Integrations Support Team

JavaScript API One Time Key Generation

Version 1.2

Change Log

Version	Description	Date	Change by
1.0	Initial Version	April 2020	Ian Daley
1.1	Updated for Production	May 2020	Ian Daley
1.2	Minor corrections	June 2020	Ian Daley

Outline

The current Ezidebit JavaScript API is intended for use in secure environments that are protected by user accounts. In some instances though there is a desire to implement this feature to accept payments through websites outside a user login.

To facilitate this method of payment it is necessary to provide a number of means of protecting the public key and to resist automated card testing fraud attacks.

Protecting the public key

The public key used in an insecure matter can be exposed to malicious use. This can lead to the key being scraped or the site automated by a script to process thousands of fraudulent transactions in a very short period of time.

The solution is to protect the public key by generating a *one-time key* on the server side that can be substituted for the public key on the front end. This will then prevent the key from being taken off site and used in an off site exploit, or for the page itself being driven by a bot.

To extend an existing integration the following new steps are required:

1. Generate a *one-time key* on the server side using the public key.
2. On the front end, where normally the public key would be used, replace this with the *one-time key*.
3. Prevent attempted re-use of the *one-time key*.

One time key generation

To support *one-time key* generation the TestPublicKey endpoint has been extended to support generation of such a key. As part of this the endpoint has been updated to a new API Base Endpoint to support the extensions. Please use this endpoint when using these API Extensions.

Base Sandbox Endpoint:
https://apix.demo.ezidebit.com.au/public-rest/v3-5/

Base Production Endpoint:
https://apix.ezidebit.com.au/public-rest/v3-5/

The REST API endpoint *TestPublicKey* is used to determine if a public key is active and links to an active account. It is ideally used prior to issuing real time payment requests to ensure the key is active for the account. In this instance we have extended it to provide a *one-time key* on a positive response that can be used in transactions on the front end.

Additional functionality has been added to lock the one time key to a specific IP address. This ensures the key cannot be distributed to a bot at another IP address or host. When using this functionality, ensure your web server proxy is passing you the real IP address of the end user. If you are unsure do not use this option.

It is a **REST GET** Method and uses the following endpoint and parameters:

Sandbox Endpoint:
https://apix.demo.ezidebit.com.au/public-rest/v3-5/TestPublicKey

Production Endpoint:
https://apix.ezidebit.com.au/public-rest/v3-5/TestPublicKey

Parameters

Parameter	Description
PublicKey	The public key you wish to test
lockIP	IP address to restrict the one time key to

Example query

https://apix.demo.ezidebit.com.au/public-rest/v3-5/TestPublicKey?PublicKey=8033150E-9882-4502-AB85-71757192AC48

Response

The response is JSON formatted data. A success means the public key is active and linked to an active account.

Example response:


```
{
  "Data": "success",
  "Error": 0,
  "ErrorMessage": null,
  "OneTimeKey":
"d8e1869f53b0c4f42f4d8e9413754f910453c110b6bd977bd8efeb64b9569f9d"
}
```


Field	Description
Data	'Success' for success or 'null' for failure
Error	'0' for success, '138' for invalid public key
ErrorMessage	'null' for success, string description of error 'Invalid Public Key'
OneTimeKey	If present this string will contain the one-time key

Example obtaining the one-time key

Example PHP implementation to return one time key.

```
<?php
$endpoint =
"https://apix.demo.ezidebit.com.au/public-rest/v3-5/TestPublicKey"
;
$querystring = "?PublicKey=";
$myPublicKey = "8033150E-9882-4502-AB85-71757192AC48"; // Replace
with your key
$url = $endpoint . $querystring . $myPublicKey;
$oneTimeKey = json_decode(file_get_contents($url))->OneTimeKey;
```

This key is then substituted in the front end where the public key would normally be implemented.

