



COLLECTING SOLUTION

Quick start

Implementation guide

Document version 1.6

Contents

1. DOCUMENT HISTORY.....	3
2. OBTAINING HELP.....	4
3. ESTABLISHING INTERACTION WITH THE PAYMENT GATEWAY.....	5
3.1. Setting up the payment page URL.....	5
3.2. Identifying yourself when exchanging with the payment gateway.....	5
3.3. Managing the interaction with the merchant website.....	8
3.4. Managing security.....	10
3.5. Setting up the Instant Payment Notification.....	12
4. SEND AN HTML PAYMENT FORM VIA POST.....	14
5. COMPUTING THE SIGNATURE.....	18
5.1. Example of implementation with JAVA.....	20
5.2. Example of implementation with PHP.....	23
6. ANALYZING THE PAYMENT RESULT.....	24
6.1. Retrieving data returned in the response.....	24
6.2. Computing the IPN signature.....	25
6.3. Comparing signatures.....	25
6.4. Processing the response data.....	26
7. RETURNING TO THE SHOP.....	33
8. PROCEEDING TO TESTING.....	34
8.1. Making payment tests.....	34
8.2. Testing the IPN.....	34

1. DOCUMENT HISTORY

Version	Author	Date	Comment
1.6	Lyra Collect	17/06/2019	The hashing algorithm is now available via Settings > Shop, Keys tab.
1.5	Lyra Collect	23/01/2019	<ul style="list-style-type: none">• Update of the chapter Identifying yourself when exchanging with the payment gateway• “Certificate” replaced with “Key” in all menus
1.4	Lyra Collect	01/10/2018	Initial version

This document and its contents are confidential. It is not legally binding. No part of this document may be reproduced and/or forwarded in whole or in part to a third party without the prior written consent of Lyra Collect. All rights reserved.

2. OBTAINING HELP

Need some help? Please, check the FAQ on our website

<https://lyra.com/doc/fr/collect/faq/sitemap.html>

For technical inquiries or support, you can reach us from Monday to Friday, between 9 a.m. and 6 p.m.

by phone at:

0811900475

Service fee 0.06 € / min
+ call charge

by e-mail:

support-ecommerce@lyra-collect.com

and via your Expert Back Office, menu **Help > Contact support**

3. ESTABLISHING INTERACTION WITH THE PAYMENT GATEWAY

The merchant website and the payment gateway interact by exchanging data.

To create a payment, this data is sent in an HTML form via the buyer's browser.

At the end of the payment, the result is transmitted to the merchant website in two ways:

- by the browser when the buyer clicks the button to return to the merchant website.
- automatically by means of a notification called Instant Notification URL (also called IPN), see chapter **Setting up the end of payment notification**.

To guarantee the security of the exchange, the data is signed with a key only known to the merchant and the payment gateway.

3.1. Setting up the payment page URL

The merchant website interacts with the payment gateway by redirecting the buyer to the following URL:

<https://secure.lyra.com/vads-payment/>

3.2. Identifying yourself when exchanging with the payment gateway

To be able to interact with the payment gateway, the merchant needs to have:

- **The shop ID:** allows to identify the merchant website during the exchange. Its value is transmitted in the **vads_site_id** field.
- **The key:** allows to compute the alphanumeric signature transmitted in the **signature** field.

To retrieve these values:

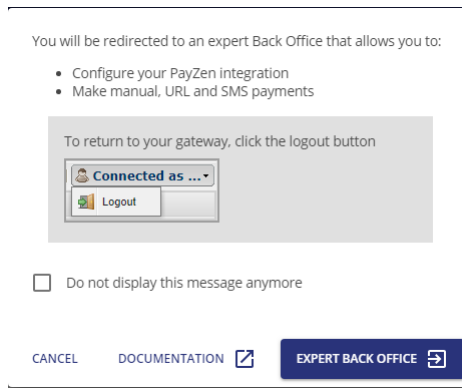
1. Sign in to the **Lyra Collect Back Office**: <https://secure.lyra.com/portal/>
2. Enter your login.
3. Enter your password.
4. Click **Login**.

In case of an entry error of the login and/or password, the error message *"Invalid username or password"* will appear.

You can correct your entry or click on the link **Forgotten password or locked account**.

5. Click **Other actions**.

The following window appears:



6. Click on **Expert Back Office** to access your Expert Back Office

7. Click **Settings > Shop**.

8. Select the **Keys** tab.

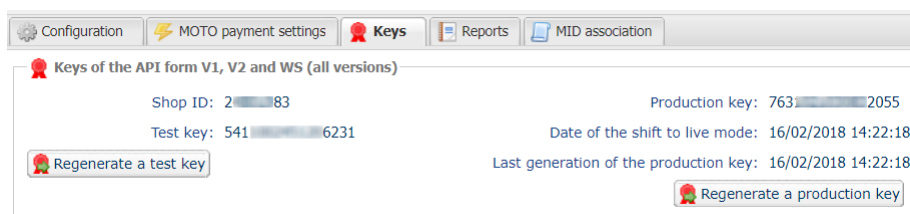


Figure 1: Keys tab

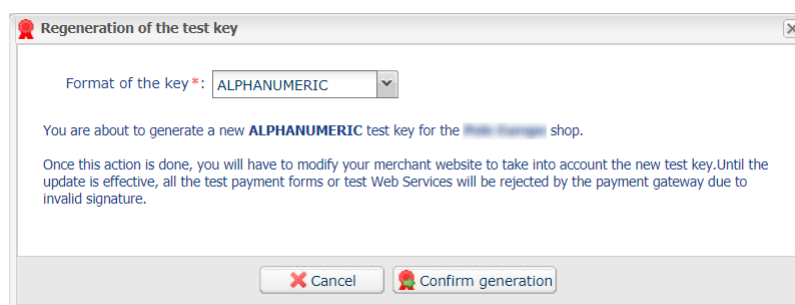
Two types of keys are available:

- The **test key** that allows to generate the form signature in test mode.
- The **production key** that allows to generate the form signature in production mode.


These keys can be numeric or alphanumeric.

For maximum security, it is recommended to use an alphanumeric key.

To change the format of your test key, click the **Regenerate a test key** button and select the format ("ALPHANUMERIC" or "NUMERIC").




To change the format of your production key, click the **Regenerate a production key** button and select the format ("ALPHANUMERIC" or "NUMERIC").

 **Regeneration of the production key** ✕

Format of the key*:

MUST READ BEFORE CONFIRMING

The type of your current key is numeric.
You are about to generate a new **ALPHANUMERIC** production key for the  shop.

- Together with your integrator, make sure that your merchant website supports this type of key.
- If you are using a payment module provided by the gateway for open source solutions such as Prestashop, Magento, WooCommerce, etc., see the module technical documentation, where the support of alphanumeric key must be specified in the "release notes" section.

Once this action is completed, you will have to modify your merchant website to take into account the new production key. As long as the update has not been made, all payment forms or production Web Services will be rejected by the payment gateway for invalid signature.

I confirm that I am aware of the risks and accept them

3.3. Managing the interaction with the merchant website

Two types of URL are used to manage dialog with the merchant website:

- **Instant Payment Notification**, also called the IPN,
- **Return URL** to the merchant website.

