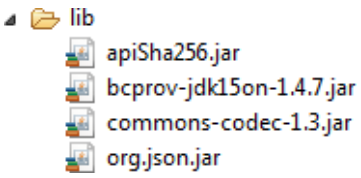
4.1.2 JAVA Library

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

1. Import the library, as shown below:

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

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



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

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

3. Capture the online notification parameters:

```
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 decode this parameter, the "decodeMerchantParameters()" library function must be called, as shown below:

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

Once the call has been made to the "decodeMerchantParameters()" function, the value of any parameter susceptible to being included in the online notification can be obtained (Annex 14.2). To obtain the value of a parameter, the "getParameter()" library function must be called with the name of the parameter, as shown below to obtain the response code:

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

Important Note: *To guarantee the security and the origin of the notifications, the merchant must validate the signature received and all the parameters that are sent in the notification.*

5. Validate the **Ds_Signature** parameter. To validate this parameter, the signature must be calculated and compared with the **Ds_Signature** parameter captured. To do so, the "createMerchantSignatureNotif()" library function must be called with the code obtained from the administration module, and the **Ds_MerchantParameters** parameter captured, as shown below:

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

Once done, it is possible to check whether the value of the signature sent matches the value of the signature calculated, 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");
}
```

4.1.3 .NET Library

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

1. Import the RedsysAPI library and Newronsoft.Json library to the project.

2. Capture the online notification parameters:

```
// 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"];
}

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

3. Calculate the Decode **Ds_MerchantParameters** parameter. To decode this parameter, the "decodeMerchantParameters()" library function must be called, which creates the JSON string of the response, as shown below:

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

Important Note: *To guarantee the security and the origin of the notifications, the merchant must validate the signature received and all the parameters that are sent in the notification.*

4. Validate the **Ds_Signature** parameter. To validate this parameter, the signature must be calculated and compared with the **Ds_Signature** parameter captured. To do so, the "createMerchantSignatureNotif()" library function must be called with the code obtained from the administration module, and the **Ds_MerchantParameters** parameter captured, as shown below:

```
var kc = "Mk9m98IfEblmPfrpsawt7BmxObt98Jev";

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

Once done, it is possible to check whether the value of the signature sent matches the value of the signature calculated, as shown below:

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

5 | Return of the browsing control

Once the customer has completed the Virtual POS process, the browsing is redirected to the online store. This return to the online store is via the URL communicated as a parameter in the initial call to the Virtual POS, or if this is not available, the configuration of the terminal is obtained from the Virtual POS administration module. There can be different return URLs depending on the result of the transactions (URL OK or URL KO).

The merchant must capture and validate, if the merchant's configuration so requires (Parameter in the URLs = SI [YES]) the parameters of the return of the browsing control before any execution on its server.

Below is an explanation of how to use the help libraries provided by Redsys for capturing and validating browsing control return parameters.

5.1. Using help libraries

The previous sections described how to access the SIS using a connection by redirection. This section described how to use the available PHP, .NET and JAVA libraries to facilitate developments for the receipt of the browsing control return parameters. Using libraries supplied by Redsys is optional, although they do simplify the developments to be carried out by the merchant.

5.1.1 PHP Library

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

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

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

The merchant must decide whether to import with the "include" or "required" function, according to the developments carried out. Define an object from the main class of the library, as shown below:

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

2. Capture the online notification parameters:

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

3. Decode the **Ds_MerchantParameters** parameter. To decode this parameter, the "decodeMerchantParameters()" library function must be called, as shown below:

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

4. Once the call has been made to the "decodeMerchantParameters()" function, the value of any parameter susceptible to being included in the online notification can be obtained (Annex 14.2). To obtain the value of a parameter, the "getParameter()" library function must be called with the name of the parameter, as shown below to obtain the response code:

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

Important Note: *It is important to validate all the parameters that are sent in the communication. To update the status of the online order, this communication must NOT be used, rather the online notification described in the other sections, since the return to browsing depends on the customer's actions in their browser.*

5. Validate the **Ds_Signature** parameter. To validate this parameter, the signature must be calculated and compared with the **Ds_Signature** parameter captured. To do so, the "createMerchantSignatureNotif()" library function must be called with the code obtained from the administration module, and the **Ds_MerchantParameters** parameter captured, as shown below:

```
$claveModuloAdmin = `MK9m98IfEblmPfrpsawt7BmxObt98Jev´´;
$signatureCalculada = $miObj->createMerchantSignatureNotif($claveModuloAdmin,$params);
```

6. Once done, it is possible to check whether the value of the signature sent matches the value of the signature calculated, as shown below:

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

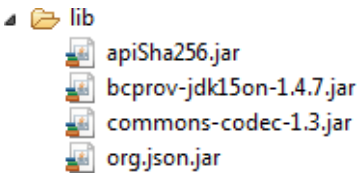
5.1.2 JAVA Library

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

1. Import the library, as shown below:

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

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



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

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

3. Capture the online notification 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 decode this parameter, the "decodeMerchantParameters()" library function must be called, as shown below:

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

Once the call has been made to the "decodeMerchantParameters()" function, the value of any parameter susceptible to being included in the online notification can be obtained (Annex 14.2). To obtain the value of a parameter, the "getParameter()" library function must be called with the name of the parameter, as shown below to obtain the response code:

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

Important Note: *It is important to validate all the parameters that are sent in the communication. To update the status of the online order, this communication must NOT be used, rather the online notification described in the other sections, since the return to browsing depends on the customer's actions in their browser.*

4. Validate the **Ds_Signature** parameter. To validate this parameter, the signature must be calculated and compared with the **Ds_Signature** parameter captured. To do so, the "createMerchantSignatureNotif()" library function must be called with the code obtained from the administration module, and the **Ds_MerchantParameters** parameter captured, as shown below:

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

Once done, it is possible to check whether the value of the signature sent matches the value of the signature calculated, 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");
}
```

5.1.3 .NET Library

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

1. Import the library, as shown below:

```
using RedsysAPIPrj;
```

2. Define an object from 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 validate all the parameters that are sent in the communication. To update the status of the online order, this communication must NOT be used, rather the online notification described in the other sections, since the return to browsing depends on the customer's actions in their browser.*

4. Validate the **Ds_Signature** parameter. To validate this parameter, the signature must be calculated and compared with the **Ds_Signature** parameter captured. To do so, the "createMerchantSignatureNotif()" library function must be called with the code obtained from the administration module, and the **Ds_MerchantParameters** parameter captured, as shown below:

