



# **Quick start**

## **Implementation Guide**

Document version 1.9

# Contents

<b>1. HISTORY OF THE DOCUMENT.....</b>	<b>3</b>
<b>2. ESTABLISHING INTERACTION WITH THE PAYMENT GATEWAY.....</b>	<b>4</b>
2.1. Setting up the payment page URL.....	4
2.2. Identifying yourself when exchanging with the payment gateway.....	4
2.3. Managing interaction with the merchant website.....	6
2.4. Managing security.....	8
<b>3. SETTING UP NOTIFICATIONS.....</b>	<b>10</b>
3.1. Setting up the Instant Payment Notification.....	10
3.2. Automatic retry in case of failure.....	11
3.3. Other cases of notification.....	13
<b>4. SENDING A PAYMENT FORM VIA POST.....</b>	<b>14</b>
<b>5. COMPUTING THE SIGNATURE.....</b>	<b>20</b>
5.1. Example of implementation with JAVA.....	22
5.2. Example of implementation with PHP.....	25
<b>6. IMPLEMENTING THE IPN.....</b>	<b>26</b>
6.1. Preparing your environment.....	27
6.2. Retrieving data returned in the response.....	28
6.3. Computing the IPN signature.....	29
6.4. Comparing signatures.....	30
6.5. Analyzing the nature of the notification.....	31
6.6. Processing the response data.....	33
6.7. Running tests and troubleshooting.....	40
<b>7. RETURNING TO THE SHOP.....</b>	<b>43</b>
<b>8. PROCEEDING TO TEST PHASE.....</b>	<b>44</b>
<b>9. ACTIVATING THE SHOP IN PRODUCTION MODE.....</b>	<b>46</b>
9.1. Generating the production key.....	46
9.2. Shifting your merchant website to production mode.....	46
9.3. Making the first production payment.....	46
<b>10. OBTAINING HELP.....</b>	<b>47</b>

## 1. HISTORY OF THE DOCUMENT

Version	Author	Date	Comment
1.9	La Banque Postale	11/20/2020	<ul style="list-style-type: none"><li>• Update of the <i>Sending a payment form via POST</i> chapter.</li></ul>
1.8	La Banque Postale	7/30/2020	<ul style="list-style-type: none"><li>• Correction of field format for <b>vads_trans_date</b>.</li><li>• Update of the <i>Setting up the Instant Payment Notification</i> chapter.</li></ul>
1.7	La Banque Postale	12/9/2019	<ul style="list-style-type: none"><li>• Update of the <b>Proceeding to testing</b> chapter</li><li>• Update of the IPN setup procedure.</li><li>• Addition of the <b>Implementing the IPN</b> chapter.</li><li>• Correction of field format for <b>vads_product_label</b>.</li><li>• Modification of field format for <b>vads_trans_id</b>.</li></ul>
1.6	La Banque Postale	8/30/2019	Initial version

This document and its contents are confidential. It is not legally binding. Any reproduction and / or distribution of all or part of this document or its content to a third party is strictly prohibited or subject to prior written authorization from La Banque Postale. All rights reserved.

## 2. 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:

- Via 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 known only to the merchant and the payment gateway.

### 2.1. Setting up the payment page URL

---

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

<https://scelliuspaiement.labanquepostale.fr/vads-payment/>

### 2.2. 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 Merchant Back Office: <https://scelliuspaiement.labanquepostale.fr/vads-merchant/>

2. Enter your login.

The login is sent to the merchant's e-mail address (the subject of the e-mail is **Connection identifiers- [your shop name]**).

3. Enter your password.

The password is sent to the merchant's e-mail address (the subject of the e-mail is **Connection identifiers- [your shop name]**).

4. Click **Sign in**.

After 3 password entry errors, the user's account is locked. Click on the link **Forgotten password or locked account** to reset it.



A user's password is valid for 90 days. Beyond this period, a renewal will be requested when connecting.

5. Click **Settings > Shop**.

6. Select **Keys**.

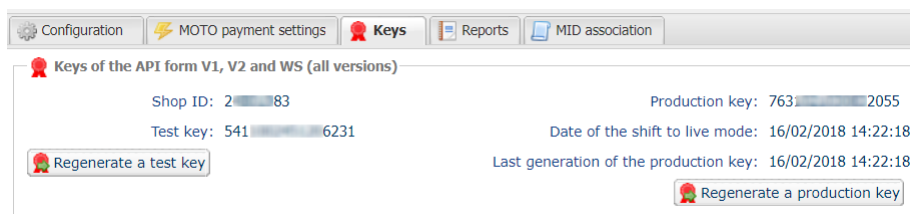


Figure 1: Keys tab

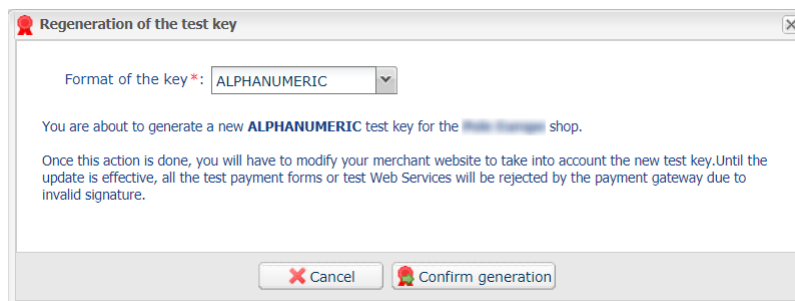
Two types of keys are available:

- 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.

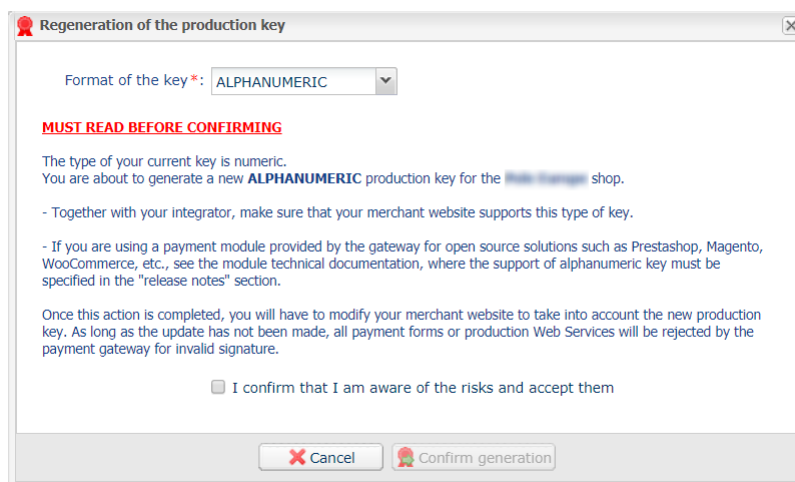
These keys can be numeric or alphanumeric.