Instant Payment Notification - IPN

The payment gateway automatically informs the merchant website about the payment result. The data is sent via **POST**.

The gateway is able to contact the websites, regardless of the protocol (http or https) that was used.

Notifications are sent from an IP address in the **194.50.38.0/24** range in Test and Production mode.

To process these notifications, the merchant must **create a page** on their website that:

- analyzes the data received via **POST**,
- verifies the integrity of the received information by computing the signature,
- makes sure that the notification is not a duplicate notification (e.g. notification returned via the Expert Back Office),
- triggers an update of their database (order status, stock, etc.),
- sends e-mails to the buyer (invoice, order tracking, etc.).

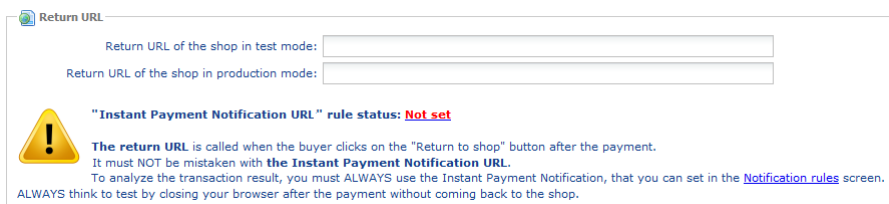
The processing time has a direct influence on the time it takes to display the payment summary page. The longer the processing takes, the more the summary display will be delayed.

To receive notifications, the merchant must **set up** the notification rules in his Expert Back Office (see chapter **Setting up notifications**).

In case of an issue during interaction with the merchant website, the payment gateway sends an e-mail to the shop administrator stating the reason of the error (HTTP error, etc.) and the instructions for returning the notification via the Expert Back Office.

Return URL to the merchant website

In the Expert Back Office, the merchant can configure the "default" return URLs via the menu **Settings > Shop > Configuration** tab:



Return URL

Return URL of the shop in test mode:

Return URL of the shop in production mode:

! "Instant Payment Notification URL" rule status: **Not set**

The return URL is called when the buyer clicks on the "Return to shop" button after the payment. It must NOT be mistaken with the **Instant Payment Notification URL**. To analyze the transaction result, you must ALWAYS use the Instant Payment Notification, that you can set in the [Notification rules](#) screen. ALWAYS think to test by closing your browser after the payment without coming back to the shop.

Figure 2: Setting up return URLs

The merchant can set up a different return URL for each mode.

By default, the buyer is redirected to the URL regardless of the payment result.

If no URL has been set up, the main URL of the shop will be used for redirection (**URL** parameter defined in the **Details** section of the shop).

The merchant will be able to override this setting in his/her payment form (see chapter **Setting up return URLs**).

Note:

The status of the "Instant Payment Notification at the End of Payment" (IPN) rule is displayed in this window. If the URL has not been set up, make sure to specify it (see chapter **Setting up notifications**).

3.4. Managing security

There are several ways to guarantee the security of online payments.

Ensuring interaction integrity

The integrity of exchanged information is preserved by the exchange of alphanumeric signatures between the payment platform and the merchant website.

The payment gateway and the merchant website interact via HTML forms.

A form contains a list of specific fields (see chapter **Generating a payment form**) used to generate a chain.

This chain is then converted to a smaller chain using a hash function (SHA-1, HMAC-SHA-256).

*The merchant will be able to choose the hash algorithm in their Expert Back Office (see chapter **Choosing the hash algorithm**).*

The resulting chain is referred to as the **digest** (*empreinte* in French) of the initial chain.

The digest must be transmitted in the **signature** field (see chapter **Computing the signature**).

Modeling security mechanisms:

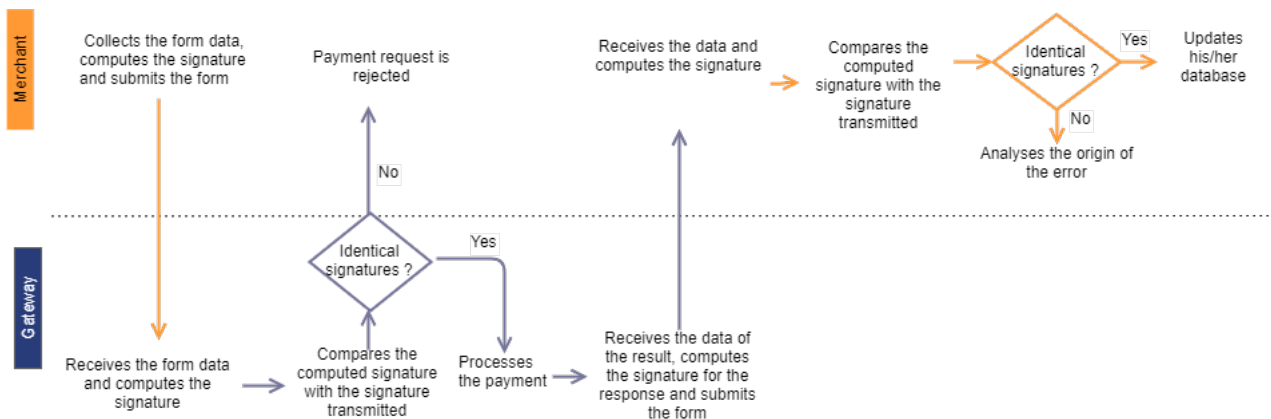


Figure 3: Diagram of a security mechanism

1. The merchant website builds the form data and computes the signature
2. The merchant website submits the form to the gateway
3. The gateway receives the form data and computes the signature
4. The gateway compares the computed signature with the signature that was transmitted by the merchant website
5. If the signatures are different, the payment request is rejected
If not, the gateway proceeds to payment
6. The gateway builds the result data and computes the response signature
7. Depending on the shop configuration (see chapter **Setting up notifications**), the payment gateway transmits the payment result to the merchant website.
8. The merchant website receives the data and computes the signature. It compares the computed signature with the signature that was transmitted by the payment gateway
9. If the signatures are different, the merchant analyses the source of the error (computation error, attempted fraud, etc.)

If not, the merchant proceeds to update their database (stock status, order status, etc.)

Selecting the hash algorithm

In the Expert Back Office, (**Settings > Shop > Keys**), the merchant can choose the hash function to use for generating signatures.



Security of data exchange

Signature algorithm in Test mode *:

SHA-1

Signature algorithm in Production mode *:

HMAC-SHA-256

SHA-1

HMAC-SHA-256 signature algorithm is applied by default.

Important

You can select a different signature algorithm for TEST mode and for PRODUCTION mode.

However, be sure to use the same method to generate your payment forms and to analyze the data transmitted by the gateway during notifications.

In order to facilitate changing the algorithm, the SHA-1 or HMAC-SHA-256 signatures will be accepted without generating rejections due to signature error for 24h.

Storing the production key

For security reasons, the production key will be masked after the first real payment made with a real card.

It is strongly recommended to store the key in a safe place (encrypted file, database etc.).

In case of losing the key, the merchant will be able to regenerate a new one via their Expert Back Office.

Remember that the production key can be viewed in the Expert Back Office via **Settings > Shop > Keys** tab.

Managing sensitive data

Online payment transactions are regulated by strict rules (PCI-DSS certification).