```
var kc = "sq7HjrUOBfkmC576ILgskD5srU870gJ7";
string signatureCalculada = r.createMerchantSignatureNotif(kc, parms);
```

Once done, it is possible to check whether the value of the signature sent matches the value of the signature calculated, as shown below:

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

6 | Other functionalities

6.1. Integration for PSP

If you are a merchant or PSP aggregator, there is a specific integration to be able to operate on behalf of merchants at terminal level using a secret code for PSP.

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

There are no specific flows defined for PSP. The input and output parameters and the error codes can be seen in Input and output parameters and Error codes.

Note: *This integration needs to be activated by the bank.*

6.1.1 Configuration

Merchants-terminals must be associated and configured to be able to send requests from a PSP. The merchant must ask its bank to set this up.

6.1.2 Request and receipt of codes

Two private codes can be received with the protocol established by Redsys.

- Code for 3DES coding of the DS_MERCHANT_PAN field (if necessary)
- Code for signing the request (DS_MERCHANTPARAMETERS) according to the normal X9.19

6.1.3 Sending request to the Virtual POS

As with the payment request sent by a merchant, the PSP has the following fields, which change, as we mentioned earlier, according to the signature and its signature version.

- **Ds_SignatureVersion:** Constant that indicates the version of the signature being used. The value to be used for this PSP integration will be T25V1.
- **Ds_MerchantParameters:** String in JSON format with all the parameters of the request coded in Base 64 and without returns to the shopping cart (the annex sections include the list of parameters that can be sent in a payment request).
- **Ds_Signature:** Signature of the sent information. This is the result of the Mac X9.19 of the JSON string coded 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 extracting the first 4 bytes (8 characters) beginning at the left of the result.

6.1.4 Receiving the result

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

6.1.5 Example of requests

› JSON string:

*{"DS_MERCHANT_AMOUNT":"145","DS_MERCHANT_ORDER":"1446068581","DS_MERCHANT_MERCHANTCODz":"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"}*

› The DS_MERCHANT_PAN parameter is coded in 3DES (if card data is included):

Card number	454881********04
Hexadecimal card number coded in CBC mode 3DES without vector, code F180E06B7A89A88F4A2A52C8EC1C5D1C	*377f34989cf0ae79857a70aa45f3e4c3*

› String in JSON with coded 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"}

Below is the JSON object coded in BASE64, DS_MERCHANT-PARAMETERS field:

eyJEU19NRVJDSEFOVF9BTU9VTlQiOiIxNDUiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjE0NDYwNjg1ODEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiwiRFNfTUVSQ0hBTlRfUEFOIjoiMzc3ZjM0OTg5Y2YwYWU3OTg1N2E3MGFhNDVmM2U0YzMiLCJEU19NRVJDSEFOVF9FWFBJUllEQVRFIjoiMTUxMiIsIkRTX01FUkNIQU5UX0NWVjIiOiIxMjMif Q==

› Signing the request information:

The complete signature is calculated on the string obtained in the previous step (DS_MERCHANT_PARAMETERS) based on standard x9.19.

We get the MAC 5D823A402DE70705 with the coding code 269289DA6EAD0B20928C8F2F2F6BC752 and the ds_merchant_parameters.

The ds_signature field comes from the first 4 bytes (8 characters) starting at the left of the MAC result from the previous step **5D823A40.**

› **Compose the request message:**

› ***Ds_SignatureVersion:*** *T25V1*

› ***Ds_MerchantParameters:***

eyJEU19NRVJDSEFOVF9BTU9VTlQiOiIxNDUiLCJEU19NRVJDSEFOVF9PUkRFUiI6IjE0NDYwNjg1ODEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVENPREUiOiI5OTkwMDg4ODEiLCJEU19NRVJDSEFOVF9DVVJSRU5DWSI6Ijk3OCIsIkRTX01FUkNIQU5UX1RSQU5TQUNUSU9OVFlQRSI6IjAiLCJEU19NRVJDSEFOVF9URVJNSU5BTCI6IjEiLCJEU19NRVJDSEFOVF9NRVJDSEFOVFVSTCI6Imh0dHA6XC9cL3d3dy5wcnVlYmEuY29tXC91cmxOb3RpZmljYWNpb24ucGhwIiwiRFNfTUVSQ0hBTlRfRUFFOIjoiMzc3ZjM0OTg5Y2YwYWU3OTg1N2E3MGFhNDVmM2U0YzMiLCJEU19NRVJDSEFOVF9FWFBJUllEQV RFIjoiMTUxMiIsIkRTX01FUkNIQU5UX0NWVjliOiIxMjMifQ==

› ***Ds_Signature:*** *5D823A40*

7 | EMV3DS Authenticated Requests (Advance)

In this integration you will go straight to authenticating the payments with the new regulations for 3DSecure (EMV3DS) without needing any specific development.

More parameters can be sent to the card companies and issuers for information purposes and thus improve the ratio of authentications without intervention of the card holder. For this use, the JSON input parameter DS_MERCHANT_EMV3DS can be added in the pre-authorisation payment requests.

For more information about this input parameter, see the document *"TPV-Virtual Parámetros Entrada-Salida.xlsx"*.

7.1. Example of payment with additional EMV3DS information

Below is an example of the Ds_Merchant_Parameters parameter before being coded 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 (Advance)

According to standard PSD2 (entry into force on 14 September 2019), European directive intended to improve security and enhance customer authentication in e-commerce transactions, a series of exemptions to the use of SCA have been defined that may be marked in the authorisation request.

Parameter	Possible values
DS_MERCHANT_EXCEP_SCA	LWV, TRA, MIT, COR, ATD