**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").



## 2.3. Managing interaction with the merchant website

---

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

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

### Instant Payment Notification - IPN

The **Notification URL** is the URL of a specific page on the merchant website that is **automatically** called by the payment gateway when certain events take place.

By default, the rules are created to manage the events below:

- end of payment (accepted or rejected),
- payment abandoned or canceled,
- token creation or update,
- recurring payment creation,
- new installment date,
- authorization made in case of a deferred payment,
- update of a transaction status by the acquirer,
- operation made via the Merchant Back Office (cancellation, refund, duplication, manual payment, etc.).

These rules must be enabled and configured according to the needs of the merchant.

With each call, the payment gateway transmits transaction details to the merchant website. It is called instant notification (or **IPN** as in Instant Payment Notification).

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

### URL of return to the merchant website

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

*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**).



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**).

## 2.4. Managing security

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

### 2.4.1. 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 Merchant 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:

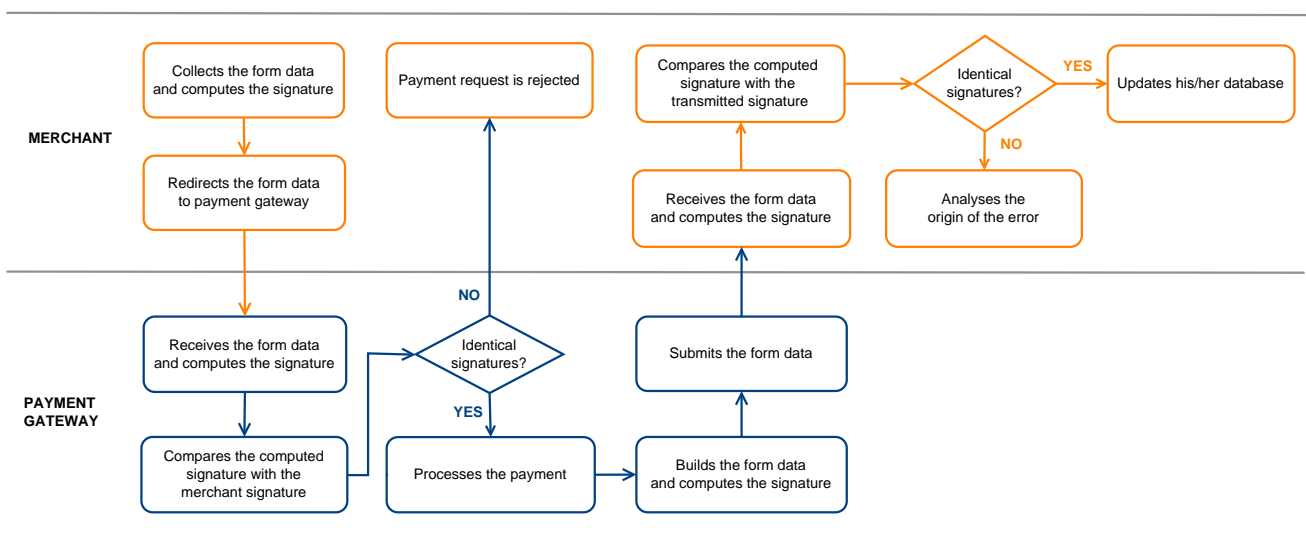


Figure 3: Diagram of a security mechanism

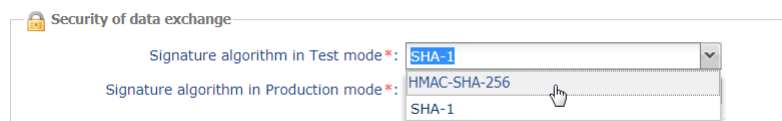
1. The merchant website builds the form data and computes the signature.
2. The merchant website submits the form to the gateway.
3. The gateway receives the form data and computes the signature.
4. The gateway compares the computed signature with the signature that was transmitted by the merchant website.
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.).

### 2.4.2. Selecting the hash algorithm

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



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



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.**

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

### 2.4.4. Managing sensitive data

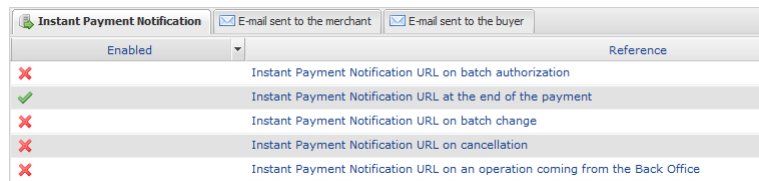
Online payment transactions are regulated by strict rules (PCI-DSS certification).

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. SETTING UP NOTIFICATIONS

To access the notification rule management, open the menu: **Settings > Notification rules**.



Instant Payment Notification	
<input checked="" type="checkbox"/> E-mail sent to the merchant <input checked="" type="checkbox"/> E-mail sent to the buyer	
Enabled	Reference
✗	Instant Payment Notification URL on batch authorization
✓	Instant Payment Notification URL at the end of the payment
✗	Instant Payment Notification URL on batch change
✗	Instant Payment Notification URL on cancellation
✗	Instant Payment Notification URL on an operation coming from the Back Office

The rule configuration tab of “Instant Payment Notification URL call” type opens.

### 3.1. Setting up the Instant Payment Notification

This rule allows to notify the merchant website in the following cases:

- Payment accepted
- Payment refused
- Token creation or update
- Creation of a recurring payment

The **Payment accepted** event corresponds to the creation of a transaction with one of the (**vads\_trans\_status**) statuses below:

- **ACCEPTED**
- **AUTHORISED**
- **AUTHORISED\_TO\_VALIDATE**
- **CAPTURED**
- **INITIAL**
- **UNDER\_VERIFICATION**
- **WAITING\_AUTHORISATION**
- **WAITING\_AUTHORISATION\_TO\_VALIDATE**
- **WAITING\_FOR\_PAYMENT**

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

It will inform the merchant website of the payment result even if your client has not clicked the “Return to the shop” button.

