WePay Signer for PHP

Release

January 05, 2017

Contents

This is a WePay-specific wrapper around the skyzyx/signer package.

This project uses Semantic Versioning for managing backwards-compatibility.

CHAPTER 1

User guide

1.1 Installation

Composer is the only method for installation that our team will support. Other installation methods may be used at your own risk.

Using Composer:

```
composer require wepay/signer-php=^1.0
```

And include it in your scripts:

```
require_once 'vendor/autoload.php';
```

1.2 Usage

1.2.1 Generate a signature

```
use WePay\Signer\Signer;

$client_id = 'your_client_id';
$client_secret = 'your_client_secret';

$signer = new Signer($client_id, $client_secret);
$signature = $signer->sign([
    'token'        => $your_token,
    'page'         => $wepay_page_to_visit,
    'redirect_uri' => $partner_page_to_return_to,
]);

$signature = wordwrap($signature, 64, "\n", true);
#=> dfbffab5b6f7156402da8147886bba3eba67bd5baf2e780ba9d39e8437db7c47
#=> 35e9a0b834aa21ac76f98da8c52a2a0cd1b0192d0f0df5c98e3848b1b2e1a037
```

1.2.2 Generate all of the query string parameters for the request

```
$querystring = $signer->generateQueryStringParams([
    'token'        => $your_token,
    'page'          => $wepay_page_to_visit,
    'redirect_uri' => $partner_page_to_return_to,
]);

// This will return query strings in following format

#=> client_id=your_client_id&
#=> page=https%3A%2F%2Fwepay.com%2Faccount%2F12345&          // in url encoded format
#=> redirect_uri=https%3A%2F%2Fpartnersite.com%2Fhome&      // in url encoded format
#=> stoken=dfbffab5b6f7156402da8147886bba3eba67bd5baf2e780ba9d39e8437db7c47...&
#=> token=acb1b5b8-af32-5356-bd2a-5bac74366e4c
```

If you are generating the query string parameters by yourself, make sure that page and redirect_uri are in url encoded format

1.3 Signing Algorithm

Documentation for how to implement your own Request Signer for languages where we do not already have an Request Signer SDK. This implementation is based on a stripped-down version of AWS Signature v4, and uses SHA-512 for hashing.

1.3.1 scope

Source:

```
# WePay/client_id/signer
scope = "WePay/" + client_id + "/signer"
```

Example:

```
WePay/client_id/signer
```

1.3.2 canonical_context

Source:

```
# Your input data, as key-value pairs
key_value_pairs = [
    "client_id": client_id,
    "client_secret": client_secret,
    "token": your_token,
    "page": wepay_page_to_visit,
    "redirect_uri": partner_page_to_return_to,
]

# 1. Go though each of the key-value pairs
# 2. Convert all keys & values to lowercase
# 3. Turn the value into a string of "key=value", indexed by the lowercased key
for (key, value) in key_value_pairs {
    lower_key = key.lowercase()
    lower_value = value.lowercase()

    list_of_lowercase_keys.push(lower_key)
    sanitized_key_value_pairs[lower_key] = lower_key + "=" + lower_value
}

# Ensure that all keys are in true alphabetical order
sanitized_key_value_pairs = sanitized_key_value_pairs.sort_by_keys()

# Get a string, consisting of a list of keys delimited by semicolons
signed_headers_string = list_of_lowercase_keys.join_as_string_with_delimiter(";")

# Get the canonical "context" for the signature that will be used for signing
canonical_context = sanitized_key_value_pairs.join_as_string_with_delimiter("\n")
                  + "\n\n"
                  + signed_headers_string
```

Example:

```
client_id={client_id}
client_secret={client_secret}
page={wepay_page_to_visit}
redirect_uri={partner_page_to_return_to}
token={your_token}

client_id;client_secret;page;redirect_uri;token
```

1.3.3 string_to_sign

Source:

```
# Determine the string that will ultimately be signed
# hash_sha512() produces a 128-character hexadecimal hash
string_to_sign = "SIGNER-HMAC-SHA512" + "\n"
               + "WePay" + "\n"
               + client_id + "\n"
               + hash_sha512(scope) + "\n"
               + hash_sha512(canonical_context)
```

Example:

```
SIGNER-HMAC-SHA512
WePay
{client_id}
EXAMPLE01452722F2366BC72EBBEF736D832F06765373D8445514573A5B411ABA042D0A97EDDA068187A8BE5B581EB24E0EA
EXAMPLE6E214454FAC7A639C3F793B7991EB98A755C701B45BB5AA4DE328455B5B5F072D14CF828A63BBE3CA392D397609AF
```

