Integración inSite integration

Version 2.2

09/12/2021

RS.ADQUI.CNOPRESENCECOMM.LIST.0000

Redsys · C/ Francisco Sancha, 12 · 28034 · Madrid · Spain

Authorisation and version control

Version	Date	Concerns	Brief description of the change
1.0	03/04/2017	The whole document	Initial version of the document
1.1	28/05/2018	The whole document	Inclusion of section 2 of Concepts and Advantages, and inclusion of improvements in the wording for greater clarity and understanding.
1.2	06.03.2018	Library download url is modified	The library download url is modified (point 5)
1.3	07.03.2019	The url to access the test environment is included	Point 4 includes the url to access the test environment.
1.4	05.07.2019	The new SIS authentication flow via WS/REST is included	Point 6 includes the new authentication flow via WS / REST
2.0	15/10/2019	The whole document	Version 2 is included with new features.
2.1	18/11/2019	The whole document	Modifications in the interface and inclusion of new functionalities
2.2	09/12/2021	Point 4.1	A new option parameter is included to hide the logo of the entity.

TABLE OF CONTENTS

1. Purpose of this guide

This document describes how to implement the Virtual POS **inSite** connection in a webstore, a connection model that allows the client's payment data to be collected without the client having to leave the merchant's website.

The advantages of this type of integration are several and are described in greater detail in the following section. The main purpose is to have a fast, simple payment process that is integrated as much as possible into the webstore pages, fully adapted to the design of the online merchant, easy to use and integrate, but at the same time maintaining the security of the payment data entered by the client, preventing the merchant from having to endure costly security processes derived from the mandatory compliance with PCI DSS regulations1.

This guide focuses on the specific features of this type of integration. For general concepts of the operation of the Virtual SIS POS service, please consult the corresponding documentation.

[1] PCI DSS (Payment Card Industry Data Security Standard) sets forth the security requirements that parties involved in the card payment process must comply. The solution described in the document makes it easier to consider the process as a SAQ-A type, by basing the implementation on iframes whose content is only accessible by our servers.

2. Concepts and benefits of inSite connection

With the inSite payment solution, the merchant or online shop achieves a series of advantages that favour the increase in sales conversion:

- **A simple and satisfactory payment experience** for your clients, as it **is fully integrated** into the merchant's website and without navigating incidents.
- **Greater control** of the checkout and payment flow since all requests are made synchronously by the webstore server and without the need for asynchronous "listening" processes.
- **Ease of use in its integration**
- **Elevated level of security,** similar to the solution based on redirecting the client to an external payment page.

In short, in addition to a payment process fully integrated into the buyer's checkout, the merchant is allowed greater **flexibility and control** in the payment process, being able to separate the steps of data capture and execution of the operation.

There are **two options when integrating the inSite connection**:

- Unified integration (all in one)
- Integration of independent elements

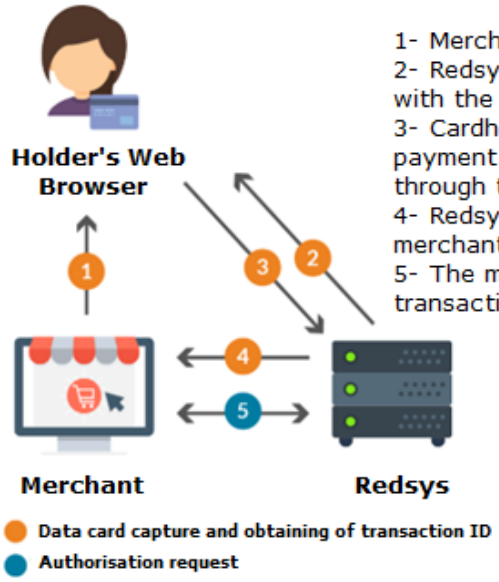
In both cases, integration can be done using code snippets which are given as examples, where only own values such as the merchant identifier or the keys used need to be changed. In addition, Redsys provides libraries for the main programming languages as an additional support.

In the inSite connection, the necessary pieces or "fields" of the payment form are provided to the online store so that they are integrated one by one (or as a set) perfectly embedded in the checkout page of the webstore and also each element allows customization of the design with configurable styles, in perfect harmony with the design of the rest of the merchant's website.

Security is preserved so that the resulting form with client payment information remain inaccessible to the same merchant's server or third parties that may have compromised the merchant's web server.

3. Flow overview

The following diagram shows the general flow of a transaction conducted with the new Virtual POS scheme.



