Callback notifications

The payment gateway API allows you to receive callback notifications of order status changes.

General Information

Operations for which notifications can be received

The merchant can receive automatic notifications from the payment gateway about the transactions with the orders indicated in the table below.

Callback notifications are not email notifications or telephone notifications. These are notifications received through the programming interface.

For online lending, only one-phase payment is applied.

Operation	**Payment type**
Holding of funds.	Two-phase payments only.
Debiting funds.	One-phase and two-phase payments.
Cancellation of funds transfer	
Refund.	

Types of notifications

Two types of notifications are available (see table below).

Notification type	**Description**
Notifications without a checksum.	Such notifications contain only information about the order. Potentially, the merchant risks accepting the notification sent by an attacker as genuine.

Notification type	Description
Notifications with a checksum.	Such notifications contain an authentication code in addition to order information. The authentication code is a checksum of order information. This checksum allows you to verify that the callback notification was actually sent by the payment gateway. There are two ways to implement callback notifications with a checksum: • using symmetric cryptography – the same (symmetric) cryptographic key is used to generate a checksum on the gateway side and to verify it on the merchant side; • using asymmetric cryptography – a private key known only to the payment gateway is used to generate the checksum on the payment gateway side, and a public key associated with the private key is used to confirm the checksum, which is known to merchants and can be freely distributed. The public key can be downloaded from personal account (if the user has relevant permissions). For greater security, use of a method in which the checksum is generated using asymmetric cryptography is recommended. To enable the ability to receive checksum notifications and get a private key, contact technical support.

Requirements for SSL certificates of the Merchant Website

If you are using an HTTPS connection to access a store that works with order status notifications, the certificate of the site where the store is located must meet the following requirements (see table below).

Requirement	Description
Length and type of certificate key.	RSA key at least 2048 bits.
The signature algorithm.	Not lower than SHA-256.

Requirement	Description
Supported certification authorities.	Examples of organizations that register certificates are provided below: • Thawte Consulting cc – https://www.thawte.com/; • VeriSign – https://www.verisign.com/; • DigiCert Inc – https://www.digicert.com/; • COMODO CA Limited – https://www.comodo.com/; • GeoTrust Inc. – https://www.geotrust.com/; • GlobalSign – https://www.globalsign.com/; • Trustis Limited – http://www.trustis.com/; • UniTrust – http://www.unitrust.co.uk/; • Let's Encrypt - https://letsencrypt.org/. It is also possible to issue certificates through providers in Russia: • RU-CENTER – http://ssl.ru/; • REG.RU – https://www.reg.ru/ssl-certificate/; • MySSL – https://myssl.ru/. Self-signed certificates are not allowed. The certificate must be signed by a trusted certification authority (see above).

Format of notification URLs

Notification without checksum

```
{merchant-
url}?mdOrder={mdOrder}&orderNumber={orderNumber}&operation={operation}&status
={status}
```

Notification with a checksum

```
{merchant-
url}?mdOrder={mdOrder}&orderNumber={orderNumber}&checksum={checksum}&operatio
n={operation}&status={status}
```

The passed parameters are given in the table below.

The table shows the basic parameters. A number of additional parameters that will be passed in notifications can be specified in personal account.

Parameter	Description
`mdOrder`	Unique order number in the payment gateway system.
`orderNumber`	Unique number (identifier) of the order in the merchant's system.
`checksum`	An authentication code, or checksum, obtained from a set of parameters.
`operation`	Type of operation that was notified: • `approved` – operation of holding the amount; • `declinedByTimeout` – the operation of rejecting the order after it expires; • `deposited` – completion operation; • `reversed` – reversal operation; • `refunded` – refund operation.
`status`	Indicator of success of the operation specified in `operation` parameter: • 1 – operation is successful; • 0 – operation failed.

Example of URL for a notification without checksum

```
https://myshop.ru/callback/?mdOrder=1234567890-098776-234-522&orderNumber=0987&operation=deposited&status=0
```

Example of URL for a notification with checksum

```
https://myshop.ru/callback/?mdOrder=1234567890-098776-234-522&orderNumber=0987&checksum=DBBE9E54D42072D8CAF32C7F660DEB82086A25C14FD813888E231A99E1220AB3&operation=deposited&status=0
```

Custom callback headers

In the admin panel in the merchant settings, you can set custom headers for callback notifications. If they are specified, then such headers will be reflected in the notifications themselves, respectively, for example:

```
'http://kzntest.ru/callback.php', headers={Authorization=token, Content-type=plain /text}, params={orderNumber=349002, mdOrder=5ffb1899-cd1e-7c1e-8750-e98500093c42, operation=deposited, status=1}
```

where `{Authorization=token, Content-type=plain/text}` is a custom header.