As a merchant, you have to make sure to never openly transcribe data that could resemble a credit card number. Your form will be rejected (code 999 - Sensitive data detected).

Special attention should be paid to order numbers containing between 13 and 16 numeric characters and beginning with 3, 4 or 5.

3.5. Setting up the Instant Payment Notification

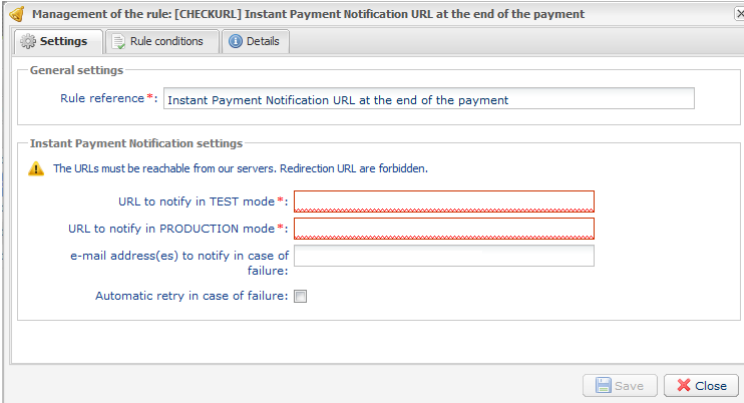
This notification is required to communicate the result of a payment request.

In your Expert Back Office, you must set up a URL that will be systematically called after a payment. It will inform the merchant website of the payment result even if your client has not clicked on return to the shop.

This parameter is called Instant Payment Notification URL at the end of payment.

To set up this notification:

1. Right-click the **Instant Payment Notification URL at the end of payment** line.
2. Select **Enable the rule**.
3. Make a right click on **Instant Payment Notification URL at the end of payment** once again.
4. Select **Manage the rule**.
5. Enter the URL of your page in the fields **URL to call in TEST mode** and **URL to call in PRODUCTION mode**.



The screenshot shows a dialog box titled "Management of the rule: [CHECKURL] Instant Payment Notification URL at the end of the payment". It has three tabs: "Settings", "Rule conditions", and "Details". The "Settings" tab is active. Under "General settings", there is a "Rule reference" field containing "Instant Payment Notification URL at the end of the payment". Under "Instant Payment Notification settings", there is a warning icon and text: "The URLs must be reachable from our servers. Redirection URL are forbidden." Below this, there are four input fields: "URL to notify in TEST mode *:", "URL to notify in PRODUCTION mode *:", "e-mail address(es) to notify in case of failure:", and "Automatic retry in case of failure:" with a checkbox. At the bottom right, there are "Save" and "Close" buttons.

6. Enter the **E-mail address(es) to notify in case of failure**.
7. To specify several e-mail addresses, separate them with a semi-colon.
8. Set up the parameters for **Automatic retry in case of failure**.
This option allows to automatically send notifications to the merchant website in case of failure (up to 4 times).
For more information, see chapter **Activating the automatic retry**
9. Save changes.

Other cases of notification

Depending on the subscribed commercial options, the payment gateway will make a call to the notification URL in the following cases :

- abort or cancellation by the buyer on the payment page
- refund from the Expert Back Office
- cancellation of a transaction from the Expert Back Office
- validation of a transaction from the Expert Back Office
- modification of a transaction from the Expert Back Office

4. SEND AN HTML PAYMENT FORM VIA POST.

The merchant website redirects the buyer to the payment gateway using a POST form from HTML to HTTPS.

This form contains :

The following technical elements:

- The `<form>` and `</form>` tags that allow to create an HTML form.
- The `method="POST"` attribute that defines the method used for sending data.
- The `action="https://secure.lyra.com/vads-payment/"` attribute that defines where to send the form data.

Form data:

All the data in the form must be encoded in **UTF-8**.

Special characters (accents, punctuation marks, etc.) will then be correctly interpreted by the payment gateway. Otherwise, the signature will not be computed correctly and the form will be rejected.

Please, consult the table below that indicates required formats.

Notation	Description
a	Alphabetic characters (from 'A' to 'Z' and from 'a' to 'z')
n	Numeric characters
s	Special characters
an	Alphanumeric characters
ans	Alphanumeric and special characters (except '<' and '>')
3	Fixed length of 3 characters
..12	Variable length up to 12 characters
json	JavaScript Object Notation. Object containing key/value pairs separated by commas. It starts with a left brace '{' and ends with a right brace '}'. Each key / value pair contains the name of the key between double-quotes followed by " : ", followed by a value. The name of the key must be alphanumeric. The value can be: <ul style="list-style-type: none">• a chain of characters (in this case it must be framed by double-quotes)• a number• an object• a table• a boolean• empty Example: {"name1":45,"name2":"value2", "name3"=false}
enum	Characterizes a field with a complete list of values. The list of possible values is given in the field definition.
Enum list	List of values separated by a " ; ". The list of possible values is given in the field definition. Example: vads_payment_cards=VISA;MASTERCARD
map	List of key / value pair separated by a " ; ". Each key / value pair contains the name of the key followed by "=", followed by a value. The value can be: <ul style="list-style-type: none">• a chain of characters• a boolean• a json object

Notation	Description
	<ul style="list-style-type: none"> an xml object <p>The list of possible values for each key/value pair is provided in the field definition. Example: <code>vads_theme_config=SIMPLIFIED_DISPLAY=true;RESPONSIVE_MODEL=Model_1</code></p>

- Mandatory fields :

Field name	Description	Format	Value
signature	Signature guaranteeing the request's integrity exchanged between the merchant website and the payment gateway.	ans	Ex : ycA5Do5tNvsnkdc/eP1bj2xa19z9q3iWPy9/rpesfS0=
vads_action_mode	Card data acquisition mode	enum	INTERACTIVE
vads_amount	Payment amount in the smallest currency unit (cents for euro).	n..12	E.g.: 3000 for 30,00 EUR
vads_ctx_mode	Defines the mode of interaction with the payment gateway.	enum	TEST or PRODUCTION
vads_currency	Numeric currency code to be used for the payment, in compliance with the ISO 4217 standard (numeric code).	n3	E.g.: 978 for euro (EUR)
vads_page_action	Action to perform	enum	PAYMENT
vads_payment_config	Payment type	enum	SINGLE for immediate payment
vads_site_id	Shop ID	n8	E.g.: 12345678
vads_trans_date	Date and time of the payment form in UTC format	n14	Respect the YYYYMMDDhhmmss format E.g.: 20170701130025
vads_trans_id	Transaction number	n6	E.g.: 123456
vads_version	Version of the exchange protocol with the payment gateway	enum	V2

Table 1: List of mandatories fields

- Recommended fields:

- Order details

Field name	Description	Format	Value
vads_order_id	Order ID	ans..64	E.g.: 2-XQ001
vads_order_info	Complementary order info	an..255	E.g.: Door phone code 3125
vads_order_info2	Complementary order info	an..255	E.g.: No elevator
vads_order_info3	Complementary order info	an..255	E.g.: Express
vads_nb_products	Number of items in the cart	n..12	E.g.: 2
vads_product_ext_idN	Product bar code on the merchant website. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	an..100	E.g.: vads_product_ext_id0 = "0123654789123654789" vads_product_ext_id1 = "0223654789123654789" vads_product_ext_id2 = "0323654789123654789"
vads_product_labelN	Item name. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	an..255	E.g.: vads_product_label0 = "tee-shirt" vads_product_label1 = "Biscuit" vads_product_label2 = "sandwich"
vads_product_amountN	Item amount. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	n..12	E.g.: vads_product_amount0 = "1200" vads_product_amount1 = "800" vads_product_amount2 = "950"
vads_product_typeN	Item type. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	enum	E.g.: vads_product_type0 = "CLOTHING_AND_ACCESSORIES"

Field name	Description	Format	Value
			vads_product_type1 = "FOOD_AND_GROCERY" vads_product_type2 = "FOOD_AND_GROCERY"
vads_product_refN	Item reference. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	an..64	E.g.: vads_product_ref0 = "CAA-25-006" vads_product_ref1 = "FAG-B5-112" vads_product_ref2 = "FAG-S9-650"
vads_product_qtyN	Quantity of items. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	n..12	E.g.: vads_product_qty0 = "1" vads_product_qty1 = "2" vads_product_qty2 = "2"

- Buyer details

Field name	Description
vads_cust_email	Buyer's e-mail address.
vads_cust_id	Buyer reference on the merchant website.
vads_cust_title	Buyer's title.
vads_cust_status	Status (PRIVATE : for private clients / COMPANY for companies)
vads_cust_first_name	First name.
vads_cust_last_name	Last name
vads_cust_legal_name	Buyer's legal name.
vads_cust_cell_phone	Cell phone number
vads_cust_phone	Phone number.
vads_cust_address_number	Street number.
vads_cust_address	Postal address.
vads_cust_district	District.
vads_cust_zip	Zip code.
vads_cust_city	City.
vads_cust_state	State / Region.
vads_cust_country	Country code according to the ISO 3166 standard.

Table 2: Parameter list - Buyer details

- Shipping details

Field name	Description	Format	Value
vads_ship_to_city	City	an..128	E.g.: Bordeaux
vads_ship_to_country	Country code in compliance with the ISO 3166 standard.	a2	E.g.: FR
vads_ship_to_district	District	ans..127	E.g.: La Bastide
vads_ship_to_first_name	First name	ans..63	E.g.: Albert
vads_ship_to_last_name	Name	ans..63	E.g.: Durant
vads_ship_to_legal_name	Legal name	an..100	E.g.: D. & Cie
vads_ship_to_phone_num	Phone number	ans..32	E.g.: 0460030288
vads_ship_to_state	State / Region	ans..127	E.g.: Nouvelle Aquitaine
vads_ship_to_status	Allows to specify the type of the shipping address.	enum	PRIVATE : for shipping to a private individual COMPANY : for shipping to a company
vads_ship_to_street_number	Street number	ans..64	E.g.: 2
vads_ship_to_street	Postal address	ans..255	E.g.: Rue Sainte Catherine
vads_ship_to_street2	Second line of the address	ans..255	
vads_ship_to_zip	Zip code	an..64	E.g.: 33000

- Optional fields :

The **Pay** button that will allow to send data:


```
<input type="submit" name="pay" value="Pay"/>
```

5. COMPUTING THE SIGNATURE

To be able to compute the signature, you must have:

- all the fields that start with **vads_**
- the signature algorithm chosen in the shop configuration
- the **key**

The value of the key is available in your Expert Back Office via **Settings > Shop > Keys** tab.

The signature algorithm is defined in your Expert Back Office via **Settings > Shop > Configuration** tab.

For maximum security, it is recommended to use HMAC-SHA-256 algorithm and an alphanumeric key.

To compute the signature:

1. Sort the fields that start with **vads_** alphabetically.
2. Make sure that all the fields are encoded in UTF-8.
3. Concatenate the values of these fields separating them with the "+" character.
4. Concatenate the result with the test or production key separating them with a "+".
5. According to the signature algorithm defined in your shop configuration:
 - a. if your shop is configured to use "SHA-1", apply the **SHA-1** hash function to the chain obtained during the previous step.
 - b. if your shop is configured to use "HMAC-SHA-256", compute and encode in Base64 format the message signature using the **HMAC-SHA-256** algorithm with the following parameters:
 - the SHA-256 hash function,
 - the test or production key (depending on the value of the **vads_ctx_mode** field) as a shared key,
 - the result of the previous step as the message to authenticate.
6. Save the result of the previous step in the **signature** field.

Example of parameters sent to the payment gateway:

```
<form method="POST" action="https://secure.lyra.com/vads-payment/">
<input type="hidden" name="vads_action_mode" value="INTERACTIVE" />
<input type="hidden" name="vads_amount" value="5124" />
<input type="hidden" name="vads_ctx_mode" value="TEST" />
<input type="hidden" name="vads_currency" value="978" />
<input type="hidden" name="vads_page_action" value="PAYMENT" />
<input type="hidden" name="vads_payment_config" value="SINGLE" />
<input type="hidden" name="vads_site_id" value="12345678" />
<input type="hidden" name="vads_trans_date" value="20170129130025" />
<input type="hidden" name="vads_trans_id" value="123456" />
<input type="hidden" name="vads_version" value="V2" />
<input type="hidden" name="signature" value="ycA5Do5tNvsnKdc/eP1bj2xa19z9q3iWPy9/rpesfS0= " />

<input type="submit" name="pay" value="Pay"/>
</form>
```

This sample form is analyzed as follows:

1. The fields whose names start with **vads_** are sorted **alphabetically**:

- vads_action_mode
- vads_amount
- vads_ctx_mode
- vads_currency
- vads_page_action
- vads_payment_config
- vads_site_id
- vads_trans_date
- vads_trans_id
- vads_version

2. The values of these fields are concatenated using the "+" character:

```
INTERACTIVE+5124+TEST+978+PAYMENT+SINGLE+12345678+20170129130025+123456+V2
```

3. The value of the test key is added at the end of the chain and separated with the "+" character. In this example, the test key is **1122334455667788**

```
INTERACTIVE+5124+TEST+978+PAYMENT+SINGLE+12345678+20170129130025+123456+V2+1122334455667788
```

4. If you use the SHA-1 algorithm, apply it to the obtained chain.

The result that must be transmitted in the signature field is:
59c96b34c74b9375c332b0b6a32e6deec87de2b

5. If your shop is configured to use "HMAC-SHA-256", compute and encode in Base64 format the message signature using the **HMAC-SHA-256** algorithm with the following parameters:

- the SHA-256 hash function,
- the test or production key (depending on the value of the **vads_ctx_mode** field) as a shared key,
- the result of the previous step as the message to authenticate.