› **LWV:** low value exemption (up to €30 with max. 5 transactions or €100 accumulated per card, these counters are controlled at card issuing bank level).

› **TRA:** exemption for the acquiring bank/merchant using a risk analysis system (and being considered low risk).

› **MIT:** merchant-initiated transaction (without being associated with a customer event or action) that is outside the scope of the PSD2. This is the case of subscription payments, recurring payments, etc.; any that require the storage of the customer's payment credentials (COF) or its equivalent through tokenised scheduled payment operations (use of "pago por referencia" ["reference payment"] functionality in payments initiated by the merchant). All merchant-initiated payments (MIT) require that when the customer initially gives permission for the merchant to use their payment credentials, this "permission or mandate" is given through an SCA authenticated operation.

› **COR:** exemption restricted to cases of use of a secure corporate payment protocol.

› **ATD:** delegated authentication exemption.

Note: *It must be taken into account that for LWV, TRA and COR exemptions, the first option will be to mark the exemption in the authentication to improve the user experience. As such, if the issuer does not want to accept the exemption proposal and requires SCA, it can request its authentication without having to reject the transaction (challenge required EMV3DS).*

8.1. Operation of the SCA exemption requests

Although a merchant can make a request to the payment service offered by its bank for a payment to qualify for one of the SCA exemptions provided for, the final acceptance of this request to "not authenticate" the holder of a card depends on the bank who issued the card. This is why it is called an exemption "request". Marking a transaction with an exemption request does not guarantee that it will be accepted by the bank who issued the card. If it is not accepted, the transaction will have to be authenticated.

As such, the Virtual POS adhering to the best practices of the sector, and to ensure the best payment experience for the user, the authentication flow with the issuer will always be prioritised (via EMV3DS, if the issuer allows), notifying the issuer in said request of the merchant's preference to not authenticate based on the requested exemption. This improves usability, and in the event that the issuer does not accept the exemption, the user is authenticated at the same time. If the issuer agrees to apply the requested exemption, the authentication flow will be closed without requiring any screens to be displayed or any action by the card holder, being a transparent process for the user.

Note: *The merchant can find out which exemptions it is allowed. To do so, it can make an "Inicia Petición" ["Start Request"] request via REST according to the indications provided in the REST Integration Guide, PSD2 Adaptations - Start Request Message (See my exemptions) section.*

8.1.1 Request with delivery of exemption

Once the exemptions that can be used are known, the merchant must add the **DS_MERCHANT_EXCEP_SCA** parameter in the transaction with one of the permitted exemptions, as shown 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. Merchant initiated transactions (MIT)

› **What is a MIT transaction?**

Merchant initiated transactions without possible interaction with the customer. For example, monthly payment of a bill or subscription. This type of exemption requires the operation to be marked as COF (Credential on File) according to the specific use being made of the stored credentials.

Initial MIT: This transaction must be authenticated with SCA and allows exemptions to be used. The card holder is present and is granted the permission and agrees with the merchant on the conditions for which their payment information will be used for subsequent charges in accordance with a service that is continuously provided over time. This operation must be duly marked following the Card-on-File (COF) specification to indicate that the card information has been stored for subsequent payments.

Subsequent MIT: In these MIT transactions, the card holder is not present and cannot be authenticated. This operation must be duly marked following the Card-on-File (COF) specification to indicate that a recurring transaction is being performed on an initial MIT that was authenticated.

› **How does PSD2 affect the 1-click-payment?**

Not all the operations in which card information/stored credentials (COF) are used are considered MIT. For example, the 1-click-payment operation, where the customer's credentials are stored or tokenised (reference payment) in order to make it as easy as possible at the time of payment without having to ask the customer for the information again, can NOT be considered a transaction initiated by the merchant since the card holder is present at the time of the purchase. In this case, according to PSD2, and whilst no other exemptions are applied, the use of strong authentication is required.

Note: *The full list of all the SIS rejection codes are available in the document "TPV-Virtual Parámetros Entrada-Salida.xlsx".*

Note 2: *Access the COF Guide for more information "Especificaciones COF ECom v1.1.pdf".*

8.2.1 MIT transactions and use of tokenisation (reference payment)

