

A library for generating and validating passwords

📄 201 commits

🌿 2 branches

📦 4 releases

👥 6 contributors

Branch: master ▾


New pull request

Create new file

Upload files

Find file

Clone or download ▾

 ircmaxell Merge pull request #24 from geekcom/master ...	Latest commit 692d168 on Aug 24, 2017
📁 build	Enable travis build 5 years ago
📁 examples	Refactor password implementation to support passing an array of param... 5 years ago
📁 lib/PasswordLib	Update PasswordLib.php 5 years ago
📁 test	Updating BigMath test. 5 years ago
📄 .gitignore	Update VFS dependency to use composer --dev 5 years ago
📄 .travis.yml	Update VFS dependency to use composer --dev 5 years ago
📄 README.markdown	update .gitignore, fix README 6 months ago
📄 composer.json	Update VFS dependency to use composer --dev 5 years ago
📄 composer.lock	Update VFS dependency to use composer --dev 5 years ago
📄 phpunit.xml.dist	Enable travis build 5 years ago

📖 README.markdown

🔗 PHP-PasswordLib

Build Status

build passing

Version

The current version is considered Beta. This means that it is *ready enough* to test and use, but beware that you should update frequently.

As this software is **BETA**, Use at your own risk!

About

PHP-PasswordLib aims to be an all-inclusive cryptographic library for all cryptographic needs. It is meant to be easy to install and use, yet extensible and powerful enough for even the most experienced developer.

Installation

PasswordLib supports multiple installation methods.

PHAR

From the [downloads](#) tab, download the latest phar build. Then, just require the phar in your code!

```
require_once '/path/to/PasswordLib.phar';
```

Composer

Add a `composer.json` file to your project with the following:

```
{
  "require": {
    "PasswordLib/PasswordLib": "*"
  }
}
```

Then, inside that folder, just run `php composer.phar install`.

Then, in your code, just use the composer autoloader:

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

That's it!

Usage

Most use-cases can simply use the root `PasswordLib` class.

```
$lib = new PasswordLib\PasswordLib();
$hash = $lib->createPasswordHash($password);
$boolean = $lib->verifyPasswordHash($password, $hash);
```

By default, `createPasswordHash` will create a blowfish hash, which is the most secure available. To create other types, just pass the prefix of the type as a second parameter.

So, to create a drupal hash:

```
$hash = $lib->createPasswordHash($password, '$S$');
```

Or to create a SHA512 hash:

```
$hash = $lib->createPasswordHash($password, '$6$');
```

It will automatically create a secure salt, and generate the hash.

You can also specify options for the hash. So to use a bcrypt cost of 12,

```
$hash = $lib->createPasswordHash($password, '$2a$', array('cost' => 12));
```

`verifyPasswordHash` will attempt to determine what type of hash is passed in. So one API call can verify multiple types of hashes. This allows for applications to be portable and authenticate against multiple databases with one API.

The `PasswordLib` class has other API methods for getting random data. Two of particular use are `getRandomNumber` and `getRandomToken`.

- `getRandomNumber([$min] [, $max]` - gets a secure random integer between the given parameters.
- `getRandomToken($size)` returns a random string using base64 characters (`a-zA-Z0-9./`). This is useful for generating nonce's and tokens to send to clients.

The library also contains other methods for generating random data and hashing data, so look around!

Design Goals

- 100% Portable

That means there are no hard (meaning required) dependencies on extensions or non-standard server configurations. Certain configurations will have better performance for certain features, but all configurations should be supported.

- **Well Designed**

The code will use industry standard design patterns as well as follow guidelines for clean and testable code.

- **Well Tested**

That means that the code should be well covered by unit tests. In addition to unit tests, standard test vectors should be run for custom implementations of algorithms to ensure proper behavior.

- **Easy To Install**

PHP-PasswordLib will support three install methods. The first method is a pear based installer. The second is a single file PHAR archive. The third is support via Composer.

- **Easy To Use**

One goal of this system is to provide a simple interface which has secure defaults for standard cryptographic needs (Random token generation, password hashing and verifying, etc). If more power is needed, additional layers of abstraction are available to wire together however is needed.

- **Easy To Extend**

The library should be very easy to extend and add new functionality.

Features

Optional Autoloading

If you include PasswordLib via a PHAR package, it will automatically autoload all of the classes for you, no extra step necessary. Simply:

```
require 'path/to/PasswordLib.phar';
```

If you include PasswordLib via a filesystem install, you can use the internal autoloader by either loading the bootstrap.php file, or loading the PasswordLib.php file

```
require_once 'path/to/PasswordLib/bootstrap.php'
```

or

```
require_once 'path/to/PasswordLib/PasswordLib.php'
```

You can also use any [PSR-0] [3](#) autoloader. PasswordLib will automatically detect if an autoloader is setup for its namespace, and will not declare its own if it finds one (it does this by testing if the class PasswordLib\Core\AutoLoader can be found. If so, that means that an autoloader was declared already. If not, it loads the core implementation).

```
$classLoader = new SplClassLoader('PasswordLib', 'path/to/');  
$classLoader->register();
```

Note that the path you supply is the directory which contains the *PasswordLib* directory. Not the PasswordLib directory itself.

Secure Random Number/String Generation

PHP-PasswordLib implements a method specified in [RFC 4086 - Randomness Requirements for Security] [2](#). Basically, it generates randomness from a number of pseudo random sources, and "mixes" them together to get better quality random data out. When you specify the "strength" of random generator, you are actually telling the system which sources you would like to use. The higher the strength, the slower and potentially more fragile the source it will use.