1. Right-click **Instant Payment Notification URL at the end of the payment**.
2. Select **Manage the rule**.
3. Enter the **E-mail address(es) to notify in case of failure** field in the **General settings** section.  
To specify several e-mail addresses, separate them with a semi-colon.
4. Check the box **Automatic retry in case of failure** if you wish to authorize the gateway to automatically resend the notification in case of a failure (can be done up to 4 times).  
For more information, please see chapter [Automatic retry in case of failure](#) on page 11.

5. In the **Instant Payment Notification URL of the API form V1, V2** section, specify the URL of your page in the fields **URL to notify in TEST mode** and **URL to notify in PRODUCTION mode**.
6. Save the changes.

## 3.2. Automatic retry in case of failure

**Automatic retry does not apply to notifications manually triggered via the Merchant Back Office.**

The merchant can enable a mechanism that allows the payment gateway to automatically return notifications when the merchant website is temporarily unavailable, **up to 4 times**.

A notification will be considered as failed if the HTTP code returned by the merchant site is not on the following list: **200, 201, 202, 203, 204, 205, 206, 301, 302, 303, 307, 308**.

Call attempts are scheduled at fixed intervals every 15 minutes (00, 15, 30, 45).

After each failed attempt, a notification e-mail is sent to the e-mail address specified in the configuration of the notification rule in question.

In this case, the subject of the e-mail contains the number corresponding to the notification retry attempt. It is presented as **attempt #** followed by the attempt number.

- Example of an e-mail subject following a first notification failure at the end of payment:

```
[MODE TEST] My Shop - Tr. ref. 067925 / FAILURE during the call to your IPN URL  
[unsuccessful attempt #1]
```

- Example of an e-mail subject following a second failure:

```
[MODE TEST] My Shop - Tr. ref. 067925 / FAILURE during the call to your IPN URL  
[unsuccessful attempt #2]
```

- Example of an e-mail subject following a third failure:

```
[MODE TEST] My Shop - Tr. ref. 067925 / FAILURE during the call to your IPN URL  
[unsuccessful attempt #3]
```

- Example of an e-mail subject following the last failure:

```
[MODE TEST] My Shop - Tr. ref. 067925 / FAILURE during the call to your IPN URL  
[unsuccessful attempt #last]
```

To notify the merchant website of the last notification attempt, the e-mail subject will contain the mention **attempt #last**.

During the automatic retry, certain details are not stored in the database or are modified.

**Examples of fields not available/not registered in the database:**

Field name	Description
<b>vads_page_action</b>	Completed operation
<b>vads_payment_config</b>	Payment type (immediate or installment).
<b>vads_action_mode</b>	Acquisition mode for payment method data.

**Examples of fields sent with different values:**

Field name	New value
<b>vads_url_check_src</b>	Always set to <b>RETRY</b> in case of automatic retry.

Field name	New value
<b>vads_trans_status</b>	The transaction status may vary between the initial call and the automatic retry (cancellation by the merchant, transaction capture at the bank, etc.).
<b>vads_hash</b>	The value of this field is regenerated with each call.
<b>signature</b>	The signature value depends on the different statuses that may vary between the initial call and the automatic retry.

These e-mails contain:

- the encountered problem,
- parts of analysis depending on the error,
- its consequences,
- instructions for manually triggering the notification from the Merchant Back Office.



After the fourth attempt, it is still possible to retry the IPN URL **manually** via your Merchant Back Office.



Warning, during the automatic retry, any manual call to the IPN URL will affect the number of automatic attempts:

- a successful manual call will stop the automatic retry,
- a failed manual call will have no impact on the current automatic retry.

### 3.3. 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 Merchant Back Office
- cancellation of a transaction from the Merchant Back Office
- validation of a transaction from the Merchant Back Office
- modification of a transaction from the Merchant Back Office
- etc.

For more information on configuring rules, see *Notification center* user guide.

## 4. SENDING A 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://scelliuspaiement.labanquepostale.fr/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 be computed incorrectly 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 " <code>{</code> " and ends with a right brace " <code>}</code> ". Each key/value pair contains the name of the key between double-quotes followed by " <code>:</code> " followed by a value. The name of the key must be alphanumeric. The value can be: <ul style="list-style-type: none"><li>• a chain of characters (in this case it must be framed by double-quotes)</li><li>• a number</li><li>• an object</li><li>• a table</li><li>• a boolean</li><li>• empty</li></ul> Example: <code>{"name1":45,"name2":"value2", "name3"=false}</code>
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 " <code>;</code> ". The list of possible values is given in the field definition. Example: <code>vads_payment_cards=VISA;MASTERCARD</code>
map	List of key / value pairs separated by a " <code>;</code> ".

Notation	Description
	<p>Each key/value pair contains the name of the key followed by "=", followed by a value. The value can be:</p> <ul style="list-style-type: none"> <li>a chain of characters</li> <li>a boolean</li> <li>a json object</li> <li>an xml object</li> </ul> <p>The list of possible values for each key/value pair is provided in the field definition. Example: vads_theme_config=SIMPLIFIED_DISPLAY=true;RESPONSIVE_MODEL=Model_1</p>

- Required fields:

Field name	Description	Format	Value
<b>signature</b>	Signature guaranteeing the integrity of the requests exchanged between the merchant website and the payment gateway.	ans	Ex : <b>ycA5Do5tNvsNkdc/ eP1bj2xa19z9q3iWPy9/rpesfS0=</b>
<b>vads_action_mode</b>	Acquisition mode for payment method data	enum	<b>INTERACTIVE</b>
<b>vads_amount</b>	Payment amount in the smallest currency unit (cents for euro)	n..12	E.g.: 4525 for EUR 45.25
<b>vads_ctx_mode</b>	Mode of interaction with the payment gateway	enum	<b>TEST</b> or <b>PRODUCTION</b>
<b>vads_currency</b>	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)
<b>vads_page_action</b>	Action to perform	enum	<b>PAYMENT</b>
<b>vads_payment_con</b>	Payment type	enum	<b>SINGLE</b> for immediate payment <b>MULTI</b> for installment payment
<b>vads_site_id</b>	Shop ID	n8	E.g.: 12345678
<b>vads_trans_date</b>	Date and time of the payment form in UTC format	n14	Respect the YYYYMMDDHHMMSS format E.g.: 20200101130025
<b>vads_trans_id</b>	Transaction number. Must be unique within the same day (from 00:00:00 UTC to 23:59:59 UTC). <b>Warning: this field is not case sensitive.</b>	an6	E.g.: xrT15p
<b>vads_version</b>	Version of the exchange protocol with the payment gateway	enum	<b>V2</b>