1- Merchant submits payment form to the holder.
2- Redsys completes the merchant's payment form with the card data capture fields.
3- Cardholder enters card details and accepts payment.Data are sent to Redsys without passing through the merchant's server.
4- Redsys generates a transaction ID and notifies the merchant.
5- The merchant launches the Host to Host payment transaction using the transaction ID received.

In short, the payment details entered by the client are sent from the merchant's page to the Virtual POS, where they are temporarily stored and associated with an Operation ID that is returned to the merchant. With this Operation Id (which is an "alias" of the customer's payment data) the merchant can subsequently request directly to the virtual POS to conduct the requested payment operation.

4. Payment Page - Obtaining Transaction IDs

As a first step to integrate the card data entry fields directly into the web page itself, the Javascript file hosted on the Redsys server must be included with the following line of code (the file varies depending on whether the test environment or the real production environment is to be used):

- Test environment:

```
<script src="https://sis-t.redsys.es:25443/sis/NC/sandbox/redsysV2.js"></script>
```

- Production environment:

```
<script src="https://sis.redsys.es/sis/NC/redsysV2.js"></script>
```

The next step to include the elements of the payment form depends on the alternative to be implemented. There are **two options when integrating the inSite connection**:

a) **Unified integration (all in one)**: The payment elements, such as the boxes for entering card number, expiry date, cvv and payment button are embedded as a single element that adapts to the merchant's page (responsive), with light design and customisable CSS styles. It includes by default animated interactive assistance and good user accessibility.
b) **Integration by independent elements**: the fields must be embedded each one independently within the web page of the webstore, which allows full control of the design, position, error management, etc.

4.1 Unified integration (all in one)

In this mode of the inSite integration, a single iframe of very small size will be provided in which the complete payment form will be included. As for the customisation of it, it will be possible to apply the CSS styles required by the merchant to the different elements.

It includes interactive elements that facilitate usability, such as card brand recognition, showing the card logo, verification of the formats and contents as the user enters them and visually highlighting immediately if any of them are incorrect (check digit, expiry date, etc.).

Example:

Credit or debit card

Once the JS file has been imported, the payment form must be created. To securely collect card data, Redsys will create and host the input fields for said data.

A single container must be created, with a unique id since it must be indicated so that the iframe is generated with the elements in it.

```
<div id="card-form-example"></div>
```

A message listener function must be included to receive the operation ID when it is generated. The storeIdOper function must be *used* with the following definition:

storeIdOper(event, idToken, idErrorMsg, validationFunction);

the event collected by the listener (*event*) must be specified, the ID of the DOM element must be stored, the operation ID must be stored once it is generated (idToken), the identifier of the element in which the error codes will be stored in case there are data validation errors (*idErrorMsg*). In the example below both are stored in an input of type "hidden."

Optionally, the option of executing a proprietary function to perform pre-validations by the merchant is available. The generation of the transaction ID will only continue if the validation function executed by the merchant returns a *True* value.

```
<input type="hidden" id="token" ></input>
<input type="hidden" id="errorCode" ></input>
<script>
function merchantValidationEjemplo(){
//Insertar validaciones…
return true;
}
<!-- Listener -->
window.addEventListener("message", function receiveMessage(event) {
        storeIdOper(event,"token", "errorCode," merchantValidationEjemplo);
});
</script>
```

Once the listener has been prepared to receive the data, the provided function will be called to generate the card data entry elements:

```
        <!-- Petición de carga de iframe -->
        getInSiteForm('card-form', estiloBoton, estiloBody, estiloCaja, estiloInputs, 'Pagar con Redsys,' fuc, terminal, merchantOrder, 'ES,' true);
```

As parameters of the functions, the id of the container reserved for its generation will be specified, as well as the style required for the different elements (CSS format). In this mode, styles can be included for different elements:

- *Payment button* Full → customization of the payment button is allowed.
- *Form body* →It is recommended for setting a background colour or modifying the colour or style of text.
- *Data entry box* → A different background colour can be established for the data entry box. The colour of the text applied to this element will be applied to the placeholder of the elements.
- *Data entry inputs* → Its use is recommended if you want to use a different font or modify the text colour of the data entry fields.

Additionally, the text to be included in the button can be customized and, finally, the value of the FUC, terminal and order number (alphanumeric text string of between 4 and 12 positions) must be informed in the iframe load request with the payment form.

Two optional parameters are included in the iframe loading.

The first parameter sets the language of the texts. The list of codes can be found in the Language directory section.

Both the SIS code and the ISO 639-1 language code can be used.

If no language is established or an incorrect code is reported, the default language will be Spanish.