The result that must be transmitted in the signature field is:

ycA5Do5tNvsnKdc/eP1bj2xa19z9q3iWPy9/rpesfS0=

5.1. Example of implementation with JAVA

Definition of the utility class SHA that will include the elements required to process the HMAC-SHA-256 algorithm

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.TreeMap;

public class VadsSignatureExample {
    /**
     * Build signature (HMAC SHA-256 version) from provided parameters and secret key.
     * Parameters are provided as a TreeMap (with sorted keys).
     */
    public static String buildSignature(TreeMap<String, String> formParameters, String
    secretKey) throws NoSuchAlgorithmException, InvalidKeyException, UnsupportedEncodingException
    {
        // Build message from parameters
        String message = String.join("+", formParameters.values());
        message += "+" + secretKey;
        // Sign
        return hmacSha256Base64(message, secretKey);
    }
    /**
     * Actual signing operation.
     */
    public static String hmacSha256Base64(String message, String secretKey) throws
    NoSuchAlgorithmException, InvalidKeyException, UnsupportedEncodingException {
        // Prepare hmac sha256 cipher algorithm with provided secretKey
        Mac hmacSha256;
        try {
            hmacSha256 = Mac.getInstance("HmacSHA256");
        } catch (NoSuchAlgorithmException nsae) {
            hmacSha256 = Mac.getInstance("HMAC-SHA-256");
        }
        SecretKeySpec secretKeySpec = new SecretKeySpec(secretKey.getBytes("UTF-8"), "HmacSHA256");
        hmacSha256.init(secretKeySpec);
        // Build and return signature
        return Base64.getEncoder().encodeToString(hmacSha256.doFinal(message.getBytes("UTF-8")));
    }
}
```

Definition of the utility class SHA that will include the elements required to process the SHA-1 algorithm

```
import java.security.MessageDigest;
import java.security.SecureRandom;

public class Sha {
    static public final String SEPARATOR = "+";
    public static String encode(String src) {
        try {
            MessageDigest md;
            md = MessageDigest.getInstance("SHA-1");
            byte bytes[] = src.getBytes("UTF-8");
            md.update(bytes, 0, bytes.length);
            byte[] shalhash = md.digest();
            return convertToHex(shalhash);
        }
        catch(Exception e){
            throw new RuntimeException(e);
        }
    }
    private static String convertToHex(byte[] shalhash) {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < shalhash.length; i++) {
            byte c = shalhash[i];
            addHex(builder, (c >> 4) & 0xf);
            addHex(builder, c & 0xf);
        }
        return builder.toString();
    }
    private static void addHex(StringBuilder builder, int c) {
        if (c < 10)
            builder.append((char) (c + '0'));
        else
            builder.append((char) (c + 'a' - 10));
    }
}
```

}

Function that computes the signature:

```
public ActionForward performCheck(ActionMapping actionMapping, Basivoiorm form,
    HttpServletRequest request, HttpServletResponse response){
    SortedSet<String> vadsFields = new TreeSet<String>();
    Enumeration<String> paramNames = request.getParameterNames();

    // retrieve and sort the fields starting with vads_* alphabetically
    while (paramNames.hasMoreElements()) {
        String paramName = paramNames.nextElement();
        if (paramName.startsWith( "vads_" )) {
            vadsFields.add(paramName);
        }
    }
    // Compute the signature
    String sep = Sha.SEPARATOR;
    StringBuilder sb = new StringBuilder();
    for (String vadsParamName : vadsFields) {
        String vadsParamValue = request.getParameter(vadsParamName);
        if (vadsParamValue != null) {
            sb.append(vadsParamValue);
        }
        sb.append(sep);
    }
    sb.append( shaKey );
    String c_sign = Sha.encode(sb.toString());
    return c_sign;}

```

5.2. Example of implementation with PHP

Example of a signature computation using the HMAC-SHA-256 algorithm:

```
function getSignature ($params,$key)
{
    /**
     *Function that computes the signature.
     * $params : table containing the fields to send in the payment form.
     * $key : TEST or PRODUCTION key
     */
    //Initialization of the variable that will contain the string to encrypt
    $contenu_signature = "";

    //sorting fields alphabetically
    ksort($params);
    foreach($params as $name=>$value){

        //Recovery of vads_ fields
        if (substr($nom,0,5)=='vads_'){

            //Concatenation with "+"
            $contenu_signature .= $value."+";

        }
    }
    //Adding the key at the end
    $contenu_signature .= $key;

    //Encoding base64 encoded chain with SHA-256 algorithm
    $signature = base64_encode(hash_hmac('sha256',$contenu_signature, $key, true));
    return $signature;
}
```

Example of a signature computation using the SHA-1 algorithm:

```
function getSignature($params, $key)
{
    /**
     * Function that computes the signature.
     * $params : table containing the fields to send in the payment form.
     * $key : TEST or PRODUCTION key
     */
    //Initialization of the variable that will contain the string to encrypt
    $contenu_signature = "" ;

    // Sorting fields alphabetically
    ksort($params);
    foreach ($params as $name =>$value)
    {
        // Recovery of vads_ fields
        if (substr($nom,0,5)=='vads_') {
            // Concatenation with "+"
            $contenu_signature .= $value."+";
        }
    }
    // Adding the key at the end
    $contenu_signature .= $key;

    // Applying SHA-1 algorithm
    $signature = sha1($contenu_signature);
    return $signature ;
}
```

6. ANALYZING THE PAYMENT RESULT

The Instant Payment Notification URL (IPN) allows the payment gateway to notify automatically the merchant website about the payment result.

The payment gateway is able to contact the merchant website regardless of the protocol (http or https) that was used.

6.1. Retrieving data returned in the response

The data returned in the response depends on the parameters sent in the payment form, the payment type and the settings of your shop. This data constitutes a field list. Each field contains a response value. The field list can be updated.

The data is always sent by the payment gateway using the **POST** method.

The first step consists in retrieving the contents received via the POST method.

Examples:

- In PHP, data is stored in the super global variable **\$_POST**,
- In ASP.NET (C#), you must use the **Form** property of the **HttpRequest** class.
- In Java, you must use the **getParameter** method of the **HttpServletRequest** interface.

The script will have to create a loop to retrieve all the transmitted fields.

Example of data sent during payment notification :

```
vads_amount = 3000
vads_auth_mode = FULL
vads_auth_number = 3fb0de
vads_auth_result = 00
vads_capture_delay = 0
vads_card_brand = VISA
vads_card_number = 497010XXXXXX0000
vads_payment_certificate = a50d15063b5ec6cb140043138b8d7576470b71a9
vads_ctx_mode = TEST
vads_currency = 978
vads_effective_amount = 3000
vads_site_id = 12345678
vads_trans_date = 20140902094139
vads_trans_id = 454058
vads_validation_mode = 0
vads_version = V2
vads_warranty_result = YES
vads_payment_src = EC
vads_sequence_number = 1
vads_contract_used = 5785350
vads_trans_status = AUTHORISED
vads_expiry_month = 6
vads_expiry_year = 2015
vads_bank_code = 17807
vads_bank_product = A
vads_pays_ip = FR
vads_presentation_date = 20140902094202
vads_effective_creation_date = 20140902094202
vads_operation_type = DEBIT
vads_threeds_enrolled = Y
vads_threeds_cavv = Q2F2dkNhdnZDYXZ2Q2F2dkNhdnY=
vads_threeds_eci = 05
vads_threeds_xid = WXJsVXpHVjFoMktzNmw5dTdlekQ=
vads_threeds_cavvAlgorithm = 2
vads_threeds_status = Y
vads_threeds_sign_valid = 1
vads_threeds_error_code =
vads_threeds_exit_status = 10
vads_trans_uuid= 1cd9994823334e31bbb579b4d716832d
vads_risk_control = CARD_FRAUD=OK;COMMERCIAL_CARD=OK
vads_result = 00
vads_extra_result = 00
```