- Highly recommended fields:
  - The payment method to be used

Field name	Description	Format	Value
<b>vads_payment_cards</b>	Allows to force the card type to be used. It is recommended to provide a different payment button for each payment method on the merchant website. <b>It is recommended not to leave the field empty.</b> See chapter <i>Managing the payment methods offered to the buyer</i> of the Hosted Payment Page Implementation guide for more information.	enum	E.g.: <ul style="list-style-type: none"> <li>• CB</li> <li>• CVCO</li> <li>• MASTERCARD</li> <li>• VISA</li> <li>• SDD</li> </ul>

- Order details

Field name	Description	Format	Value
<b>vads_order_id</b>	Order ID Can contain uppercase or lowercase characters, numbers or hyphens ([A-Z] [a-z], 0-9, _ -).	ans..64	E.g.: 2-XQ001
<b>vads_order_info</b>	Additional order info	ans..255	E.g.: Door code 3125
<b>vads_order_info2</b>	Additional order info	ans..255	E.g.: No elevator
<b>vads_order_info3</b>	Additional order info	ans..255	E.g.: Express
<b>vads_ext_info_xxxx</b>	Additional information necessary for the merchant that will be displayed both in the payment confirmation e-mail sent to the merchant and in the Merchant Back Office ( <b>Extra</b> tab of the transaction details). <b>xxxx</b> corresponds to the name of the transmitted data. For example: vads_ext_info_departure_c	ans..255	E.g.: LHR



- Buyer details

Field name	Description	Format	Value
vads_cust_email	Buyer's e-mail address	ans..150	E.g.: ABC@example.com
vads_cust_id	Buyer reference on the merchant website	an..63	E.g.: C2383333540
vads_cust_national_id	National identifier	ans..255	E.g.: 940992310285
vads_cust_title	Buyer's title	an..63	E.g.: M
vads_cust_status	Status	enum	<b>PRIVATE:</b> for a private individual <b>COMPANY:</b> for a company
vads_cust_first_name	First name	ans..63	E.g.: Laurent
vads_cust_last_name	Last name	ans..63	E.g.: Durant
vads_cust_legal_name	Buyer's legal name	an..100	E.g.: D. & Cie
vads_cust_phone	Phone number	an..32	E.g.: 0467330222
vads_cust_cell_phone	Cell phone number	an..32	E.g.: 06 12 34 56 78
vads_cust_address_number	Street number	ans..64	E.g.: 109
vads_cust_address	Postal address	ans..255	E.g.: Rue de l'Innovation
vads_cust_address2	Address line 2	ans..255	E.g.:
vads_cust_district	District	ans..127	E.g.: Centre ville
vads_cust_zip	Zip code	an..64	E.g.: 31670
vads_cust_city	City	an..128	E.g.: Labège
vads_cust_state	State / Region	ans..127	E.g.: Occitanie
vads_cust_country	Country code in compliance with the ISO 3166 alpha-2 standard	a2	E.g.: "FR" for France, "PF" for French Polynesia, "NC" for New Caledonia, "US" for the United States.

- Recommended fields:

- 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 (required for triggering one or more actions if the <b>Shipping country control</b> profile is enabled).	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	Last 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	<b>PRIVATE:</b> for shipping to a private individual <b>COMPANY:</b> for shipping to a company

Field name	Description	Format	Value
<b>vads_ship_to_street_number</b>	Street number	ans..64	E.g.: 2
<b>vads_ship_to_street</b>	Postal address	ans..255	E.g.: Rue Sainte Catherine
<b>vads_ship_to_street2</b>	Address line 2	ans..255	
<b>vads_ship_to_zip</b>	Zip code	an..64	E.g.: 33000

- Shopping cart details

Field name	Description	Format	Value
<b>vads_nb_products</b>	Number of items in the cart	n..12	E.g.: 2
<b>vads_product_ext_idN</b>	Product barcode 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"
<b>vads_product_labelN</b>	Item name. N corresponds to the index of the item (0 for the first one, 1 for the second one, etc.).	ans..255	E.g.: vads_product_label0 = "tee-shirt" vads_product_label1 = "Biscuit" vads_product_label2 = "sandwich"
<b>vads_product_amountN</b>	Price of the item incl. VAT. 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"
<b>vads_product_typeN</b>	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" vads_product_type1 = "FOOD_AND_GROCERY" vads_product_type2 = "FOOD_AND_GROCERY"
<b>vads_product_refN</b>	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"
<b>vads_product_qtyN</b>	Item quantity. 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"

**Note:**

When the **vads\_nb\_products** field is populated, the **Shopping cart** tab becomes available in the transaction details in the Merchant Back Office.

However, if the other fields that start with **vads\_product\_** are not populated, the tab will not include any information. For this reason, when populating the **vads\_nb\_products** field, it becomes mandatory to populate the other fields that start with **vads\_product\_**.

- Optional fields :

You can use additional optional parameters.

See the chapter **Data Dictionary** of the Hosted Payment Page Implementation guide available on our web site to see the list of the available 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 key value is available in your Merchant Back Office via **Settings > Shop > Keys** tab.

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



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

The use of SHA-1 algorithm is deprecated but maintained for compliance reasons.

To compute the signature:

1. Sort the fields whose name begins with **vads\_** alphabetical order.
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 the "+ character".
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. **Deprecated.**
  - 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://scelliuspaiement.labanquepostale.fr/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. We sort in **alphabetical** order the fields whose name begins with **vads\_** :

- 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. We concatenate the value of these fields with 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 for processing 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 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
    $signature_content = "";

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

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

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

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

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

### Example of 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
    $signature_content = "" ;

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

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

## 6. IMPLEMENTING THE IPN

The script must include at least the following steps:

- Retrieve the field list sent with the POST response
- Compute the signature taking into account the received data
- Compare the computed signature with the received signature
- Analyze the nature of the notification
- Retrieve the payment result

The script may check the order status (or any information of your choice) to see if it has not already been updated.