Algorithm for processing order status notifications

The sections below present the algorithm for processing order status notifications, depending on the type of such notifications.

Notification without checksum

1. The payment gateway sends the following **GET** HTTP request to the merchant's server.

   ```
   https://myshop.ru/callback/?mdOrder=1234567890-098776-234-522&orderNumber=0987&operation=deposited&status=0
   ```

2. The server sends an HTTP code to the payment gateway `200 OK`.

Notification with a checksum

1. The payment gateway sends the following **HTTP GET** request to the merchant's server; notably:
   - when using symmetric cryptography, the checksum is generated using a key common for the payment gateway and the merchant;
   - when using asymmetric cryptography, the checksum is generated using a private key known only to the payment gateway.

   ```
   http://site.ru/path?amount=123456&orderNumber=10747&checksum=DBBE9E54D42072D8CAF32C7F660DEB82086A25C14FD813888E231A99E1220AB3&mdOrder=3ff6962a-7dcc-4283-ab50-a6d7dd3386fe&operation=deposited&status=1
   ```

   The order of parameters in the notification can be undefined.

2. On the merchant's side, the `checksum` and `sign_alias` parameters are removed from the notification parameter string, and the value of the checksum is saved for verifying the notification's authenticity.
3. The following string is generated from the remaining parameters and their values.

   ```
   parameter_name1; parameter_value1; parameter_name2; parameter_value2; …; parameter_nameN; parameter_valueN;
   ```

   Use semicolon (»;») without a space as the separator. The string should end with semicolon («;») as well.

   Notably, the «parameter_name;parameter_value» pairs must be sorted in direct alphabetical order by parameter name.

   An example of the generated parameter string is shown below.

   ```
   amount;123456;mdOrder;3ff6962a-7dcc-4283-ab50-a6d7dd3386fe;operation;deposited;orderNumber;10747;status;1;
   ```

Please note that the string must also end with a semicolon («;»).

1. The checksum is calculated on the merchant's side, the method of calculation depends on the method of its formation:
    - when using symmetric cryptography – with the help of HMAC-SHA256 algorithm and a private key shared with the payment gateway;
    - when using asymmetric cryptography – with the help of a hashing algorithm that depends on how the key pair is created, and a public key that is associated with a private key located on the payment gateway side.
2. In the resulting checksum string, all lower-case letters are replaced by upper-case letters.
3. The resulting value is compared with the checksum taken earlier from `checksum` parameter.
4. If the checksums match, the server sends to the payment gateway the 200 OK HTTP code.

If the checksums match, this notification is authentic and was sent by the payment gateway. Otherwise, it is likely that the attacker is trying to pass off their notification as a payment gateway notification.

If a response other than the 200 OK HTTP code is returned to the payment gateway, the notification sending is considered to be unsuccessful. In this case, the payment gateway repeats sending the notification (the first attempt is made after 30 seconds, and the next with an interval of 10 minutes) until one of the following conditions is met:

- the payment gateway receives the 200 OK HTTP code in response to the callback notification

or

- there are six unsuccessful attempts to inform in sequence.

When either of the above conditions is met, attempts to send a callback notification about the operation are terminated.