```
vads_card_country = FR
vads_language = fr
vads_hash = 299d81f4b175bfb7583d904cd19ef5e38b2b79b2373d9b2b4aab74e5753b10bc
vads_url_check_src = PAY
vads_action_mode = INTERACTIVE
vads_payment_config = SINGLE
vads_page_action = PAYMENT
signature = FxGvazgW0dqgOrVrx6bqKZSXh2y5Dp3bWC9HFfn33t+Q=
```

6.2. Computing the IPN signature

The signature is computed by following the same procedure as for creating the payment form.

IMPORTANT

The data submitted by the payment gateway is encoded in UTF-8. Any alteration of received data will result in signature computation error.

You must compute the signature with the fields received in the notification and not the ones that you transmitted in your payment form.

To compute the signature:

1. Take all the fields whose name starts with **vads_**.
2. Sort these fields alphabetically.
3. Concatenate the values of these fields separating them with the "+" character.
4. Concatenate the result with the test or production key separating them with a "+".
5. According to the signature algorithm defined in your shop configuration:
 - a. if your shop is configured to use "SHA-1", apply the **SHA-1** hash function to the chain obtained during the previous step.
 - b. if your shop is configured to use "HMAC-SHA-256", compute and encode in Base64 format the message signature using the **HMAC-SHA-256** algorithm with the following parameters:
 - the SHA-256 hash function,
 - the test or production key (depending on the value of the **vads_ctx_mode** field) as a shared key,
 - the result of the previous step as the message to authenticate.

6.3. Comparing signatures

In order to make sure the response is complete, you must compare the value of the **signature** field received in the response with the one computed in the step "Computing the IPN signature".

IMPORTANT

You must not compare the IPN signature with the signature that you transmitted in your form.

If the signatures match,

- You may consider the response as safe and proceed with the analysis.
- If they do not, the script will have to throw an exception and warn the merchant .

The signatures may not match because of:

- an implementation error (error in your calculation, problem with UTF-8 encoding, etc.),
- an error in the value of the key or in the **vads_ctx_mode** (field value - frequent issue when going to live mode),
- a data corruption attempt.

6.4. Processing the response data

Here is an example of analysis to guide you through processing the response data.

1. Identify the mode (TEST or PRODUCTION) that was used for creating the transaction by analyzing the value of the **vads_ctx_mode** field.
2. Identify the order by retrieving the value of the **vads_order_id** field if you have transmitted it to the payment gateway.
Make sure that the order status has not already been updated.
3. Retrieve the payment result transmitted in the **vads_trans_status** field.
Its value allows you to define the order status.

Value	Description
ABANDONED	Abandoned payment abandoned by the buyer. The transaction has not been created, and therefore cannot be viewed in the Expert Back Office.
ACCEPTED	Accepted. Status of a VERIFICATION type transaction for which the authorization request or information request has been successfully completed. This status can not evolve. Transactions with "ACCEPTED" will never be captured.
AUTHORISED	Waiting for capture The transaction has been accepted and will be automatically captured at the bank on the expected date.
AUTHORISED_TO_VALIDATE	To be validated The transaction, created with manual validation, is authorized. The merchant must manually validate the transaction in order for it to be captured. The transaction can be validated as long as the expiration date of the authorization request has not passed. If the authorization validity period has passed, the payment status changes to EXPIRED . The Expired status is final.
CANCELLED	Canceled The transaction has been canceled by the merchant.
CAPTURED	Captured The transaction has been captured by the bank.
CAPTURE_FAILED	The transaction capture has failed. Contact the technical support.
EXPIRED	Expired The expiry date of the authorization request has passed and the merchant has not validated the transaction. The account of the cardholder will, therefore, not be debited.
INITIAL	Pending This status concerns all the payment methods that require integration via a payment form with redirection. This status is returned when: <ul style="list-style-type: none"> • no response has been returned by the acquirer or

Value	Description
	<ul style="list-style-type: none"> the delay of the response from the acquirer has exceeded the payment session on the payment gateway. This status is temporary. The final status will be displayed in the Expert Back Office immediately after the synchronization has been completed.
NOT_CREATED	Transaction not created The transaction has not been created, and therefore cannot be viewed in the Expert Back Office.
REFUSED	Declined Transaction is declined.
SUSPENDED	Suspended The capture of the transaction is temporarily blocked by the acquirer (AMEX GLOBAL or SECURE TRADING). Once the transaction has been correctly captured, its status changes to CAPTURED .
UNDER_VERIFICATION	For PayPal transactions, this value means that PayPal withholds the transaction for suspected fraud. For CB or PPRO transactions, this status indicates that a refund has been requested. Verification is being made in order to validate the refund. The payment will remain in the Transactions in progress tab until the verification process has been completed. The transaction will then take one of the following statuses: AUTHORISED or CANCELED . A notification will be sent to the merchant to warn them about the status change (Instant Payment Notification on batch change).
WAITING_AUTHORISATION	Waiting for authorization The capture delay exceeds the authorization validity period.
WAITING_AUTHORISATION_TO_VALIDATE	To be validated and authorized The capture delay in the bank exceeds the authorization validity period. An authorization of 1 EUR (or information request about the CB network if the acquirer supports it) has been accepted. The merchant must manually validate the transaction for the authorization request and the capture to occur.

Table 3: Values associated with the vads_trans_status field

- Retrieve the payment reference transmitted in the **vads_trans_id** field.
- Analyze the **vads_sequence_number** field to know the number of attempts that have been made to make the payment.

vads_payment_config = SINGLE:

vads_url_check_src	vads_sequence_number	Description
PAY	1	Payment made in 1 attempt
	2	Payment made in 2 attempts
	3	Payment made in 3 attempts
BATCH_AUTO	1	Deferred payment made in 1 attempt
	2	Deferred payment made in 2 attempts
	3	Deferred payment made in 3 attempts

Note

Installment payments are not compatible with the feature of additional attempts in case of a rejected payment.

6. Retrieve the value of the **vads_trans_date** field to identify the payment date.

7. Analyze the **vads_payment_option_code** field to determine whether it is an installment payment:

Value	Description
1	Payment in 1 installment
2	Payment in 2 installments
3	Payment in 3 installments
n	Payment in n installments

Table 4: vads_payment_option_code field analysis

8. Retrieve the value of the **vads_capture_delay** field to identify the number of days before the capture in the bank.

It will allow you to identify whether the payment is an immediate or a deferred payment.

9. Retrieve the used amount and currency. To do this, retrieve the values of the following fields:

Field name	Description
vads_amount	Payment amount in the smallest currency unit.
vads_currency	Code of the currency used for the payment.
vads_change_rate	Exchange rate used to calculate the effective payment amount (see vads_effective_amount).
vads_effective_amount	Payment amount in the currency used for the capture in the bank.
vads_effective_currency	Currency used for the capture in the bank.

Table 5: Analysis of the payment amount and currency

10. Retrieve the value of the **vads_auth_result** field to identify the result of the authorization request.

The complete list of returned codes can be viewed in the data dictionary.

Here is a list of frequently returned codes that can help you understand the reason of the rejection:

Value	Description