Once these steps are completed, the script can update the database (new order status, stock update, registration of payment information, etc.).

In order to facilitate support and diagnosis by the merchant in the event of a notification error, we recommend to write messages that will allow you to know at which stage of processing the error occurred.

The gateway reads and stores the first 256 bytes of the HTTP response.

You can write messages throughout the processing. Here are some examples of messages that you can use:

Message	Use case
<b>Data received.</b>	Message to display when retrieving data. Allows to confirm that the notification has been received by the merchant website.
<b>POST is empty.</b>	Message to display when retrieving data. Allows to bring out a possible redirection that would have caused the parameters posted by the payment gateway to be lost.
<b>An error occurred while computing the signature.</b>	Message to be displayed when the verification of the response signature has failed.
<b>Order successfully updated.</b>	Message to be displayed at the end of the file once your processing has been successfully completed.
<b>An error occurred while updating the order.</b>	Message to be displayed at the end of the file if an error occurred during your processing.

## 6.1. Preparing your environment

---



The notifications of Instant Payment Notification URL call type are very important as they represent the only reliable way for the merchant website to obtain the payment result.

It is therefore necessary to make sure the notifications function properly.

Here are some guidelines:

- In order for the dialog between the payment gateway and your merchant website to work, you must make sure, together with your technical teams, that the **194.50.38.0/24** IP address range is authorized on the various devices within your system (firewalls, apache server, proxy server, etc.).

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

- Using redirection leads to losing data presented in POST.

This is the case if there is a configuration on your devices or on the side of your host that redirects the URLs of “<http://www.example.com>” type to “<http://example.com>” or “<http://example.com>” to “<https://example.com>”.

- HTML must not be visible on the page. Access to images or CSS slows down the exchange between the payment gateway and the merchant website.

- Avoid integrating time-consuming tasks, such as PDF invoice generation or sending e-mails in your script.

The processing time has a direct influence on the time it takes to display the payment summary page.

**The longer the processing of the notification, the greater the delay for displaying the page. After 35 seconds, the payment gateway considers that the call has failed (timeout).**

- If your page is only accessible in https, test your URL on the Qualys SSL Labs website (<https://www.ssllabs.com/ssltest/>) and, if necessary, change your configuration if necessary in order to obtain the A score.

Your SSL certificate must be signed by a certification authority known and recognized on the market.

- Make sure that you use the latest version of the TLS protocol in order to maintain a high level of security.

## 6.2. Retrieving data returned in the response

---

The data returned in the response depends on the parameters sent in the payment request, the payment type, the settings of your shop and the notification format.

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 response consists of a field list. Each field contains a response value. The field list can be updated.

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

It is recommended to test the presence of the **vads\_hash** field, which is only present during a notification.

```
if (empty ($_POST)){
    echo 'POST is empty';
}
else{
    echo 'Data Received ';
    if (isset($_POST['vads_hash'])){
        echo 'Form API notification detected';
        //Signature computation
        //Signature verification
        //Order Update
    }
}
```

## 6.3. Computing the IPN signature

The signature is computed by following the same logic as for creating the payment request.



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 the payment request.**

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 the "+" character".
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. **Deprecated.**
  - 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.

### Examples in PHP:

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

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

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

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

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

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

## 6.4. Comparing signatures

---

To ensure the integrity of the response, you must compare the signature contained in the IPN with the value computed in the previous step.



You should not compare the signature of the IPN with the signature that you transmitted in your payment request.

If the signatures match

- You may consider the response as safe and proceed with the analysis.
- Otherwise, the script will have to raise an exception and notify the merchant about the anomaly.

**Example in PHP:**

```
if ($_POST['signature'] == $sign){  
    //Processing data  
}  
else{  
    throw new Exception('An error occurred while computing the signature');  
}
```

The signatures may not match in case of:

- an implementation error (error in your calculation, problem with UTF-8 encoding, etc.),
- an error in the key value or in the **vads\_ctx\_mode** field (frequent issue when shifting to production mode),
- a data corruption attempt.

## 6.5. Analyzing the nature of the notification

During a notification, the **vads\_url\_check\_src** field allows to differentiate the notifications based on their triggering event:

- creation of a transaction
- new notification sent by the merchant via the Merchant Back Office

It specifies the applied notification rule:

Value	Applied rule
<b>PAY</b>	The PAY value is sent in the following cases: <ul style="list-style-type: none"><li>• immediate payment (or first installment payment of a recurring payment)</li><li>• payment deferred for less than 7 days Only if the merchant has configured the <b>Instant Payment Notification URL at the end of payment</b> rule.</li><li>• payment abandoned or canceled by the buyer Only if the merchant has configured the <b>Instant Payment Notification URL on cancellation</b> rule.</li></ul>
<b>BO</b>	Execution of the notification via the Merchant Back Office (right-click a transaction > <b>Send the Instant Payment Notification</b> ).
<b>BATCH</b>	The BATCH value is sent in case of an update of a transaction status after its synchronization on the acquirer side. This is the case of payments with redirection to the acquirer. Only if the merchant has configured the rule <b>Instant Payment Notification URL on batch change</b> .
<b>BATCH_AUTO</b>	The BATCH_AUTO value is sent in the following cases: <ul style="list-style-type: none"><li>• payment deferred for more than 7 days</li><li>• installments of a recurring payment (except the first one) Only if the merchant has configured the <b>Instant Payment Notification URL on batch authorization</b> rule.</li></ul> The notification is sent with the authorization request for payments with "Waiting for authorization" status.
<b>REC</b>	The REC value is sent only for recurring payments if the merchant has configured the <b>Instant Payment Notification URL when creating recurring payments</b> rule.
<b>MERCH_BO</b>	The MERCH_BO value is sent: <ul style="list-style-type: none"><li>• during an operation made via the Merchant Back Office (refund, cancellation, modification, validation, duplication, creation and/or update of token), only if the merchant has configured the following notification rule: <b>Instant Payment Notification URL on an operation coming from the Back Office</b></li></ul>
<b>RETRY</b>	Automatic retry of the IPN.