The second parameter establishes if you want to show the logo of the entity, indicating **true** if you want to show it or **false** if not.

If no value is set, the entity's logo will be displayed by default.

***The merchantOrder used in loading the iframe and generating the idOper must be reused in the subsequent authorization request.**

In this way, when the client enters his/her card details in the elements generated by Redsys and clicks on the payment button, an ID associated with the transaction will be generated and stored in the merchant's form so that the client can formalise the purchase without having to process card details.

4.2 Integration of independent elements

In this inSite integration modality, merchants will be allowed to fully customize the payment page, so they can place the card data entry fields and the payment button with total freedom, by generating differentiated and customizable iframes with styles for each of them.

Once the file has been imported, the payment form must be created. To securely collect card data, Redsys will create and host the input fields for said data.

Empty containers must be created, with a unique id since it must be specified so that the data entry field is generated in it.

```
<div class="cardinfo-card-number">
 <label class="cardinfo-label" for="card-number">Card number</label>
 <div class='input-wrapper' id="card-number"></div>
</div>
<div class="expiry-date">
 <div class="cardinfo-exp-date">
        Expiration <label class="cardinfo-label" for="expiration-date">Month (MM)</label>
        <div class='input-wrapper' id="expiration-month"></div>
 </div>
<div class="cardinfo-exp-date2">
        Expiration <label class="cardinfo-label" for="expiration-date2">Year (AA)</label>
        <div class='input-wrapper' id="expiration-year"></div>
</div>
<div class="cardinfo-cvv">
        <label class="cardinfo-label" for="cvv">CVV</label>
        <div class='input-wrapper' id="cvv"></div>
</div>
</div>
```

In this example, these are the elements with id "card-number", "*expiration-month*", "*expiration-year*" and "*cvv*".

A message listener function must be included to receive the operation ID when it is generated. The storeIdOper function must be *used* with the following definition:

storeIdOper(event, idToken, idErrorMsg, validationFunction);

the event collected by the listener (*event*) must be specified, the ID of the DOM element must be stored, the operation ID must be stored once it is generated (idToken), the identifier of the element in which the error codes will be stored in case there are data validation errors (*idErrorMsg*). In the example below both are stored in an input of type "hidden."

Optionally, the option of executing a proprietary function to perform pre-validations by the merchant is available. The generation of the transaction ID will only continue if the validation function executed by the merchant returns a *True* value.

```
<input type="hidden" id="token" ></input>
<input type="hidden" id="errorCode" ></input>
<script>
function merchantValidationEjemplo(){
//Insertar validaciones…
return true;
}
<!-- Listener -->
window.addEventListener("message", function receiveMessage(event) {
        storeIdOper(event,"token", "errorCode," merchantValidationEjemplo);
});
</script>
```

Once the sending of data and subsequent reception has been prepared, the function provided will be called to generate the card data entry elements:

```
<!-- Petición de carga de iframes -->
getCardInput('card-number',estilosCSS);
getExpirationMonthInput('expiration-month', estilosCSS);
getExpirationYearInput('expiration-year', estilosCSS);
getCVVInput('cvv', estilosCSS);
getPayButton('boton', estilosCSS, 'Pagar con Redsys', fuc, terminal, merchantOrder);
```

As parameters of the functions, the id of the container reserved for its generation will be indicated, as well as the style required for it (CSS format). Additionally, the text of the payment button can be customized and, finally, the value of the FUC, terminal and order number (alphanumeric between 4 and 12 positions) must be informed in the request to load the iframe with the payment button.

The merchantOrder used in the idOper generation will be reused in the subsequent authorisation request.

In this way, when the client enters his/her card details in the elements generated by Redsys and clicks on the payment button, an ID associated with the transaction will be generated and stored in the merchant's form so that the client can formalise the purchase without having to process card details.

5. Error directory

When the button generated by Redsys is pressed, the validations will be launched based on the data entered. The following catalogue of errors may be received.