The mixing function is also dependent upon the strength required. For non-cryptographic numbers, a simple XOR mixing function is used (for speed). As strength requirements increase, it will use a SHA512 based mixing function, then a DES based mixing function and finally an AES-128 based mixing function at "High" strength.

And all of this is hidden behind a simple API.

To generate user-readable strings, you can use the PasswordLib class (which generates medium strength numbers by default):

```
$crypt = new PasswordLib\PasswordLib;
$token = $crypt->getRandomToken(16);
```

Or you can use the core generator to get more control:

```
$factory = new PasswordLib\Random\Factory;
$generator = $factory->getHighStrengthGenerator();
$token = $generator->generateString(16);
```

To generate salts, simply use PasswordLib::getRandomString() or Generator::generate()

Password Hashing And Validation

A number of password hashing algorithms are supported. When creating a new hash, the algorithm is chosen via a prefix (a CRYPT() style prefix). The library will do the rest (salt generation, etc):

```
$crypt = new PasswordLib\PasswordLib;
$hash = $crypt->createPasswordHash($password, '$2a$'); // Blowfish
$hash = $crypt->createPasswordHash($password, '$S$'); // Drupal
```

When validating password hashes, where possible, the library will actually auto-detect the algorithm used from the format and verify. That means it's as simple as:

```
$crypt = new PasswordLib\PasswordLib;
if (!$crypt->verifyPasswordHash($password, $hash)) {
    //Invalid Password!
}
```

You can bypass the auto-detection and manually verify:

```
$hasher = new PasswordLib\Password\Implementation\Joomla;
$hash = $hasher->create($password);
if (!$hasher->verify($password, $hash)) {
    //Invalid Hash!
}
```

Specifications

- Supported Password Storage Functions
 - **APR1** - Apache's internal password function
 - **Blowfish** - BCrypt
 - **Crypt** - Crypt DES hashing
 - **Drupal** - Drupal's SHA512 based algorithm
 - **Hash** - Raw md5, sha1, sha256 and sha512 detected by length
 - **Joomla** - Joomla's MD5 based algorithm
 - **Crypt MD5** - Support for Crypt's MD5 algorithm
 - **PBKDF** - A PBKDF implementation (which supports any supported password based key derivation)
 - **PHPASS** - An implementation of the portable hash from the PHPASS library
 - **PHPBB** - PHPBB's MD5 based algorithm
 - **Crypt SHA256** - Crypt's SHA256 algorithm
 - **Crypt SHA512** - Crypt's SHA512 algorithm

- Supported Random Number Sources
 - **CAPICOM** - A COM object method call available on Windows systems
 - **MTRand** - Generation based upon the mt_rand() functions
 - **MicroTime** - A low entropy source based upon the server's microtime
 - **Rand** - A low entropy source based upon rand()
 - **URandom** - Generation from the system's /dev/urandom source
 - **UniqID** - A low entropy source based upon uniqid()

Library Dependencies:

The only dependency PHP-PasswordLib has to use as a library is the PHP version. It is made to be completely independent of extensions, implementing functionality natively where possible.

Required

- PHP >= 5.3.2

Optional

- [MCrypt] [1](#) Support Compiled In

Build (Testing) Dependencies:

These dependencies are necessary to build the project for your environment (including running unit tests, packaging and code-quality checks)

Pear Dependencies

- PDepend Channel (pear.pdepend.org)
 - pdepend/PHP_Depend >= 0.10.0
- Phing Channel (pear.phing.info)
 - phing/Phing >= 2.4.0
- PHPMD Channel (pear.phpmd.org)
 - phpmd/PHP_PMD >= 1.1.0
- PHPUnit Channel (pear.phpunit.de)
 - phpunit/PHPUnit >= 3.5.0
 - phpunit/PHP_CodeBrowser >= 1.0.0
 - phpunit/phpcpd >= 1.3.0
 - phpunit/phploc >= 1.6.0
- PHP-Tools Channel (pear.php-tools.net)
 - pat/vfsStream >= 0.8.0
- Default Pear Channel
 - pear/PHP_CodeSniffer >= 1.3.0
 - pear/PHP_UML >= 1.5.0

Note: You can install all of them with the following commands:

```
pear channel-discover pear.pdepend.org
pear channel-discover pear.phing.info
pear channel-discover pear.phpmd.org
pear channel-discover pear.phpunit.de
```

```
pear channel-discover pear.php-tools.net
pear channel-discover components.ez.no
pear channel-discover pear.symfony-project.com
```

```
pear install pdepend/PHP_Depend
pear install phpmd/PHP_PMD
pear install pat/vfsStream
pear install PHP_CodeSniffer
pear install PHP_UML
pear install phpunit/PHPUnit
pear install phpunit/PHP_CodeBrowser
pear install phpunit/phpcpd
pear install phpunit/phploc
pear install phing/Phing
```

PHP Dependencies

- PHP \geq 5.3.2
 - `php.ini` Settings:
 - `phar.readonly = Off`
- PHP Extensions
 - XDebug
 - MCRYPT
 - Hash (usually enabled)
 - Phar
 - Zip (For Packaging)
 - BZ2 (For Packaging)
 - XSL (For Documentation)

Security Vulnerabilities

If you have found a security issue, please contact the author directly at me@ircmaxell.com.