Table 1: Values associated with the **vads\_url\_check\_src** field

After checking its value, the script can process differently depending on the nature of the notification.

For example:

If **vads\_url\_check\_src** is set to **PAY** or **BATCH\_AUTO**, the script will update the order status, etc.

If **vads\_url\_check\_src** is set to **REC**, the script will retrieve the recurring payment reference and will increment the number of the expired installment payments in case the payment has been accepted, etc.





## 6.6. 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 been updated yet.
3. Retrieve the payment result transmitted in the **vads\_trans\_status** field.  
Its value allows you to define the order status.

Value	Description
<b>ABANDONED</b>	<b>Abandoned</b> Payment abandoned by the buyer The transaction was not created, and <b>is therefore not visible in the Merchant Back Office.</b>
<b>ACCEPTED</b>	<b>Accepted.</b> Status of a VERIFICATION type transaction for which the authorization request or information request has been successfully completed. This status cannot evolve. Transactions with the “ <b>ACCEPTED</b> ” status will never be captured.
<b>AUTHORISED</b>	<b>Waiting for capture</b> The transaction has been accepted and will be automatically captured at the bank on the expected date.
<b>AUTHORISED_TO_VALIDATE</b>	<b>To be validated</b> 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 <b>EXPIRED</b> . The <b>Expired</b> status is final.
<b>CANCELLED</b>	<b>Canceled</b> The transaction has been canceled by the Merchant.
<b>CAPTURED</b>	<b>Captured</b> The transaction has been captured by the bank.
<b>CAPTURE_FAILED</b>	<b>Capture failed</b> Contact the technical support.
<b>EXPIRED</b>	<b>Expired</b> This status appears in the lifecycle of a payment with deferred capture. The expiry date of the authorization request has passed and the Merchant has not validated the

Value	Description
	transaction. The account of the cardholder will therefore not be debited.
<b>REFUSED</b>	<b>Refused</b> Transaction is declined.
<b>SUSPENDED</b>	<b>Suspended</b> 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 <b>CAPTURED</b> .
<b>UNDER_VERIFICATION</b>	<b>Control in progress</b> Waiting for the response from the acquirer. This status is temporary. A notification will be sent to the merchant website to inform the Merchant of the status change. Requires the activation of the Instant Payment Notification URL on batch change notification rule.
<b>WAITING_AUTHORISATION</b>	<b>Waiting for authorization</b> The capture delay in the bank exceeds the authorization validity period.
<b>WAITING_AUTHORISATION_TO_VALIDATE</b>	<b>To be validated and authorized</b> The capture delay in the bank exceeds the authorization validity period. A EUR 1 (or information request about the CB network if the acquirer supports it) authorization has been accepted. The Merchant must manually validate the transaction for the authorization request and the capture to occur.

4. Analyze the **vads\_occurrence\_type** field to determine if it is a single payment or a payment that is part of a series (subscription or installment payment).

Value	Description
<b>UNITAIRE</b>	Single payment (immediate payment).
<b>RECURRENT_INITIAL</b>	First payment of a series
<b>RECURRENT_INTERMEDIAIRE</b>	Nth payment of a series
<b>RECURRENT_FINAL</b>	Last payment of a series

5. Analyze the **vads\_payment\_config** field to determine whether it is an **installment payment**.

Field name	Value for an immediate payment	Value for a payment in installments
<b>vads_payment_config</b>	SINGLE	MULTI (the exact syntax is MULTI:first=X;count=Y;period=Z)

For a payment in installments, identify the installment number by retrieving the value of the **vads\_sequence\_number** field.

Warning: with the application of Soft Decline, the **vads\_sequence\_number** field no longer allows to easily identify the first installment of a payment in installments. Since the sequence number of the first installment can be different from 1, the sequence number of the second installment will not necessarily be 2.

6. Retrieve the value of the **vads\_trans\_date** field to identify the payment date.
7. 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.
8. Retrieve the used amount and currency. To do this, retrieve the values of the following fields:

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

9. 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
<b>03</b>	<b>Invalid acceptor</b> 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.). <b>To find out the specific reason of the rejection, the buyer must contact his or her bank.</b>
<b>05</b>	<b>Do not honor</b> This code is sent by the card issuer. This code is used in the following cases: <ul style="list-style-type: none"> <li>• Invalid expiry date</li> <li>• Invalid CVV</li> <li>• Exceeded credit limit</li> <li>• Insufficient funds (etc.)</li> </ul> <b>To find out the specific reason of the rejection, the buyer must contact his or her bank.</b>
<b>51</b>	<b>Insufficient balance or exceeded credit limit</b> 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. <b>To find out the specific reason of the rejection, the buyer must contact his or her bank.</b>
<b>56</b>	<b>Card absent from the file</b> This code is sent by the card issuer. The entered card number is incorrect or the card number + expiration date combination does not exist.
<b>57</b>	<b>Transaction not allowed for this cardholder</b> This code is sent by the card issuer. This code is used in the following cases: <ul style="list-style-type: none"> <li>• The buyer attempts to make an online payment with a cash withdrawal card</li> </ul>

Value	Description
	<ul style="list-style-type: none"> <li>The authorized payment limit is exceeded</li> </ul> <p><b>To find out the specific reason of the rejection, the buyer must contact his or her bank.</b></p>
59	<p><b>Suspected fraud</b></p> <p>This code is sent by the card issuer. This code appears when an incorrect CVV code or expiration date has been entered several times.</p> <p><b>To find out the specific reason of the rejection, the buyer must contact his or her bank.</b></p>
60	<p><b>The acceptor of the card must contact the acquirer</b></p> <p>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 sale contract with manual entry of contract data).</p> <p><b>Contact the customer service to resolve the problem.</b></p>
81	<p><b>Unsecured payment is not accepted by the issuer</b></p> <p>This code is sent by the card issuer. After receiving this code, the payment gateway automatically makes a new payment attempt with 3D Secure authentication, when possible.</p>

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

- b. Retrieve the result of cardholder authentication by retrieving the value of the **vads\_threeds\_status** field.

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

11. 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 Merchant 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
<b>CARD_FRAUD</b>	Verifies whether the cardholder's card number is on the card greylist.