Code examples

Asymmetric Cryptography

Java

```
package ru.bpc.test;
```

```
import org.apache.commons.codec.binary.Base64;

import org.apache.commons.codec.binary.Hex;



import java.io.ByteArrayInputStream;

import java.io.InputStream;

import java.security.Signature;

import java.security.cert.CertificateFactory;

import java.security.cert.X509Certificate;

import java.util.Comparator;

import java.util.Map;

import java.util.stream.Collector;

import java.util.stream.Collectors;

import java.util.stream.Stream;



public class App99 {

    public static void main(String[] args) throws Exception {

        String callbackParamsString = "amount=35000099, sign_alias=SHA-256
with RSA,
checksum=163BD9FAE437B5DCDAAC4EB5ECEE5E533DAC7BD2C8947B0719F7A8BD17C101EBDBE
ACDB295C10BF041E903AF3FF1E6101FF7DB9BD024C6272912D86382090D5A7614E174DC034EB
BB541435C80869CEED1F1E1710B71D6EE7F52AE354505A83A1E279FBA02572DC4661C1D75ABF
5A7130B70306CAFA69DABC2F6200A698198F8, mdOrder=12b59da8-
f68f-7c8d-12b5-9da8000826ea, operation=deposited, status=1";



        Map<String, String> callbackParamsMap =
Stream.of(callbackParamsString.split(","))

                .map(String::trim)

                .map(s -> s.split("="))

                .collect(Collectors.toMap(s -> s[0].trim(), s ->
s[1].trim()));
```

```
        String checksum = callbackParamsMap.get("checksum");

        callbackParamsMap.remove("checksum");

        callbackParamsMap.remove("sign_alias");



        String signString = callbackParamsMap.entrySet().stream()

                .sorted(Map.Entry.comparingByKey(Comparator.naturalOrder()))

                .collect(Collector.of(

                        StringBuilder::new,

                        (accumulator, element) -> accumulator

                                .append(element.getKey()).append(";")

                                .append(element.getValue()).append(";"),

                        StringBuilder::append,

                        StringBuilder::toString

                ));



        String cert =
"MIICcTCCAdqgAwIBAgIGAWAnZt3aMA0GCSqGSIb3DQEBCwUAMHwxIDAeBgkqhkiG9w0BCQEWEWt
6\n" +

"bnRlc3RAeWFuZGV4LnJ1MQswCQYDVQQGEwJSVTESMBAGA1UECBMJVGF0YXJzdGFuMQ4wDAYDVQQ
H\n" +

"EwVLYXphbjEMMAoGA1UEChMDUkJTMQswCQYDVQQLEwJRQTEMMAoGA1UEAxMDUkJTMB4XDTE3MTI
w\n" +

"NTE2MDEyMFoXDTE4MTIwNTE2MDExOVowfDEgMB4GCSqGSIb3DQEJARYRa3pudGVzdEB5YW5kZXg
u\n" +

"cnUxCzAJBgNVBAYTAlJVMRIwEAYDVQQIEwlUYXRhcnN0YW4xDjAMBgNVBAcTBUthemFuMQwwCgY
D\n" +

"VQQKEwNSQlMxCzAJBgNVBAsTAlFBMQwwCgYDVQQDEwNSQlMwgZ8wDQYJKoZIhvcNAQEBBQADgY0
A\n" +

"MIGJAoGBAJNgxgtWRFe8zhF6FE1C8s1t/dnnC8qzNN+uuUOQ3hBx1CHKQTEtZFTiCbNLMNkgWtJ
/\n" +
```

```
"CRBBiFXQbyza0/Ks7FRgSD52qFYUV05zRjLLoEyzG6LAfihJwTEPddNxBNvCxqdBeVdDThG81zC
0\n" +

"DiAhMeSwvcPCtejaDDSEYcQBLLhDAgMBAAEwDQYJKoZIhvcNAQELBQADgYEAfRP54xwuGLW/Cg0
8\n" +

"ar6YqhdFNGq5TgXMBvQGQfRvL7W6oH67PcvzgvzN8XCL56dcpB7S8ek6NGYfPQ4K2zhgxhxpFED
H\n" +

"PcgU4vswnhhWbGVMoVgmTA0hEkwq86CA5ZXJkJm6f3E/J6lYoPQaKatKF24706T6iH2htG4Bkjr
e\n" +

                "gUA=";



        byte[] b = Base64.decodeBase64(cert);

        CertificateFactory certFactory =
CertificateFactory.getInstance("X.509");



        InputStream in = new ByteArrayInputStream(b);

        X509Certificate x509Cert =
(X509Certificate)certFactory.generateCertificate(in);



        Signature sig = Signature.getInstance("SHA512withRSA");

        sig.initVerify(x509Cert.getPublicKey());



        sig.update(signString.getBytes());

        boolean verifies =
sig.verify(Hex.decodeHex(checksum.toLowerCase().toCharArray()));

        System.out.println("signature verifies: " + verifies);

    }

}
```