03	Invalid acceptor This code is sent by the card issuer. It refers to a configuration problem on authorization servers. (e.g. closed contract, incorrect MCC declared, etc.). To find out the specific reason of the rejection, the buyer must contact his or her bank.
05	Do not honor This code is sent by the card issuer. This code is used in the following cases: <ul style="list-style-type: none"> Invalid expiry date Invalid CVV Exceeded credit limit Insufficient funds (etc.) To find out the specific reason of the rejection, the buyer must contact his or her bank.
51	Insufficient balance or exceeded credit limit This code is sent by the card issuer. This code appears if the funds on the buyer's account are insufficient for making the purchase. To find out the specific reason of the rejection, the buyer must contact his or her bank.
56	Card absent from the file This code is sent by the card issuer. The entered card number is incorrect or the card number + expiration date combination does not exist.
57	Transaction not allowed for this cardholder This code is sent by the card issuer. This code is used in the following cases: <ul style="list-style-type: none"> The buyer attempts to make an online payment with a cash withdrawal card, The authorized payment limit is exceeded. To find out the specific reason of the rejection, the buyer must contact his or her bank.
59	Suspected fraud

Value	Description
	This code is sent by the card issuer. This code appears when an incorrect CVV code or expiration date has been entered several times. To find out the specific reason of the rejection, the buyer must contact his or her bank.
60	The acquirer of the card must contact the acquirer This code is sent by the card issuer. It refers to a configuration problem on authorization servers. It is used when the merchant ID does not correspond to the used sales channel (e.g.: an e-commerce transaction with a distant sales MID with manual entry of contract data). Contact the customer service to resolve the problem.

Table 6: Values associated with the vads_auth_result field

11. Retrieve the 3D Secure authentication result. To do this:

- a. Retrieve the value of the **vads_threeds_enrolled** field to identify the status of the card enrollment.

Value	Description
Empty	Incomplete 3DS authentication process (3DS disabled in the request, the merchant is not enrolled or the payment method is not eligible for 3DS).
Y	Authentication available, cardholder enrolled.
N	Cardholder not enrolled.
U	Impossible to identify the cardholder or authentication is not available for the card (e.g. commercial or prepaid cards).

Table 7: Values of the vads_threeds_enrolled field

- b. Retrieve the result of 3D Secure authentication by retrieving the value of the **vads_threeds_status** field.

Value	Description
Empty	Incomplete 3DS authentication (3DS disabled in the request, the cardholder is not enrolled or the payment method is not eligible for 3DS).
Y	Cardholder authentication success.
N	Cardholder authentication error.
U	Authentication impossible.
A	Authentication attempted but not completed.

Table 8: Values of the vads_threeds_status field

12. Retrieve the result of fraud checks by identifying the value of the **vads_risk_control** field. This field is sent only if the merchant has:

- subscribed to the "Risk management" service
- enabled at least one verification process in the Expert Back Office (**Settings > Risk management** menu).

It is populated with the list of values separated by ";" with the following syntax: **vads_risk_control = control1=result1;control2=result2**

the possible values for **control** are:

Value	Description
CARD_FRAUD	Checks whether the buyer's card number is on the card greylist.
SUSPECT_COUNTRY	Checks whether the issuing country of the buyer's card is on the list of forbidden countries.
IP_FRAUD	Checks whether the buyer's IP address is on the IP greylist.
CREDIT_LIMIT	Checks the purchase frequency and amounts for the same card number, or the maximum amount of an order.
BIN_FRAUD	Checks whether the BIN code of the card is on the BIN code greylist.
ECB	Checks whether the buyer's card is of "e-carte bleue" type.
COMMERCIAL_CARD	Checks whether the buyer's card is a commercial card.
SYSTEMATIC_AUTO	Checks whether the buyer's card is a MAESTRO or VISA ELECTRON card.

Value	Description
INCONSISTENT_COUNTRIES	Checks whether the country of the IP address, the country of the payment card and the buyer's country of residence match.
NON_WARRANTY_PAYMENT	Liability shift.
SUSPECT_IP_COUNTRY	Checks whether the buyer's country, identified by their IP address, is on the list of forbidden countries.

Table 9: List of fraud verification processes

The possible values for **result** are:

Value	Description
OK	OK.
WARNING	Informational control failed.
ERROR	Blocking control failed.

Table 10: List of fraud verification processes

13. Retrieve the type of the card used for the payment.

Two scenarios are possible:

- For a payment processed with **only one card**. The fields to process are:

Field name	Description
vads_card_brand	Brand of the card used for the payment, e.g.: CB, VISA, VISA_ELECTRON, MASTERCARD, MAESTRO, VPAY
vads_card_number	Card number used for the payment.
vads_expiry_month	Expiry month between 1 and 12 (e.g.: 3 for March, 10 for October).
vads_expiry_year	Expiry year in 4 digits (e.g.: 2023).
vads_bank_code	Code of the issuing bank
vads_bank_product	Product code of the card
vads_card_country	Country code of the country where the card was issued (alpha ISO 3166-2 code, e.g.: "FR" for France, "PF" for French Polynesia, "NC" for New Caledonia, "US" for the United States).

Table 11: Analysis of the card used for the payment

- For a **split payment** (i.e. a transaction using several payment methods), the following fields must be processed:

Field name	Value	Description
vads_card_brand	MULTI	Several types of payment card are used for the payment.
vads_payment_seq	In Json format, see details below.	Details of performed transactions.

The **vads_payment_seq** field (json format) describes the split payment sequence. It contains:

- "trans_id": transaction identifier used for the entire payment sequence.
- "transaction": table of sequence transactions. It contains the following elements:

Field name	Description
amount	Amount of the payment sequence.
operation_type	Debit transaction.
auth_number	Authorization number. Example: 949478
auth_result	Return code of the authorization request.
capture_delay	Delay before the capture (in days). <ul style="list-style-type: none"> For a payment by card, this parameter is the requested capture date (ISO 8601 format). If not sent in the payment form, the value defined in the Expert Back Office will be used.
card_brand	Used payment method. For a payment by card (e.g. CB or Visa or MasterCard co-branded CB cards), this parameter is set to "CB".

Field name	Description																														
	See the Payment Gateway Implementation Guide available in our online documentation archive to see the complete list of card types.																														
card_number	Payment method number																														
expiry_month	Expiry month of the payment method																														
expiry_year	Expiry year of the payment method																														
payment_certificate	Payment certificate.																														
contract_used	Contract used for the payment																														
identifier	Unique identifier (token) associated with a payment method.																														
identifier_status	<p>Only present if the requested action is a token creation or update. Possible values:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>CREATED</td> <td>The authorization request has been accepted. Token (or UMR for SEPA payment) has been successfully created.</td> </tr> <tr> <td>NOT_CREATED</td> <td>The authorization request has been declined. The token (or UMR for SEPA payment) has not been created, and therefore cannot be viewed in the Expert Back Office.</td> </tr> <tr> <td>UPDATED</td> <td>Token (or UMR for SEPA payment) has been successfully updated.</td> </tr> <tr> <td>NOT_UPDATED</td> <td>The token (or UMR for SEPA payment) has not been updated.</td> </tr> <tr> <td>ABANDONED</td> <td>The action has been abandoned by the buyer (debtor). The token (or UMR for SEPA payment) has not been created, and therefore cannot be viewed in the Expert Back Office.</td> </tr> </tbody> </table>	Value	Description	CREATED	The authorization request has been accepted. Token (or UMR for SEPA payment) has been successfully created.	NOT_CREATED	The authorization request has been declined. The token (or UMR for SEPA payment) has not been created, and therefore cannot be viewed in the Expert Back Office.	UPDATED	Token (or UMR for SEPA payment) has been successfully updated.	NOT_UPDATED	The token (or UMR for SEPA payment) has not been updated.	ABANDONED	The action has been abandoned by the buyer (debtor). The token (or UMR for SEPA payment) has not been created, and therefore cannot be viewed in the Expert Back Office.																		
