How to work with users' passwords and how to securely hash passwords in PHP?

5 min read · Last change October 1, 2017

When you save users' passwords onto a database, you should NEVER store them in plain-text due to security and privacy concerns. A database where users' passwords are stored could be compromised at some point in the future, and by hashing them, at the very least, it will be more difficult for an attacker to determine the original passwords of the affected users.

```
<?php // Plain-text password example $password = 'secretcode';
```

Cryptography is a large and very complex field for many people, so a good rule of a thumb would be to leave it to the experts.

One of the once most used ways of hashing passwords, now considered extremely unsafe, was to use the `md5()` function which calculates the md5 hash of a string. Hashing passwords with md5 (or sha1, or even sha256) is not safe anymore, because these hashes can be decrypted very quickly.

```
<?php // Plain-text password $password = 'secretcode'; // Hashing the password with md5 $md5 =
md5($password);
```

A common solution to preventing decryption is using a salt.

```
<?php // Plain-text password $password = 'secretcode'; // Add a random number of random characters
(the salt) $salt = '3x%%$bf83#dls2qgdf'; // Hash salt and password together $md5 =
md5($salt.$password);
```

This is still not good enough though (rainbow tables).

The right way to hash passwords in PHP

Currently, the right way to hash passwords is to use the latest PHP version and its native passwords hashing API, which provides an easy to use wrapper around the crypt function.

An example for using the native PHP password hashing API:

```
<?php // Plain-text password $password = 'secretcode'; $options = ['cost' => 12]; echo
password_hash($password, PASSWORD_DEFAULT, $options);
```

The `password_hash()` function currently provides three different algorithm options. `PASSWORD_DEFAULT`, `PASSWORD_BCRYPT`, and (as of PHP >= 7.2.0) `PASSWORD_ARGON2I`. Currently, the options `PASSWORD_DEFAULT` and `PASSWORD_BCRYPT` will both result in the use of the BCRYPT hashing algorithm, making them essentially the same. `PASSWORD_ARGON2I` will result in the use of the Argon2 hashing algorithm. As cryptography and the PHP language as a whole progress, there'll likely be other, new types of algorithms supported. `PASSWORD_DEFAULT` will likely be changed in the future as recommendations for the best hashing algorithm to use evolve and as new hashing algorithms become available, and so, generally, `PASSWORD_DEFAULT` is the best option to choose when hashing passwords.

The type of field used for storing passwords in databases should be `varchar(255)` for future-proof algorithm changes.

Using your own salt is not recommended. It's generally recommended to use a bullet-proof without setting your own salts, allowing

the `password_hash()` function handle this itself (salts are randomly generated by default when using `password_hash()`).

Another important option to mention is the `cost`, which controls the hash speed. On servers with better resources, `cost` can be increased. There's a script for calculating the cost for your environment in the PHP manual. It's good security practice is to try increasing this to a higher value than the default (`10`).

Verifying passwords can be done with password_verify():

```
<?php // This is the hash of the password in example above. $hash =
'$2y$12$VD3vCfuHcxU0zcgDvArQSOlQmPv3tXW0TWoteV4QvBYL66khev0oq'; if (password_verify('secretcode',
$hash)) { echo 'Password is valid!'; } else { echo 'Invalid password.'; }
```

Another useful function is password_needs_rehash(), which checks if the given hash matches the given options. This comes in handy in the event of server hardware upgrades and when increasing the `cost` option is possible.

The hash string returned by `password_hash()` consists of the following parts:

```
$2y$10$VCbjoi9DnyQyVxf4/RRoFeyOCeMPnCitAG07ZRpivwglmpbP0jOdW | | | | | | |_ password (length
depends on algorithm) | | | | | |_ salt (22 characters) | | | |_ cost (2 characters) | |_
algorithm (length depends on algorithm)
```

So, you can extract raw password hash components like this:

```
$hash = '$2y$10$VCbjoi9DnyQyVxf4/RRoFeyOCeMPnCitAG07ZRpivwglmpbP0jOdW'; list(, $algo, $cost,
$salt_and_password) = explode('$', $hash); $salt = substr($salt_and_password, 0, 22); $password =
substr($salt_and_password, 22);
```

Or simply use `password_get_info()` to get more readable information.

```
$hash = '$2y$10$VCbjoi9DnyQyVxf4/RRoFeyOCeMPnCitAG07ZRpivwglmpbP0jOdW';
print_r(password_get_info($hash));
```

Output

```
Array ( [algo] => 1 [algoName] => bcrypt [options] => Array ( [cost] => 10 ) )
```

Password hashing in older PHP versions (PHP <= 5.5)

In case you're still using some older PHP version, there is a way to properly secure passwords. Since PHP version > 5.3.7, you can use the PHP library password_compat. The PHP library `password_compat` works in exactly the same way as the native PHP password hashing API, so when you upgrade to the latest PHP version, you won't need to refactor your code.

For PHP versions below 5.3.6, phpass might be a good solution, but try to avoid these and use the native password hashing API instead.

Password hashing in open source projects

Some of the most widely used PHP open source projects use different hashing algorithms for passwords because they either support older PHP versions where `password_hash()` wasn't available yet, or they already use the latest security recommendations by PHP security experts:

Project	Password hashing
CMS Airship	Argon2i
Drupal	SHA512Crypt with multiple rounds
Joomla	bcrypt
Laravel	bcrypt with other options
Symfony	bcrypt with other options
Wordpress	salted MD5

See also

- PHP.net passwords FAQ
- csiphp.com - Interesting blog post about encrypting passwords
- password-validator - PHP library that validates passwords against PHP's `password_hash` function using `PASSWORD_DEFAULT`. Will rehash when needed, and will upgrade legacy passwords with the Upgrade decorator.
- securepasswords.info - A polyglot repo of examples for using secure passwords (typically bcrypt).
- How to Safely Store Your Users' Passwords in 2016

Security

How to secure PHP web applications and prevent attacks? How to work with users' passwords and how to securely hash passwords in PHP? What is SQL injection and how to prevent it? How to securely upload files with PHP? Configuration in PHP applications How to protect and hide PHP source code? How to install an SSL certificate and enable HTTPS? Encryption, hashing, encoding and obfuscation

About PHP.earth

Sitemap Team Get Involved Status

PHP.earth documentation

Index PHP installation wizard FAQ <?php tips

Community

Facebook Group GitHub Slack Twitter