1.3.4 signing_key

Source:

```
# raw_hmac_sha512(data, secret) returns raw binary data
self_key_sign  = raw_hmac_sha512("WePay", client_secret)
client_id_sign = raw_hmac_sha512(client_id, self_key_sign)
salt           = raw_hmac_sha512("signer", client_id_sign)

# Convert the raw binary data to a hexadecimal value
signing_key = hex_encode(salt)
```

Example:

```
EXAMPLE6E214454FAC7A639C3F793B7991EB98A755C701B45BB5AA4DE328455B5B5F072D14CF828A63BBE3CA392D397609AF
```

1.3.5 signature

Source:

```
# raw_hmac_sha512(data, secret) returns raw binary data
signature = raw_hmac_sha512(string_to_sign, signing_key)
```

Example:

```
EXAMPLE01452722F2366BC72EBBEF736D832F06765373D8445514573A5B411ABA042D0A97EDDA068187A8BE5B581EB24E0EA
```

1.4 Debug Logging

NOTE: You should *only* use logging during development — never in production.

Signer implements the PSR-3 `Psr\Log\LoggerAwareInterface`. Because of this, you can inject any PSR-3-compatible logging package, and Signer will use it to log `DEBUG`-level messages.

```
use Monolog\Logger;
use Monolog\Handler\StreamHandler;
use WePay\Signer\Signer;

// create a log channel
$log = new Logger('name');
$log->pushHandler(new StreamHandler('path/to/your.log', Logger::DEBUG));

// inject a logger
$signer = new Signer();
$signer->setLogger($log);

$signer->sign( ... );
```

1.5 API Reference

The API Reference is generated by a tool called Sami. You can generate updated documentation by running the following command in the root of the repository.

```
make docs
```

You can view the API reference at https://wepay.github.io/signer-php/.

1.6 Testing

Firstly, run `composer install --optimize-autoloader` to download and install the dependencies.

You can run the tests as follows:

```
make test
```

You can check on the current test status at https://travis-ci.org/wepay/signer-php.

1.7 Deploying (WePay team)

1. The `Makefile` (yes, `Makefile`) has a series of commands to simplify the development and deployment process.
2. Also install Chag. This is used for managing the CHANGELOG and annotating the Git release correctly.

1.7.1 Updating the CHANGELOG

Make sure that the `CHANGELOG.md` is human-friendly. See http://keepachangelog.com if you don't know how.

1.7.2 make

Running `make` by itself will show you a list of available sub-commands.

```
$ make
all
docs
install
pushdocs
tag
test
version
```

1.7.3 make pushdocs

You will need to have write-access to the `wepay/signer-php` repository on GitHub. You should have already set up:

- Your SSH key with your GitHub account.
- Had your GitHub user given write-access to the repository.

Then you can run:

```
make pushdocs
```

You can view your changes at https://wepay.github.io/signer-php/.

1.7.4 make version

This allows you to set the version number for the next release.

1.7.5 make tag

This will leverage Chag to generate a commit for the tag.

```
make tag
```

1.7.6 Drafting a GitHub release

1. Go to https://github.com/wepay/signer-php/tags
2. Find the new tag that you just pushed. Click the ellipsis (`...`) to see the commit notes. Copy these.
3. To the right, choose *Add release notes*. Your *Tag version* should be pre-filled.
4. The *Release title* should match your *Tag version*.
5. Inside *Describe this release*, paste the notes that you copied on the previous page.
6. Choose *Publish release*.
7. Your release should now be the latest. https://github.com/wepay/signer-php/releases/latest

1.8 Contributing

Here's the process for contributing:

1. Fork WePay Signer for PHP to your GitHub account.
2. Clone your GitHub copy of the repository into your local workspace.
3. Write code, fix bugs, and add tests with 100% code coverage.
4. Commit your changes to your local workspace and push them up to your GitHub copy.
5. You submit a GitHub pull request with a description of what the change is.
6. The contribution is reviewed. Maybe there will be some banter back-and-forth in the comments.
7. If all goes well, your pull request will be accepted and your changes are merged in.

1.9 Authors, Copyright & Licensing

- Copyright (c) 2015–2016 WePay.

See also the list of contributors who participated in this project.

Licensed for use under the terms of the Apache 2.0 license.

1.10 Coding Standards

PSR-0/1/2 are a solid foundation, but are not an entire coding style by themselves. I have taken the time to document all of the nitpicky patterns and nuances of my personal coding style. It goes well-beyond brace placement and tabs vs. spaces to cover topics such as docblock annotations, ternary operations and which variation of English to use. It aims for thoroughness and pedanticism over hoping that we can all get along.

https://github.com/skyzyx/php-coding-standards