Value	Description																														
CREATED	The authorization request has been accepted. Token (or UMR for SEPA payment) has been successfully created.																														
NOT_CREATED	The authorization request has been declined. The token (or UMR for SEPA payment) has not been created, and therefore cannot be viewed in the Expert Back Office.																														
UPDATED	Token (or UMR for SEPA payment) has been successfully updated.																														
NOT_UPDATED	The token (or UMR for SEPA payment) has not been updated.																														
ABANDONED	The action has been abandoned by the buyer (debtor). The token (or UMR for SEPA payment) has not been created, and therefore cannot be viewed in the Expert Back Office.																														
presentation_date	For card payments, this parameter is the requested capture date (ISO 8601 format).																														
trans_id	Transaction number.																														
ext_trans_id	This field is not sent for credit card payments.																														
trans_uuid	Unique reference generated by the payment gateway after the creation of a payment transaction. Guarantees that each transaction is unique.																														
extra_result	<p>Numeric code of the risk controls result.</p> <table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Empty</td> <td>No verification completed.</td> </tr> <tr> <td>00</td> <td>All the verification processes have been successfully completed.</td> </tr> <tr> <td>02</td> <td>Credit card velocity exceeded.</td> </tr> <tr> <td>03</td> <td>The card is on the merchant's greylist.</td> </tr> <tr> <td>04</td> <td>The country of origin of the card is on the merchant's greylist.</td> </tr> <tr> <td>05</td> <td>The IP address is on the merchant's greylist.</td> </tr> <tr> <td>06</td> <td>The BIN code is on the merchant's greylist..</td> </tr> <tr> <td>07</td> <td>Detection of an e-carte bleue.</td> </tr> <tr> <td>08</td> <td>Detection of a national commercial card.</td> </tr> <tr> <td>09</td> <td>Detection of a foreign commercial card.</td> </tr> <tr> <td>14</td> <td>Detection of a card that requires systematic authorization.</td> </tr> <tr> <td>20</td> <td>Relevance verification: countries do not match (country IP address, card country, buyer's country).</td> </tr> <tr> <td>30</td> <td>The country of the this IP address belongs to the greylist.</td> </tr> <tr> <td>99</td> <td>Technical issue encountered by the server during a local verification process.</td> </tr> </tbody> </table>	Code	Description	Empty	No verification completed.	00	All the verification processes have been successfully completed.	02	Credit card velocity exceeded.	03	The card is on the merchant's greylist.	04	The country of origin of the card is on the merchant's greylist.	05	The IP address is on the merchant's greylist.	06	The BIN code is on the merchant's greylist..	07	Detection of an e-carte bleue.	08	Detection of a national commercial card.	09	Detection of a foreign commercial card.	14	Detection of a card that requires systematic authorization.	20	Relevance verification: countries do not match (country IP address, card country, buyer's country).	30	The country of the this IP address belongs to the greylist.	99	Technical issue encountered by the server during a local verification process.
Code	Description																														
Empty	No verification completed.																														
00	All the verification processes have been successfully completed.																														
02	Credit card velocity exceeded.																														
03	The card is on the merchant's greylist.																														
04	The country of origin of the card is on the merchant's greylist.																														
05	The IP address is on the merchant's greylist.																														
06	The BIN code is on the merchant's greylist..																														
07	Detection of an e-carte bleue.																														
08	Detection of a national commercial card.																														
09	Detection of a foreign commercial card.																														
14	Detection of a card that requires systematic authorization.																														
20	Relevance verification: countries do not match (country IP address, card country, buyer's country).																														
30	The country of the this IP address belongs to the greylist.																														
99	Technical issue encountered by the server during a local verification process.																														
sequence_number	Sequence number.																														
trans_status	Transaction status.																														

Table 12: JSON object content

Note: canceled transactions are also displayed in the table.

14. Store the value of the **vads_trans_uuid** field. It will allow you to assign unique identification to the transaction if you use the Web Service APIs.

15.Retrieve all the order, buyer and shipping details.

These details will be provided in the response only if they have been transmitted in the payment form.

Their values are identical to the ones submitted in the form.

16.Proceed to order update.

7. RETURNING TO THE SHOP

By default, when the buyer returns to the merchant website, no parameters will be transmitted by their browser.

However, if the **vads_return_mode** field has been transmitted in the payment form (see chapter **Managing the return to the merchant website** of the Hosted Payment Page Implementation Guide available in our online document archive) it will be possible to retrieve the data:

- either via GET, the data is presented in the URL as follows: ?field1=value1&field2=value2
- or via POST: the data is sent in a POST form.

The data transmitted to the browser is the same as for notifications (IPN).

The **vads_url_check_src** and **vads_hash** fields will be sent only in the instant notification.

To analyze this data, see chapter **Analyzing the payment result**.

Note: the return to the shop will allow you to show only the visual context to the buyer. Do not use the received data for processing in the database.

8. PROCEEDING TO TESTING

Before going to live mode it is necessary to run tests to make sure that there are no interaction problems between the merchant website and the payment gateway.

These tests must be done before requesting to go to live mode.

8.1. Making payment tests

The test payment requests submitted via the HTTP POST form must:

- Contain the **vads_ctx_mode** field set to **TEST**.
- Use the **test key** retrieved earlier for signature computation.

In test phase, the merchant can test the 3D Secure configuration (if the merchant is enrolled in 3DS and if this option is not disabled).

Different cases of payments can be simulated by using test card numbers specified on the payment page.

All the transactions completed in test mode can be viewed by all persons authorized to use the Lyra Collect Back Office .

8.2. Testing the IPN

First of all, check the status of the IPN URL in the Expert Back Office.

To do this:

1. Select a transaction with a right-click.
2. Select **Display transaction details**.
3. Check the status of the IPN.
 - In case the status is **Sent**, it means that you have correctly specified the URL in the Expert Back Office.
 - In case the status is **Undefined URL**, it means that you have not specified the URL in the Expert Back Office.
 1. Check the address of the IPN URL entered in TEST and PRODUCTION mode.
 2. Click **Settings > Notification rules**.
 3. Enter the IPN URL (Instant Payment Notification URL at the end of payment).

Do not enter an address in "localhost". The call to this URL is made from one server to another.
 4. Click **Save**.
 - In case the status is **Failed**, see chapter **Processing errors** of the Hosted Payment Page Implementation Guide available in our online documentation archive.