The description of the errors is included, but it is the merchant's responsibility to show them in the way it deems appropriate.

msg1	You have to fill in the card details
msg2	The card is mandatory
msg3	The card must be numeric
msg4	The card cannot be negative
msg5	The expiration month of the card is mandatory
msg6	The card's expiration month must be numeric
msg7	The card's expiration month is incorrect
msg8	The expiration month of the card is mandatory
msg9	The card's expiration month must be numeric
msg10	The expiration year of the card cannot be negative
msg11	The card's security code is not the correct length
msg12	The card's security code must be numeric
msg13	The card's security code cannot be negative
msg14	The security code is not required for the card
msg15	Card length is not correct
msg16	You must enter a valid card number (without spaces or dashes).
msg17	Incorrect validation by the merchant
msg18	Domain initialization error

6. Language directory

LANGUAGE	CODE	ISO 639-1
Spanish	1	EN
English	2	EN
Catalan	3	CA
French	4	FR
German	5	DE
Dutch	6	NL
Italian	7	IT
Swedish	8	SV
Portuguese	9	PT
Valencian	10	VA
Polish	11	PL
Galician	12	GL
Basque	13	EU
Bulgarian	100	BG
Chinese	156	ZH
Croatian	191	HR
Czech	203	CS
Danish	208	DA
Estonian	233	ET
Finnish	246	FI
Greek	300	EL
Hungarian	348	HU
Indian	356	HI
Japanese	392	JA
Korean	410	KO
Latvian	428	LV
Lithuanian	440	LT
Maltese	470	MV
Romanian	642	RO
Russian	643	RU

Arabic	682	AR
Slovak	703	SK
Slovenian	705	SL
Turkish	792	TR

7. Transaction submission after transaction ID generation

Once the transaction ID has been received and stored by the merchant as described in the previous sections, you can launch the authorization transaction using any of the interfaces available on the Virtual POS.

In the authorization operation, the DS_MERCHANT_IDOPER parameter will have to be sent instead of the usual card data sending fields. In addition, the same order number (DS_MERCHANT_ORDER) must be used as the one used in the generation of the idOper.

Libraries that simplify this connection in Java and PHP languages are made available to merchants. Its download is available in the download section of the Redsys developer's website:

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

The library download includes help documentation for its use.

7.1 Implementation without use of help libraries

If you do not want to use the help libraries or you want to implement it for other programming languages, you can implement the REST call to the Virtual POS. **For more information**, we recommend consulting the Integration via REST documentation.

The authorisation request is made through a request to the Virtual POS. In this request you must include the following parameters:

- 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 Input and output parameters).
- 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

These parameters must be sent to the following endpoints, depending on whether you want to make a request in the test environment or actual operations:

URL Connection	Environment
https://sis-t.redsys.es:25443/sis/rest/trataPeticionREST	Tests
https://sis.redsys.es/sis/rest/trataPeticionREST	Real

Once the request has been processed, the Virtual POS will inform the merchant's server of the result of the request, with the result information included in a JSON file. The following fields will be included in it:

- 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.
- 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 modified and that the actual origin is the Virtual POS.**

The following describes the data of the Ds_MerchantParameters to be included to send an authorization request to the REST Service:


```
{
  "DS_MERCHANT_ORDER":1552572812,
"DS_MERCHANT_MERCHANTCODE":"999008881",
"DS_MERCHANT_TERMINAL":"2",
"DS_MERCHANT_CURRENCY":"978",
"DS_MERCHANT_TRANSACTIONTYPE":"0",
  "DS_MERCHANT_AMOUNT":"1000",
  "DS_MERCHANT_IDOPER":" 455097a74c21b761be86acb26c32609dce222e66",
}
```


In response you will obtain:


```
{
  "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":"454881******0004",
"Ds_Card_Type":"C",
"Ds_MerchantData":"",
"Ds_Card_Country":"724",
  "Ds_Card_Brand":"1"
}
```

8. Identification of inSite operations in the Administration Portal of the Virtual POS

In the Virtual POS administration portal, you can identify the operations conducted with inSite by consulting the "entry" field of the operations request.

The operations will be recorded with the input "inSite REST"

9. 3DSecure authentication in InSite

Merchants using the inSite connection have the ability to include the 3DSecure (3DS) protocol to authenticate cardholders for an additional level of fraud protection.

Including 3DS authentication means redirecting the client's navigation to the bank/card issuer's authentication server so that it can request the necessary credentials. This step should be performed in a subsequent action to the collection of card data described in the previous sections.

To use 3DS authentication, the Virtual POS terminal must be configured by the bank to support 3D Secure authentication. Likewise, it could be necessary that due to the configuration of the Virtual POS this step is not only optional, but also required for the correct authorization of the operations (if in doubt, consult the 3DS configuration with your bank that provides the Virtual POS).