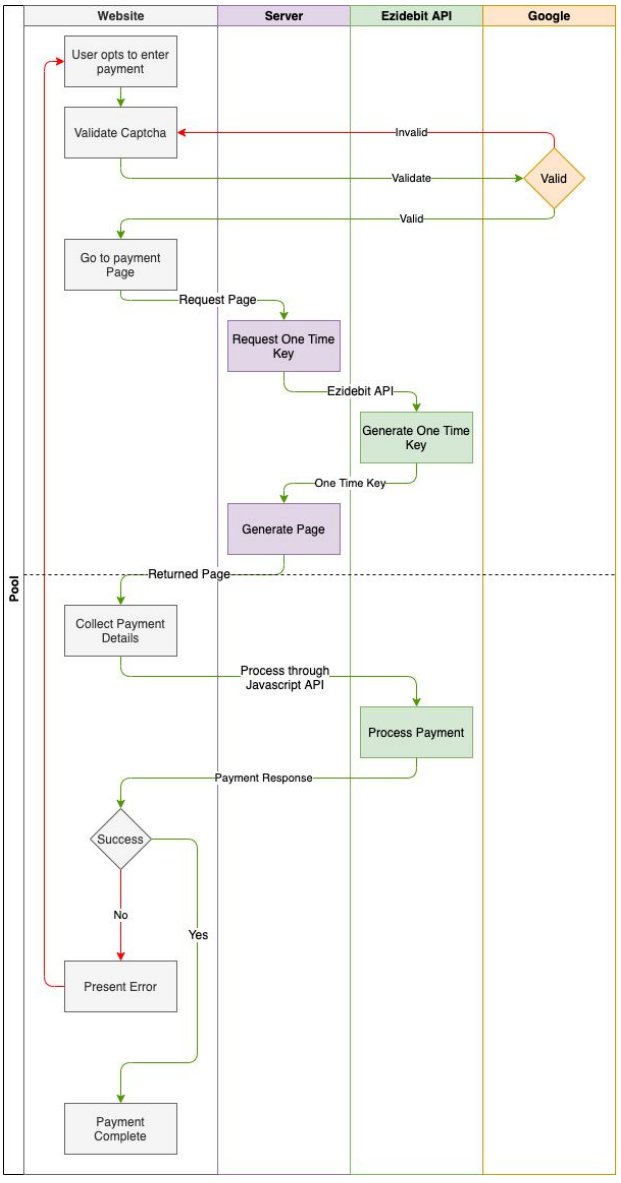
Best practices for generating the one-time key

In generating the one time key, take care to ensure the site hosting it does not become a OneTimeKey generator service for card testing. To do this it needs to be used in conjunction with Captcha and the following other steps:

1. The *one-time key* **must be** generated on the server side.
2. The *one-time key* generation must be protected by Captcha or other means to restrict offsite generation (presence of a session ID for example).
3. Refreshing the page hosting the Javascript payment form **should not** generate a new *one-time key* without re-validating the page first;- instead the user should be redirected to the previous page requiring completion of Captcha or other validation to generate the key.
4. **Do not** create a simple endpoint to get this key - it must be accompanied with validation on the back end.
5. In the event of a payment error, the key must be generated again repeating the validation routine.
6. The key expires after 10 minutes automatically.
7. The key expires immediately after use.

Example payment flows

Two page back end validation using a *one-time key* and Captcha



This is probably the simplest method to implement used in conjunction with a Captcha.

The first page presents the payment finalisation options before taking payment method, and asks the user to validate a Captcha before moving to the next screen.

Then when posting to the server and requesting the next page the one time key is obtained and added in the page that collects the payment details.

When the page is loaded, any retained Captcha variables are cleared.

In the event of a failed payment the user is taken back to the previous page where they will be required to complete the Captcha again and any variables cleared.

If the payment page is refreshed,it should automatically be directed back to the previous page to revalidate the user.

This is the simplest method but does have a down side in that it requires the payer to go back to the previous page on error.

One page back end validation using a *one-time key* and Captcha

This method is a little more complex and requires the development of a service that can be called to validate the Captcha and *one-time key* in the back end.

The payer is presented with the option to pay and completes their payment details. They must complete a Captcha to process payment.

On successful completion of the Captcha, a call is made to the server to both validate the Captcha with Google and then subsequently generate a *one-time key* through Ezidebit.

The *one-time key* is returned and substituted into the payment variables and submitted to the system with payment.

In the event of an error the user is prompted with the error message and required to resubmit their details, including completing the Captcha.

This has proved a robust single page solution for ensuring the card is not exploited by card testing attacks.