The customer's payment information is often tokenised so that Redsys is responsible for its secure storage and for ensuring compliance with the PCI DSS security standards in order to generate payments initiated by the merchant at a later date, without the card holder being present.

In these cases, in the initial transaction in which the token or reference is requested, 3D Secure must be used under PSD2 to apply strong authentication, and the use for which they are being stored must be correctly marked using COF parameters. Furthermore, in subsequent payments initiated by the merchant with the token/reference, the COF parameters must be indicated according to that indicated in the original transaction. If these parameters are omitted in subsequent transactions, the SIS Virtual POS will try to automatically incorporate the suitable use marking information and any additional information required according to the card company based on that indicated in the original transaction e.g. Original transaction ID required for COF payments in Visa "DS_MERCHANT_COF_TXNID")

› **Use of tokenisation and MIT:**

In some cases, reference payment or tokenisation is used to make successive transactions without the card holder being present. For these cases, the merchant must mark the first transaction as MIT and always send the COF parameters and the reference generation request. For subsequent payments, the MIT exemption, the reference to be used and the corresponding COF parameters must be marked.

1. In the first transaction, "send the COF parameters" (DS_MERCHANT_COF_INI, DS_MERCHANT_COF_TYPE) and the reference generation request parameters must be marked.

2. In successive payments, DS_MERCHANT_COF_INI (N), DS_MERCHANT_EXCEP_SCA must be marked as well as MIT and the reference to be used.

9 | Advanced EMV3DS Functionalities (Advance)

In view of the PSD2 regulation that requires the authentication of all e-commerce transactions, the protocol offers the possibility of also authenticating those transactions in which the card holder is not present. There are two different cases:

› **Recurring Transactions:** that offer the merchant the possibility of making subsequent authenticated recurring transactions.

› **OTA transactions:** which offer the merchant the possibility of making a single authentication transaction for the full amount and, in association, collect partial payments in the different businesses that form part of the transaction at different times after the authentication.

These transactions have two different parts.

› **Initial request with card holder authentication.** In this request, the card holder is present and SCA is authenticated. By requesting that it will be a 3RI transaction, the information needed to carry out the subsequent transactions will be returned.

› **Subsequent requests.** Successive authorisation transactions are carried out using the information obtained in the initial authentication transaction.

Note: *this integration needs to be activated by the bank and will only be available with protocol version 2.2 of the EMV3DS.*

Note 2: *these functionalities are subject to possible changes required by the card companies, and as such the following sections show an approximation with the details currently known.*

9.1. Recurring EMV3DS payments ("3RI-Recurring")

This type of operation offers the merchant the possibility of authenticating the recurring payments, exercising the change of responsibility and the merchant is protected in the event of fraud with these payments.

The first recurring payment must always be authenticated by the card holder. Successive payments will be authenticated (3RI) without intervention from the card holder, but the merchant must provide the Virtual POS with the information needed for the authentication.

Note: *The initial recurring payment can be made by redirection to obtain the necessary information, although successive payments must be made via REST. Therefore, the indications in the REST Integration Guide, Advanced Functionalities - Advanced EMV3DS Functionalities, Recurring Payments - Successive Recurring Payment section must be followed.*

9.1.1 Initial recurring payment

The first payment will be processed as an authenticated EMV3DS transaction (frictionless or challenge), following the normal steps. The COF parameters must be marked to request it as a recurring transaction.

The transaction must be marked as Initial Recurring COF in the transaction:

- DS_MERCHANT_COF_INI = "S" → OPERACIÓN COF INICIAL SI
- DS_MERCHANT_COF_TYPE = "R" → OPERACIÓN COF RECURRENTE

If the card belongs to the EMV3DS 2.2 protocol or above, the response received from the Virtual POS will return the parameters needed to be able to make successive 3RI-Recurring transactions. These parameters are:

- **Ds_Merchant_Cof_Txnid:** CAMPO OPCIONAL [OPTIONAL FIELD], Initial transaction ID to send in successive authorisations.
- **Ds_EMV3DS:** this will comprise the following fields:
  - **Eci:** authentication type indicator
  - **traceID:** 3RI reference identifier
  - **threeDSRequestorPriorAuthenticationInfo:**
    - threeDSReqPriorRef: {reference of the initial authentication}
    - threeDSReqPriorAuthMethod: {initial authentication method}
    - threeDSReqPriorAuthTimestamp: {initial authentication timestamp}
    - threeDSReqPriorAuthData: {additional information}

› **Example of response:**