9.1 3DSecure v1.0.2

9.1.1 Request authorisation

The authorisation request is made through a request to the Virtual POS. In this request you must include the following parameters:

- 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 Input and output parameters).
- 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

These parameters must be sent to the following endpoints, depending on whether you want to make a request in the test environment or actual operations:

URL Connection	Environment
https://sis-t.redsys.es:25443/sis/rest/trataPeticionREST	Tests
https://sis.redsys.es/sis/rest/trataPeticionREST	Real

Once the request has been processed, the Virtual POS will inform the merchant's server of the result of the request, with the result information included in a JSON file. The following fields will be included in it:

- 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.
- 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 modified and that the actual origin is the Virtual POS.**

The following describes the data of the Ds_MerchantParameters to be included to send an authorization request to the REST Service:


```
{
  "DS_MERCHANT_ORDER":1552642885,
  "DS_MERCHANT_MERCHANTCODE":"999008881",
"DS_MERCHANT_TERMINAL":"2",
"DS_MERCHANT_CURRENCY":"978",
"DS_MERCHANT_TRANSACTIONTYPE":"0",
  "DS_MERCHANT_AMOUNT":"1000",
  "DS_MERCHANT_IDOPER":" 455097a74c21b761be86acb26c32609dce222e66",
  "DS_MERCHANT_EMV3DS":{
            "threeDSInfo":"AuthenticationData",
            "protocolVersion":"1.0.2",
                  "browserAcceptHeader":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
            ,application/json",
                  "browserUserAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
            (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36"
            }
}
```

In response, the following will be obtained:


```
{
  "Ds_Order":"1552642885",
  "Ds_MerchantCode":"999008881",
  "Ds_Terminal":"2",
  "Ds_Currency":"978",
  "Ds_Amount":"1000",
  "Ds_TransactionType":"0",
  "Ds_EMV3DS": {
            "threeDSInfo":"ChallengeRequest",
            "protocolVersion":"1.0.2",
            "acsURL": "https://sis.redsys.es/sis-simulador-web/authenticationRequest.jsp",
"PAReq":"eJxVUttygjAQ/RWG95KEooKzpkPVjj7QOpZ+QBp2KlYuDVDx77tRqS0zmdmzJ+zlnMBDXxycbzRN
XpUzV3jcdbDUVZaXHzP3LX26C90HCenOlC5eUXcGJSTYNOoDnTybuU1Rq7zPsFGlFmlU+mOfi4j7vrAfn3D
Ow9HYlbCJt/gl4dpKUifPBzZAqmn0TpWtBKW/HtfPMhgFYSiAXSEUaNYL+brcJstNnCy381X8nAK7pKFUBcp
5RUjnlZOhU5sO35XzZFSXlbAzD7rqytac5MQPgA0AOnOQu7atp4wdj0fPYNacGk9XBTBLAbvNtuls1FCpPs9k
so/7lzQ+Jftln6R09p88WcRHOjNg9gZkqkU5KOIIPhVi6kfAznlQhZ1BintPcNr0gqC2TeKBsszfDJAHhiw6yWgS
0hYDAuzrqkS6QbL+xkDOaEY73Cafr6zGuiXZ/lololEYWLnPjK2WkzhBJC7ILABm/2VXJ9n1GVD073n8AOa7w
W0=",
            "MD":"cd164a6d0b77c96f7ef476121acfa987a0edf602"
  }
}
```


9.1.2 Authentication execution

The merchant will set up a form that sends a POST to the URL of the acsURL parameter obtained in the response to the above authorisation request. This form sends 3 parameters necessary for authentication:

- *PaReq*, whose value is obtained from the PAReq parameter obtained in the response of the previous authorization request.
- *MD*, whose value is obtained from the MD parameter obtained in the response of the previous authorization request.
- *TermUrl*, which identifies the URL to which the Issuing entity will make a POST with the authentication result. Said form will send a *single* PARES parameter, which contains the result of the authentication and that must be collected by the merchant for later submission in the authorization confirmation request.

9.1.3 3DSecure 1.0 authorization confirmation after Challenge

The following describes the data that the Ds_MerchantParameters must include to send a 3DSecure 1.0 authorisation confirmation request to the REST Service:


```
{
  "DS_MERCHANT_ORDER":1552642885,
  "DS_MERCHANT_MERCHANTCODE":"999008881",
"DS_MERCHANT_TERMINAL":"2",
"DS_MERCHANT_CURRENCY":"978",
"DS_MERCHANT_TRANSACTIONTYPE":"0",
  "DS_MERCHANT_AMOUNT":"1000",
  "DS_MERCHANT_EMV3DS":{
            "threeDSInfo":"ChallengeResponse",
            "protocolVersion":"1.0.2",
            "PARes":"eJzFWNmSo0iyfecrymoeNVVsWqBNmWPBKlaJVcAbmwBJLALE9vWDlJVZWT3VNn3vw
 70yyRR4uDvu
 ESeOu8X2X0N+/dLFdZOVxctX9Dvy9UtchGWUFcnLV8vkvhFf//W6NdM6jhkjDu91/LpV4qbxk/hL .....",
"MD":"035535127d549298f11d7d2fc1b0d4e9300f93f1"
            }
}
```