Symmetric cryptography

Java

```
import org.apache.commons.codec.binary.Hex;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.StandardCharsets;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

public class Example {

    public static String generateHMacSHA256(final String key, final String
data) throws InvalidKeyException, NoSuchAlgorithmException {

        final Mac hMacSHA256 = Mac.getInstance("HmacSHA256");
        byte[] hmacKeyBytes = key.getBytes(StandardCharsets.UTF_8);

        final SecretKeySpec secretKey = new SecretKeySpec(hmacKeyBytes,
"HmacSHA256");
        hMacSHA256.init(secretKey);

        byte[] dataBytes = data.getBytes(StandardCharsets.UTF_8);
        byte[] res = hMacSHA256.doFinal(dataBytes);

        return new String(Hex.encodeHex(res));
    }

    public static void main(String[] args) throws NoSuchAlgorithmException,
InvalidKeyException {
        String secretToken = "123";
        String message = "amount;1500;mdOrder;ed6f3abf-cea0-427e-
afdf-0ba43ead124f;operation;deposited;orderNumber;89312;status;1;";

        String signature = Expample.generateHMacSHA256(secretToken,
message).toUpperCase();
        System.out.println(signature);
    }
```

Symmetric cryptography

PHP

```
<?php
```

```
$data = 'amount;123456;mdOrder;3ff6962a-7dcc-4283-ab50-
a6d7dd3386fe;operation;deposited;orderNumber;10747;status;1;';
$key = 'yourSecretToken';
$hmac = hash_hmac ( 'sha256' , $data , $key);

echo "[$hmac]\n";
?>
```

1. Add the string as the `data` variable value.
2. Add the private key in `key` variable.
3. Function `hash_hmac` (`'sha256'`, `$data`, `$key`) calculates the checksum from the string using the private key by the SHA-256 algorithm.
4. Save function output to `hmac` variable.
5. Display the function output using the `echo` function.
6. Compare this value with the value reported in the order status notification.

Asymmetric Cryptography

PHP

```
<?php
// data from response
$data = 'amount;35000099;mdOrder;12b59da8-
f68f-7c8d-12b5-9da8000826ea;operation;deposited;status;1;';
$checksum =
'9524FD765FB1BABFB1F42E4BC6EF5A4B07BAA3F9C809098ACBB462618A9327539F975FEDB4C
F6EC1556FF88BA74774342AF4F5B51BA63903BE9647C670EBD962467282955BD1D57B16935C9
56864526810870CD32967845EBABE1C6565C03F94FF66907CEDB54669A1C74AC1AD6E39B67FA
7EF6D305A007A474F03B80FD6C965656BEAA74E09BB1189F4B32E622C903DC52843C454B7ACF
76D6F76324C27767DE2FF6E7217716C19C530CA7551DB58268CC815638C30F3BCA3270E1FD44
F63C14974B108E65C20638ECE2F2D752F32742FFC5077415102706FA5235D310D4948A780B08
D1B75C8983F22F211DFCBF14435F262ADDA6A97BFEB6D332C3D51010B';

// your public key (e.g. SHA-512 with RSA)
// if you have a CERT, please see openssl_get_publickey()
$publicKey = <<<EOD
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwtuGKbQ4WmfdV1gjWWys
5jyHKTWXnxX3zVa5/Cx5aKwJpOsjrXnHh6l8bOPQ6Sgj3iSeKJ9plZ3i7rPjkfmw
qUOJ1eLU5NvGkVjOgyi11aUKgEKwS5Iq5HZvXmPLzu+U22EUCTQwjBqnE/Wf0hnI
wYABDgc0fJeJJAHYHMBcJXTuxF8DmDf4DpbLrQ2bpGaCPKcX+04POS4zVLVCHF6N
6gYtM7U2QXYcTMTGsAvmIqSj1vddGwvNGeeUVoPbo6enMBbvZgjN5p6j3ItTziMb
Vba3m/u7bU1dOG2/79UpGAGR10qEFHiOqS6WpO7CuIR2tL9EznXRc7D9JZKwGfoY
/QIDAQAB
-----END PUBLIC KEY-----
EOD;
```

```
$binarySignature = hex2bin(strtolower($checksum));
$isVerify = openssl_verify($data, $binarySignature, $publicKey,
OPENSSL_ALGO_SHA512);
if ($isVerify == 1) {
    echo "signature ok\n";
} elseif ($isVerify == 0) {
    echo "bad (there's something wrong)\n";
} else {
    echo "error checking signature\n";
}
?>
```