```
{
"Ds_Amount":"1000",
 "Ds_Currency":"978",
"Ds_Order":"1552572812",
"Ds_MerchantCode":"999008881",
"Ds_Terminal":"2",
"Ds_Response":"0000",
"Ds_AuthorisationCode":"694432",
"Ds_TransactionType":"0",
"Ds_SecurePayment":"1",
"Ds_Language":"1",
"Ds_CardNumber":"****************",
"Ds_Card_Type":"C",
"Ds_MerchantData":"",
"Ds_Card_Country":"724",
"Ds_Card_Brand":"1",
"Ds_Merchant_Cof_Txnid": "Initial transaction ID to send successive authorisations",
"DS_EMV3DS" : {
        "Eci":"05",
        "traceId":"0100000000000000",
        "threeDSRequestorPriorAuthenticationInfo":{
            "threeDSReqPriorRef":"5a31771c-6e88-4378-9f99-b114d90f8040",
            "threeDSReqPriorAuthMethod":"01",
            "threeDSReqPriorAuthTimestamp":"201912051020"
        }
}
```


9.2. Authenticated payments from different merchants ("3RI-OTA")

This type of operation offers the merchant the possibility of making a single authentication transaction for the full amount and, in association, collect partial payments in the different businesses that form part of the transaction.

Note: *This solution can be extended to online travel agencies and solutions such as Marketplace.*

Note 2: *The initial OTA payment can be made by redirection to obtain the authentication information, and the successive OTA payments must follow the indications of the REST Integration Guide, Advanced Functionality - Advanced EMV3DS Functionalities, Authenticated Payments from Different Merchants - OTA Payment section.*

9.2.1 Initial OTA payment

The first payment shall be processed as an authenticated EMV3DS transaction. To request that it be an authenticated OTA transaction, the following parameters must be sent in the request:

› DS_MERCHANT_OTA = "S" → OPERACIÓN OTA INICIAL SI

› DS_TRANSACTION_TYPE = "7" → PETICIÓN DE AUTENTICACIÓN

If the card belongs to the EMV3DS 2.2 protocol or above, the response received from the Virtual POS will return the parameter needed to be able to make successive 3RI-Recurring transactions. This parameter is:

› **Ds_EMV3DS:** this will comprise the following fields:

- **authenticacionValue:** OPTIONAL FIELD with the authentication value
- **Eci:** authentication type indicator
- **traceID:** 3RI reference identifier
- **threeDSRequestorPriorAuthenticationInfo :**
  - threeDSReqPriorRef: {reference of the initial authentication}
  - threeDSReqPriorAuthMethod: {initial authentication method}
  - threeDSReqPriorAuthTimestamp: {initial authentication timestamp}
  - threeDSReqPriorAuthData: {additional information}

› **Example of response:**