In response, the final result of the operation will be obtained:


```
{
"Ds_Amount":"1000",
  "Ds_Currency":"978",
  "Ds_Order":"1552642885",
  "Ds_MerchantCode":"999008881",
"Ds_Terminal":"2",
"Ds_Response":"0000",
"Ds_AuthorisationCode":"694432",
"Ds_TransactionType":"0",
"Ds_SecurePayment":"1",
"Ds_Language":"1",
"Ds_CardNumber":"454881******0004",
"Ds_Card_Type":"C",
"Ds_MerchantData":"",
"Ds_Card_Country":"724",
  "Ds_Card_Brand":"1"
}
```

9.2 EMV3DS

9.2.1 Start Request

This request allows obtaining the type of 3D Secure authentication that can be performed, in addition to the URL of the 3DSMethod, if it exists.

The start of the request is made through a REST request to the Virtual POS. The request must include the following parameters:

- 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 Input and output parameters).
- 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

These parameters must be sent to the following endpoints, depending on whether you want to make a request in the test environment or actual operations:

URL Connection	Environment
https://sis-t.redsys.es:25443/sis/rest/iniciaPeticionREST	Tests
https://sis.redsys.es/sis/rest/iniciaPeticionREST	Real

Once the request has been processed, the Virtual POS will inform the merchant's server of the result of the request, with the result information included in a JSON file. The following fields will be included in it:

- 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.
- 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 modified and that the actual origin is the Virtual POS.**

The following describes the data that the Ds_MerchantParameters must include to send an Inicia request to the REST Service:


```
{
  "DS_MERCHANT_ORDER":"1552571678",
  "DS_MERCHANT_MERCHANTCODE":"999008881",
"DS_MERCHANT_TERMINAL":"2",
"DS_MERCHANT_CURRENCY":"978",
"DS_MERCHANT_TRANSACTIONTYPE":"0",
  "DS_MERCHANT_AMOUNT":"1000",
  "DS_MERCHANT_IDOPER":" 455097a74c21b761be86acb26c32609dce222e66",
  "DS_MERCHANT_EMV3DS": {"threeDSInfo":"CardData"}
}
```


In response, the following will be obtained:


```
{
"Ds_Order":"1552571678",
"Ds_MerchantCode":"999008881",
"Ds_Terminal":"2",
"Ds_TransactionType":"0",
"Ds_EMV3DS": {
"protocolVersion":"2.1.0",
"threeDSServerTransID":"8de84430-3336-4ff4-b18d-f073b546ccea",
"threeDSInfo":"CardConfiguration",
"threeDSMethodURL":https://sis.redsys.es/sis-simulador-web/threeDsMethod.jsp

  }
}
```


The **Ds_EMV3DS** parameter will consist of the following fields:

- protocolVersion: will always indicate the major version number allowed in the operation. The merchant will be responsible for using the version number for which it is prepared.
- threeDSServerTransID: EMV3DS transaction identifier.
- threeDSInfo: CardConfiguration.
- threeDSMethodURL: URL del 3DSMethod.

9.2.2 3DSMethod Execution

The 3DSMethod is a process that allows the issuing entity to capture the information of the device the holder is using. This information, together with the EMV3DS data, which are sent in the authorization, will be used by the entity to make a risk assessment of the transaction. Based on this, the issuer can determine if the transaction is dependable and therefore does not require the intervention of the holder to verify its authenticity (frictionless).

The data capture of the device is done through an iframe hidden in the client's browser, which will establish a connection directly with the issuing entity in a transparent way for the user. The merchant will receive a notification when the information capture is finished and in the next step, when making the authorization request to the Virtual POS, the merchant must send the threeDSCompInd parameter indicating the execution of the 3DSMethod.