Value	Description
SUSPECT_COUNTRY	Checks whether the issuing country of the buyer's card is on the list of forbidden countries.
IP_FRAUD	Verifies whether the cardholder'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.
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.

The possible values for **result** are:

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

## 12. Retrieve the card type 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_brand_management	Permits to know the brand used when paying, the list of available brands and also if the buyer has changed the default brand chosen by the merchant.
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_label	Name 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).

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

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

The **vads\_payment\_seq** field (json format) describes the split payment sequence. It contains the following elements:

- "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. Will not be returned if not applicable to the used payment method. Example: 949478								
auth_result	Return code of the authorization request.								
capture_delay	Delay before the capture (in days). <ul style="list-style-type: none"> <li>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 Merchant Back Office will be used.</li> </ul>								
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 " <b>CB</b> ". 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	Only present if the requested action is a token creation or update. Possible values: <table border="1"> <thead> <tr> <th>Value</th><th>Description</th></tr> </thead> <tbody> <tr> <td><b>CREATED</b></td><td>The authorization request has been accepted. The token (or UMR for SEPA payment) has been successfully created.</td></tr> <tr> <td><b>NOT_CREATED</b></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 Merchant Back Office.</td></tr> <tr> <td><b>UPDATED</b></td><td>The token (or UMR for SEPA payment) has been successfully updated.</td></tr> </tbody> </table>	Value	Description	<b>CREATED</b>	The authorization request has been accepted. The token (or UMR for SEPA payment) has been successfully created.	<b>NOT_CREATED</b>	The authorization request has been declined. The token (or UMR for SEPA payment) has not been created, and therefore cannot be viewed in the Merchant Back Office.	<b>UPDATED</b>	The token (or UMR for SEPA payment) has been successfully updated.
Value	Description								
<b>CREATED</b>	The authorization request has been accepted. The token (or UMR for SEPA payment) has been successfully created.								
<b>NOT_CREATED</b>	The authorization request has been declined. The token (or UMR for SEPA payment) has not been created, and therefore cannot be viewed in the Merchant Back Office.								
<b>UPDATED</b>	The token (or UMR for SEPA payment) has been successfully updated.								

Field name	Description																															
	Value	Description																														
	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 Merchant Back Office.																														
presentation_date	For a payments by card, 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	Numeric code of the risk assessment result. <table><tr><th>Code</th><th>Description</th></tr><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 is on the greylist.</td></tr><tr><td>99</td><td>Technical issue encountered by the server during a local verification process.</td></tr></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 is on 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 is on the greylist.																															
99	Technical issue encountered by the server during a local verification process.																															
sequence_number	Sequence number.																															
trans_status	Status of the transaction.																															



Canceled transactions are also displayed in the table.

**13.** 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.

**14.** 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.

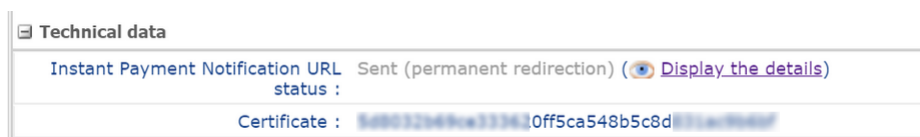
Their values are identical to the ones submitted in the form.

**15.** Proceed to order update.

## 6.7. Running tests and troubleshooting

In order to test the notifications, follow the steps below:

1. Make a payment (in TEST mode or in PRODUCTION mode).
2. Once the payment is complete, look for the transaction in your Back Office (**Management > Transactions** or **TEST Transactions** menu if you made the payment in TEST mode).
3. Double-click the transaction to view the **transaction details**.
4. In the transaction details, search for the section entitled **Technical data**.
5. Check the status of the Instant Payment Notification URL:



The list of possible statuses is provided below:

Status	Description
<b>N/A</b>	The transaction did not result in a notification or no notification rules have been enabled.
Undefined URL	An event has triggered the end of payment notification rule but the URL is not configured.
Call in progress	The notification is in progress. This status is temporary.
Sent	The notification has been successfully sent and a remote device returned an HTTP 200, 201, 202, 203, 204, 205 or 206 response status code.
Sent (permanent redirection)	The merchant website has returned an HTTP 301 or 308 response status code with a new URL to contact. A new call in POST mode has been made to the new URL.
Sent (temporary redirection)	The merchant website has returned an HTTP 302 or 307 response status code with a new URL to contact. A new call in POST mode has been made to the new URL.
Sent (redirection to another page)	The merchant website has returned an HTTP 303 response status code with a new URL to contact. A new call in GET mode has been made to the new URL.
Failed	Generic error different from the codes described below.
Server unavailable	The notification has lasted more than 35s.
<b>SSL handshake failure</b>	Your server is incorrectly configured. Run a test on the Qualys website ( <a href="https://www.ssllabs.com/ssltest/">https://www.ssllabs.com/ssltest/</a> ) and correct the errors.
Connection interrupted	Communication error.
Connection refused	Communication error.
Server error 300	Case of redirection not supported by the gateway.
Server error 304	Case of redirection not supported by the gateway.
Server error 305	Case of redirection not supported by the gateway.
Server error 400	The merchant website returned a HTTP 400 Bad Request code.
<b>Server error 401</b>	The merchant website returned a HTTP 401 Unauthorized code. Make sure that the resource is not protected by an .htaccess file.
Server error 402	The merchant website returned a HTTP 402 Payment Required code.