```
{
"Ds_Amount":"1000",
"Ds_Currency":"978",
"Ds_Order":"1552572812",
"Ds_MerchantCode":"999008881",
"Ds_Terminal":"2",
"Ds_Response":"0000",
"Ds_AuthorisationCode":"694432",
"Ds_TransactionType":"7",
"Ds_SecurePayment":"1",
"Ds_Language":"1",
"Ds_CardNumber":"****************",
"Ds_Card_Type":"C",
"Ds_MerchantData":"",
"Ds_Card_Country":"724",
"Ds_Card_Brand":"1",
“DS_EMV3DS” : {
        "Eci":"05",
        "traceId":"0100000000000000",
        “authenticationValue”: “AJkBAoIpI5dGUThEhSkjAAAAAAA=”,
        "threeDSRequestorPriorAuthenticationInfo":{
            "threeDSReqPriorRef":"5a31771c-6e88-4378-9f99-b114d90f8040",
            "threeDSReqPriorAuthMethod":"01",
            "threeDSReqPriorAuthTimestamp":"201912051020"
        }
}
```

10 | Testing environment

There is a testing environment where tests needed to check that the system is working properly once the integration has been carried out are run, before implementing in the real environment.

Below are the URLs for accessing the administration portal and the address of the service for running the tests. Contact your bank for the access information.

The URL for sending payment orders is as follows:

https://sis-t.redsys.es:25443/sis/realizarPago

The URL for accessing the administration module is as follows:

https://sis-t.redsys.es:25443/canales

Note: *The testing environment will be the same as the real environment, the only exception being that the payments made in this environment will not have any accounting value.*

Redsys provides generic testing data for all customers. As mentioned before, contact your bank to get the data for your business.

› **Generic Testing Data:**

- › **Merchant code (Ds_Merchant_MerchantCode):** Enter the number provided by your bank (e.g. 999008881).
- › **Terminal (Ds_Merchant_Terminal):** Enter the number provided by your bank (e.g. 01)
- › **Passcode:** sq7HjrUOBfKmC576ILgskD5srU870gJ7

Authorised Cards:

- › **Authorised card 1:**
  - **Numbering:** 4548812049400004
  - **Expiry:** 12/20
  - **CVV2 Code:** 123
  - For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**
- › **Accepted card 2 (3DSecure 1.0 with *"iniciapeticion"* NO_3DS_V2):**
  - **Numbering:** 4548812049400004
  - **Expiry:** 12/20
  - **CVV2 Code:** 123
  - For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**
- › **Accepted card 3 (EMV3DS 2.1 *with "iniciapeticion" with threeDSMethodURL with FRICTIONLESS authentication)***
  - **Numbering:** 4918019160034602
  - **CVV2 Code:** 123
  - For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**
- › **Accepted card 4 (EMV3DS 2.1 with *"iniciapeticion" with threeDSMethodURL with FRICTIONLESS authentication)***
  - **Numbering:** 4548814479727229
  - **Expiry:** 12/20
  - **CVV2 Code:** 123
  - For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

› **Authorised card 5 (EMV3DS 2.1 with *"iniciapeticion" with threeDSMethodURL with CHALLENGE authentication)***

- **Numbering:** 4918019199883839
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

› **Accepted card 6 (EMV3DS 2.1 with *"iniciapeticion" with threeDSMethodURL with CHALLENGE authentication)***

- **Numbering:** 4548817212493017
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

Rejected Cards (Response Code 190):

› **Rejected card 1:**

- **Numbering:** 5576440022766500
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

› **Rejected card 2 (EMV3DS 2.1 with *"iniciapeticion" with threeDSMethodURL with FRICTIONLESS authentication)***

- **Numbering:** 4907277775205123
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

› **Rejected card 3 (EMV3DS 2.1 with *"iniciapeticion" with threeDSMethodURL with FRICTIONLESS authentication)***

- **Numbering:** 4907271141151707
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

10.1. Testing cards version 2.2 (Advance)

› **Generic Testing Data:**

› **Merchant code (Ds_Merchant_MerchantCode):** Enter the number provided by your bank (e.g. 999008881)

› **Terminal (Ds_Merchant_Terminal):** Enter the number provided by your bank (e.g. 01)

› **Passcode:** sq7HjrUOBfKmC576ILgskD5srU870gJ7

Authorised Cards:

› **Accepted card 7 (3DSecure 2.2 with *"iniciapeticion" without threeDSMethodURL with FRICTIONLESS authentication):***

- **Numbering:** 4548816134581156
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

› **Accepted card 8 (3DSecure 2.2 with *"iniciapeticion" with threeDSMethodURL with CHALLENGE authentication):***

- **Numbering:** 4548816131164384
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

› **Accepted card 9 (3DSecure 2.2 with *"iniciapeticion" without threeDSMethodURL accepts exceptions with FRICTIONLESS without CHALLENGE exemptions):***

- **Numbering:** 4548815324058868
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

› **Accepted card 10 (3DSecure 2.2 with *"iniciapeticion" without threeDSMethodURL only accepts MIT exceptions with FRICTIONLESS without CHALLENGE exemptions or other exemptions):***

- **Numbering:** 4548815374025114
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

› **Accepted card 11 (3DSecure 2.2 with *"iniciapeticion" without threeDSMethodURL only accepts 3RI-OTA payments):***

- **Numbering:** 5576441563045037
- **Expiry:** 12/20
- **CVV2 Code:** 123
- For secure purchases that require card holder authentication, **the personal authentication code (CIP) is 123456.**

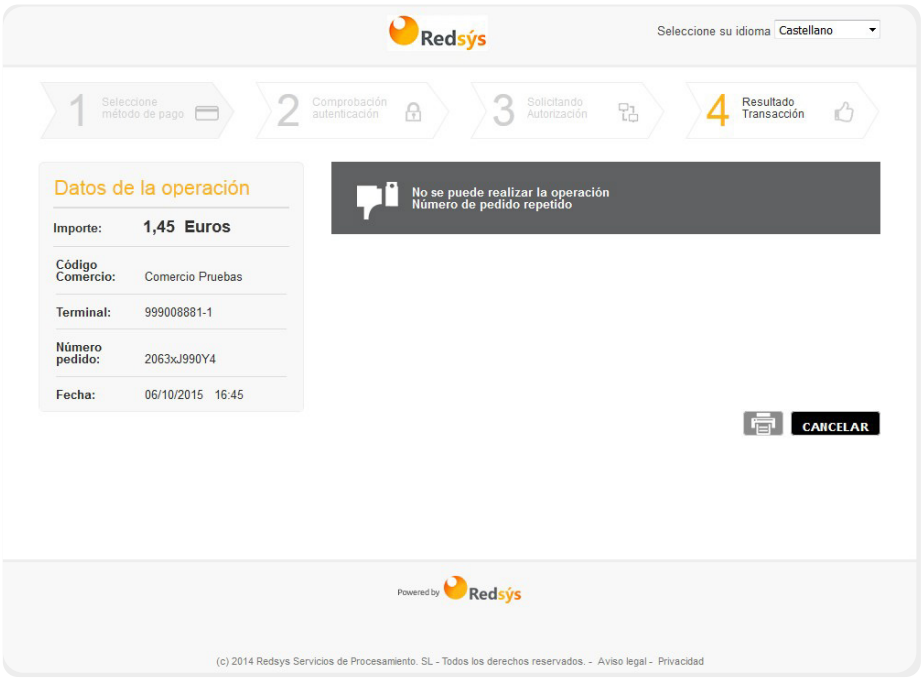
11 | Error codes

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

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

The error that occurred can be obtained by consulting the source code of the transaction results page, as shown below:

› Transaction results page



› Transaction results page (source code)