Steps for the execution of 3DSMethod:

1. In the response received with the card configuration (iniciaPeticion) the following data are received to run the 3DSMethod:
    a. threeDSMethodURL: 3DSMethod url
    b. threeDSServerTransID: EMV3DS transaction identifier

    If threeDSMethodURL is not received in the response, the process ends. In the authorization threeDSCompInd = N must be sent

2. Build the JSON Object with the parameters:
    a. threeDSServerTransID: value received in the card request response
    b. threeDSMethodNotificationURL: url of the merchant to which the completion of 3DSMethod from the entity will be notified

3. Encode the previous JSON in Base64url encode

4. A hidden iframe must be included in the client's browser, and a threeDSMethodData **field** with the value of the previous json object must be sent, in an http post form to the url obtained in the initial threeDSMethodURL **query**

5. The issuing entity interacts with the browser to proceed to capture information. On completion it will send the **threeDSMethodData** field in the html browser iframe via http post to the **threeDSMethodNotificationURL** url (indicated in step 2) after which the 3DSMethod ends.

6. If the 3DSMethod has been completed in less than 10 seconds, threeDSCompInd **= Y** will be sent in the authorization. If it has not been completed within 10 seconds it must stop the wait and send the authorisation with**threeDSCompInd = N**

9.2.3 Request for authorization with EMV3DS data

The start of the request is made through a REST request to the Virtual POS. The request must include the following parameters:

- 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 Input and output parameters).
- 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

These parameters must be sent to the following endpoints, depending on whether you want to make a request in the test environment or actual operations:

URL Connection	Environment
https://sis-t.redsys.es:25443/sis/rest/trataPeticionREST	Tests
https://sis.redsys.es/sis/rest/trataPeticionREST	Real

Once the transaction is managed, the Virtual POS will report the merchant's server of the result of the transaction, with the information of the result included in a JSON file. The following fields will be included in it:

- 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.
- 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 modified and that the actual origin is the Virtual POS.**

The following describes the data the Ds_MerchantParameters must include to send an authorization request with EMV3DS authentication to the REST Service:

```
{
"DS_MERCHANT_ORDER":1552572812,
"DS_MERCHANT_MERCHANTCODE":"999008881",
"DS_MERCHANT_TERMINAL":"2",
"DS_MERCHANT_CURRENCY":"978",
"DS_MERCHANT_TRANSACTIONTYPE":"0",
  "DS_MERCHANT_AMOUNT":"1000",
  "DS_MERCHANT_IDOPER":" 455097a74c21b761be86acb26c32609dce222e66",
  "DS_MERCHANT_EMV3DS":
{
"threeDSInfo":"AuthenticationData",
"protocolVersion":"2.1.0",
"browserAcceptHeader":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8,application/json",
        "browserUserAgent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/71.0.3578.98 Safari/537.36",
        "browserJavaEnabled":"false",
"browserJavaScriptEnabled":"false",
        "browserLanguage":"ES-es",
        "browserColorDepth":"24",
        "browserScreenHeight":"1250",
        "browserScreenWidth":"1320",
        "browserTZ":"52",
        "threeDSServerTransID":"8de84430-3336-4ff4-b18d-f073b546ccea",
        "notificationURL":"https://comercio-inventado.es/recibe-respuesta-autenticacion",
        "threeDSCompInd":"Y"
}
}
```


In response you will obtain:

- If a Frictionless is done, the final result of the operation will be obtained directly:


```
{
  "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":"454881******0004",
"Ds_Card_Type":"C",
"Ds_MerchantData":"",
"Ds_Card_Country":"724",
  "Ds_Card_Brand":"1"
}
```


- If not, the execution of a Challenge will be requested:


```
{
  "Ds_Amount":"1000",
  "Ds_Currency":"978",
  "Ds_Order":"1552572812",
  "Ds_MerchantCode":"999008881",
  "Ds_Terminal":"2",
  "Ds_TransactionType":"0",
  "Ds_EMV3DS":{
        "threeDSInfo":"ChallengeRequest",
        "protocolVersion":"2.1.0",
         "acsURL":"https://sis.redsys.es/sis-simulador-web/authenticationRequest.jsp",
```

"creq":"eyJ0aHJlZURTU2VydmVyVHJhbnNJRCI6IjhkZTg0NDMwLTMzMzYtNGZmNC1iMThkLWYwNzNiNTQ2Y2NlYSIsImFjc1RyYW5zSUQiOiJkYjVjOTljNC1hMmZkLTQ3ZWUtOTI2Zi1mYTBiMDk0MzUyYTAiLCJtZXNzYWdlVHlwZSI6IkNSZXEiLCJtZXNzYWdlVmVyc2lvbiI6IjIuMS4wIiwiY2hhbGxlbmdlV2luZG93U2l6ZSI6IjA1In0"
}
}

9.2.4 Challenge Execution

We describe this process in 3 steps:

Step 1.- Connection from the merchant to the ACS of the issuing bank

The next step consists in connecting from the merchant with the issuing entity so that the client may authenticate. This connection is made by sending an http POST form to the bank ACS url. For this connection we use the data received in the parameter Ds_EMV3DS of the previous step (acsURL and creq parameters):

"Ds_EMV3DS": {"threeDSInfo":"ChallengeRequest",
"protocolVersion":"2.1.0",
"acsURL":"https://sis.redsys.es/sis-simulador-web/authenticationRequest.jsp",
"creq":"eyJ0aHJlZURTU2VydmVyVHJhbnNJRCI6ImU5OWMzYzI2LTFiZWItNGY4NS05ZmE3LTI3OTJiZjE5NDZlMiIsImFjc1RyYW5zSUQiOiIyMTQzNDFhYi0wMjlhLTRmMGEtOTEyNi1iMDFkYmE5OTc2MTkiLCJtZXNzYWdlVHlwZSI6IkNSZXEiLCJtZXNzYWdlVmVyc2lvbiI6IjIuMS4wIiwiY2hhbGxlbmdlV2luZG93U2l6ZSI6IjA1In0"}

Example:

```
<form action="{acsURL}" method="POST" enctype = "application/x-www-form-urlencoded">
        <input type="hidden" name="creq" value="{creq}" ">
</form>
```

With the data received in Ds_EMV3DS it would be:

```
<form action="https://sis.redsys.es/sis-simulador-web/authenticationRequest.jsp" method="POST" enctype =
"application/x-www-form-urlencoded">
<input type="hidden" name="creq"
value="eyJ0aHJlZURTU2VydmVyVHJhbnNJRCI6ImU5OWMzYzI2LTFiZWItNGY4NS05ZmE3LTI3OTJiZjE5NDZlMiIsImFjc1RyYW5zSUQiOiIyMTQzNDFhYi0wMjlhLTRmMGEtOTEyNi1iMDFkYmE5OTc2MTkiLCJtZXNzYWdlVHlwZSI6IkNSZXEiLCJtZXNzYWdlVmVyc2lvbiI6IjIuMS4wIiwiY2hhbGxlbmdlV2luZG93U2l6ZSI6IjA1In0">
</form>
```

Step 2.-Challenge Execution

The holder is authenticated by the methods required by his/her issuing entity: OTP, static password, biometrics, etc.

Step 3.- Receipt of the authentication result

Once the challenge is over, the issuing entity will send the result to the merchant, making an http POST to the url of *the* notificationURL parameter that the merchant previously has sent in the authorization request:

"notificationURL":" https://comercio-inventado.es/recibe-respuesta-autenticacion"

The merchant will receive the "cres" parameter that will be used in the final authorization request that we see in the following section.

9.2.5 EMV3DS authorization confirmation after Challenge

The following describes the data of the Ds_MerchantParameters to be included to send an EMV3DS authorisation confirmation request to the REST Service:


```
{
  "DS_MERCHANT_ORDER":1552577128,
"DS_MERCHANT_MERCHANTCODE":"999008881",
"DS_MERCHANT_TERMINAL":"2",
"DS_MERCHANT_CURRENCY":"978",
"DS_MERCHANT_TRANSACTIONTYPE":"0",
"DS_MERCHANT_AMOUNT":"1000",
"DS_MERCHANT_PAN":" XXXXXXXXXXXXXXXXX ",
"DS_MERCHANT_EXPIRYDATE":"XXXX",
"DS_MERCHANT_CVV2":"XXX",
  "DS_MERCHANT_EMV3DS":{
            "threeDSInfo":"ChallengeResponse",
            "protocolVersion":"2.1.0",
            "cres":"eyJ0aHJlZURTU2VydmVyVHJhbnNJRCI6IjhkZTg0NDMwLTMzMzYtNGZmNC1iMThkLWY
            wNzNiNTQ2Y2NlYSIsImFjc1RyYW5zSUQiOiJkYjVjOTljNC1hMmZkLTQ3ZWUtOTI2Zi1mYTBiMDk
            0MzUyYTAiLCJtZXNzYWdlVHlwZSI6IkNSZXMiLCJtZXNzYWdlVmVyc2lvbiI6IjIuMS4wIiwidHJhbnN
            TdGF0dXMiOiJZIn0="
            }
}
```


In response, the final result of the operation will be obtained:


```
{
"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":"454881******0004",
"Ds_Card_Type":"C",
"Ds_MerchantData":"",
"Ds_Card_Country":"724",
  "Ds_Card_Brand":"1"
}
```


NOTE: For more details on the EMV3DS 2.0 protocol, you should consult the Virtual POS Integration via REST guide.