Status	Description
<b>Server error 403</b>	The merchant website returned a HTTP 403 Forbidden code. Make sure that the resource is not protected by an .htaccess file.
<b>Server error 404</b>	The merchant website returned a HTTP 404 Not Found code. Make sure that the URL is correctly specified in the rule configuration. Make sure that the file is present on your server.
Server error 405	The merchant website returned a HTTP 405 Method Not allowed code.
Server error 406	The merchant website returned a HTTP 406 Not Acceptable code.
Server error 407	The merchant website returned a HTTP 407 Proxy Authentication Required code.
Server error 408	The merchant website returned a HTTP 408 Request Time-out code.
Server error 409	The merchant website returned a HTTP 409 Conflict code.
Server error 410	The merchant website returned a HTTP 410 Gone code.
Server error 411	The merchant website returned a HTTP 411 Length Required code.
Server error 412	The merchant website returned a HTTP 412 Precondition Failed code.
Server error 413	The merchant website returned a HTTP 413 Request Entity Too Large code.
Server error 414	The merchant website returned a HTTP 414 Request-URI Too long code.
Server error 415	The merchant website returned a HTTP 415 Unsupported Media Type code.
Server error 416	The merchant website returned a HTTP 416 Requested range unsatisfiable code.
Server error 417	The merchant website returned a HTTP 417 Expectation failed code.
Server error 419	The merchant website returned a HTTP 419 Authentication Timeout code.
Server error 421	The merchant website returned a HTTP 421 Misdirected Request code.
Server error 422	The merchant website returned a HTTP 422 Unprocessable Entity code.
Server error 423	The merchant website returned a HTTP 423 Locked code.
Server error 424	The merchant website returned a HTTP 424 Failed Dependency code.
Server error 425	The merchant website returned a HTTP 425 Too Early code.
Server error 426	The merchant website returned a HTTP 426 Upgrade Required code.
Server error 429	The merchant website returned a HTTP 431 Request Header Fields Too Large code.
Server error 431	The merchant website returned a HTTP 415 Unsupported Media Type code.
Server error 451	The merchant website returned a HTTP 451 Unavailable For Legal Reasons code.
<b>Server error 500</b>	The merchant website returned a HTTP 500 Internal Server Error code. An application error has occurred on the level of the server hosting your shop. See the logs of your HTTP server (usually apache).



## 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 TEST PHASE


Before the shop goes into production, it is necessary to carry out tests to ensure that the merchant website and the payment gateway are working properly.

The test payment requests must:

- contain the **vads\_ctx\_mode** field set to **TEST**
- use the **test key** for signature computation


Different cases of payments can be simulated by using test card numbers specified on the payment page. The Merchant will be able to test all 3D Secure authentication results (if the Merchant is enrolled and 3DS is not disabled).

The list of tests to be performed to generate the production key is provided in the Merchant Back Office, **Settings > Shop > Keys** menu.


 **Tests control**

Here is a summary of the tests performed up to now.  
You must perform a valid payment for each row in the table below.  
\* manual payments are not taken into account ;  
\* test payments are deleted after 30 days ;  
\* the vads\_page\_action parameter must be set to PAYMENT or REGISTER\_PAY.

CB	Mastercard	Maestro	Visa Electron	Payment date	Test status
4970100000000014	59701003000000018	50005500000000029	49174800000000008		✗
4970100000000055	59701003000000067	50005500000000052	49174800000000057		✗
4970100000000063	59701003000000075	50005500000000060	49174800000000065		✗
4970100000000071	59701003000000083	50005500000000078	49174800000000073		✗

 Refresh the table

The "Generate the production key" button below will become operational once you have successfully completed all the required tests.  
Click on the Refresh the table button to update the test progress.

 Generate production key

Each row of the list contains card numbers associated with the same scenario (i.e. 2 accepted payments and 2 refused payments).

Each column corresponds to a different card type: CB/VISA, MASTERCARD, MAESTRO, VISA ELECTRON.

To perform the test phase:

1. Make an order on your merchant website as if you were one of your buyers.
2. Once redirected to the payment page, select the card type of your choice.
3. Refer to the list of tests to identify the card number to use.
4. Once a test has been validated, its status is updated on the list. Click the **Refresh the table** button if the status has not been updated automatically.
5. Once the 4 tests have been validated, the **Generate production key** button becomes available.

Tests control

Here is a summary of the tests performed up to now.

You must perform a valid payment for each row in the table below.

- \* manual payments are not taken into account ;
- \* test payments are deleted after 30 days ;
- \* the vads\_page\_action parameter must be set to PAYMENT or REGISTER\_PAY.

CB	Mastercard	Maestro	Visa Electron	Payment date	Test status
4970100000000014	5970100300000018	5000550000000029	4917480000000008	03/01/2019 10:53:24	✓
4970100000000055	5970100300000067	5000550000000052	4917480000000057	03/01/2019 10:55:29	✓
4970100000000063	5970100300000075	5000550000000060	4917480000000065	03/01/2019 10:56:32	✓
4970100000000071	5970100300000083	5000550000000078	4917480000000073	03/01/2019 10:57:39	✓

Refresh the table

All the required tests have been successfully completed. You can now generate the production key by clicking on the below button.

Generate the production key

6. Click the **Generate production key** button and accept the notification messages that will appear.

The production key is now available.

## 9. ACTIVATING THE SHOP IN PRODUCTION MODE

---

### 9.1. Generating the production key

---

You can generate the production key via **Settings > Shop > Keys** tab > **Generate the production key** button.

Once the production key has been generated, its value appears in the **Keys** tab.

An e-mail is sent to the company administrator to confirm that the production key has been generated.

### 9.2. Shifting your merchant website to production mode

---

1. Set the **vads\_ctx\_mode** field to **PRODUCTION**.
2. Edit the value of the test key with the value of your production key to compute the signature.  
You will find this value via **Settings > Shop > Keys** tab.
3. Enter the correct IPN URL in PRODUCTION mode via **Settings > Notification rules**.

### 9.3. Making the first production payment

---

We recommend checking the two following points:

- The correct end-to-end functioning of the production environment.  
To do that, make a real transaction of at least €2.  
This transaction can later be canceled via the Merchant Back Office, menu: **Management > Transactions > Transactions in progress** tab. This transaction will not be captured in the bank.  
However, it is recommended to let the transaction be captured at the bank in order to confirm that the merchant's account has been credited. It will then be possible to proceed to a refund.
- The correct functioning of the IPN URL (Instant Payment Notification URL at the end of the payment) specified in the Merchant Back Office.

To do this, do not click **Return to the shop** after a payment.

View the transaction details in the Merchant Back Office and make sure that the IPN URL status is **Sent**.

## 10. OBTAINING HELP

---

Looking for help? Check our FAQ on our website

<https://scelliuspaiement.labanquepostale.fr/doc/en-EN/faq/sitemap.html>

For any technical inquiries or if you need any help, contact [technical support](#).

In view of facilitating the processing of your requests, please specify your shop ID (an 8-digit number) in your query.

This information is available in the “registration of your shop” e-mail or in the Merchant Back Office (**Settings > Shop > Configuration**).