```
192 <div class="result-code error">
193 <p>
194 <text lngid="noSePuedeRealizarOperacion">No se puede
195 <br>Número de pedido repetido<br>
196 <!--SIS0051:-->
197 </p>
198 </div>
199 </div>
200 <div class="result-info">
201 <table class="tableresults">
202 <tbody></tbody>
```

12 | Frequent errors

› **I have not received the online response of the Virtual POS purchases.**

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

In the "Notificaciones" ["Notifications"] tab, you can see all the notifications that have been sent to the merchant, both those that were accepted and those in which the merchant's server has returned an error.
Given that the response of the servers follows an international protocol, the specific meaning of the generic errors can be seen on a number of websites. For example:

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

If the merchant is not able to correctly install a server that allows it to receive the HTTP notifications and wishes to request a direct and personalised consultation, it must contact its bank.

› **I have not received the confirmation email of the Virtual POS purchases.**

It often happens that notifications sent by the Virtual POS do reach the merchant, but they get sent straight to the Junk Emails. Check your Junk Emails.

› **I have not received the parameters with the OK and KO result of the purchase transactions in my URLs.**

The OK and KO URLs must **NEVER** be used to receive the parameters with the transaction results. These are URLs that must only be used to redirect the card holder to the merchant's website for commercial purposes or for the merchant to show the card holder the receipt, if it is configured as such.

› **Signature error**

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

› That the information used to do the signature is the same as that sent in the form, taking into account that any change to the value or format after calculating the signature will make it incorrect.

› That the passcode used by the merchant is the same as the code that the merchant has loaded in the administration module (merchants' section).

› It must be checked that merchants are not sending blank spaces in the signature. If the request is made through the URL or the Safari browser, it may turn "+" symbols into blank spaces. To ensure that this does not happen, "+" symbols in the signature must be replaced by "%2B" (encoded URL value).

› If the merchant cannot find the erroneous parameter, it must contact the Redsys Customer Service Centre or the Redsys Integration Support department if its bank has given it the contact information.

› **My business has rejections due to a repeated number, but I don't have any record of having repeated them.**

This often happens because the merchant's platform is only generating repeated order numbers when it receives rejections or authorisations, but it is repeating them when the transactions are not completed. There are two options in this circumstance:

› Ask the Support service to configure the POS to be able to repeat order numbers. Maximum of one transaction authorised per day, and with no limit for rejected transactions.

› Always generate different order numbers, not just for authorised and rejected transactions, but for those that have not been completed after a certain amount of time.

› **I need to return a transaction, but I can't see the return option in the administration module.**

This happens because the user accessing Channels does not have permission to make returns. If you need this permission, contact your bank.

13 | FAQs

› **I am a merchant and I need to know the encryption code of my Virtual POS.**

The To see the Virtual POS code, follow the steps below:

- › Access your Virtual POS administration module.
- › Select the “comercio" [“merchant"] option and click “ver clave” [“see code"].
- › Enter your Virtual POS user password and click OK.
- › You will be able to see the merchant code for 10 seconds.

› **My merchant username to access the Channels administration module is blocked. How can I unblock it?**

There is an “He olvidado mi contraseña” ["I have forgotten my password"] link underneath the user and password boxes. After clicking on it, enter your username and confirm the email address for sending the new password.

› **I want to change my business' OK and KO URLs.**

The OK and KO URLS are Internet addresses defined by the merchant where the card holder can be redirected to once the purchase receipt screen has been displayed. There are two ways to define these URLs:

- › If they are configured in the administration tool.
- › If the merchant sends them in the payment form in the Ds_Merchant_UrlOK and Ds_Merchant_UrlKO fields, the merchant must change them in the payment form that is sent.

14 | Annexes

14.1. Request parameters

In the SIS Virtual POS payment request, a series of mandatory data and other optional data must be sent, which will depend on the type of transaction and operation being performed.

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

14.1.1 Payment/pre-authorisation example

Below is an example of the Ds_Merchant_Parameters parameter before being coded 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 PAGO [PAYMENT]*
** DS_MERCHANT_TRANSACTIONTYPE":"1" for PREAUTORIZACIÓN [PRE-AUTHORISATION]*

14.2. Parameters of the online notification

The online notification is a parallel communication and independent to the customer's Virtual POS browsing process, through which the merchant is sent a POST with the information of the transaction result.

The result of the transaction is reported in the Ds_Response or "Código de respuesta" ["Response code"] parameter. This response code will also be reported in the transactions query provided that the transaction is not authorised, as shown in the following image:

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, in the merchant's server side, there must be a process that collects this response and performs the tasks necessary to manage the orders. To do so, a parameter will have to be provided; a URL where these responses can be received in the online form that is sent when making an authorisation request (see the Ds_Merchant_MerchantURL in "Datos del formulario de pago" ["Payment form information"]). This URL will be a CGI, Servlet, etc. developed in the language considered appropriate by the merchant to integrate in its server (C, Java, Perl, PHP, ASP, etc.), capable of interpreting the response sent by the Virtual POS.

Note: *The reported data will also be included in the OK URL (Ds_Merchant_UrlOK) or KO URL (Ds_Merchant_UrlKO) if the merchant has activated the send parameters in the response redirection.*

Note 2: *The full list of SIS parameters is available in the "TPV- Virtual Parámetros Entrada-Salida.xlsx" document.*

14.3. Payment retries

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

In order for the customer to be shown this option, the merchant must have a specific configuration in the Virtual POS administration module (Allow Order Retry = SI [YES], with retries) and fulfil the following requirements:

› The merchant must be able to receive different notifications associated with the same order number and only take into account the first notification that identifies that the transaction has been authorised, or if it does not receive an authorised transaction notification, it must take into account the last notification received about the order number in question.

› At present, the retry option is not compatible with the payment modules provided by Redsys for virtual stores such as Prestashop, Megento, etc. In payment modules external to Redsys, this option may not work correctly, resulting in an error when updating the orders in the virtual store.

The following image shows an example of the receipt displayed to the customer, offering the retry option: