Two-way encryption: I need to store passwords that can be retrieved

Asked 13 years, 10 months ago Modified 2 years, 6 months ago

Viewed 74k times Part of PHP Collective

175

I am creating an application that will store passwords, which the user can retrieve and see. The passwords are for a hardware device, so checking against hashes are out of the question.

What I need to know is:

1. How do I encrypt and decrypt a password in PHP?
2. What is the safest algorithm to encrypt the passwords with?
3. Where do I store the private key?
4. Instead of storing the private key, is it a good idea to require users to enter the private key any time they need a password decrypted? (Users of this application can be trusted)
5. In what ways can the password be stolen and decrypted? What do I need to be aware of?

PHP php security encryption passwords

Share

Improve this question

Follow

edited Oct 12, 2014 at 0:16

jww

102k ● 100 ● 437 ● 936

asked Feb 23, 2011 at 10:48

HyderA

21.3k ● 48 ● 116 ● 183

2 Note: Libsodium is now compiled into the PHP core for >= 7.2. This would be the "go to" solution now as it's full of modern methods unlike mcrypt which is considered deprecated and has been removed. – Exhibitioner Mar 4, 2018 at 20:37

8 Answers

Sorted by: Highest score (default)

219

+100

Personally, I would use `mcrypt` like others posted. But there is much more to note...

1. How do I encrypt and decrypt a password in PHP?

   See below for a strong class that takes care of everything for you:

2. What is the safest algorithm to encrypt the passwords with?

   safest? any of them. The safest method if you're going to encrypt is to protect against information disclosure vulnerabilities (XSS, remote inclusion, etc.). If it gets out, the attacker can eventually crack the encryption (no encryption is 100% un-reversible without the key - As @NullUserException points out

this is not entirely true. There are some encryption schemes that are impossible to crack such as one-time pad).

3. Where do I store the private key?

   I would use three keys. One is user supplied, one is application specific and the other is user specific (like a salt). The application specific key can be stored anywhere (in a configuration file outside of the web-root, in an environmental variable, etc.). The user specific one would be stored in a column in the db next to the encrypted password. The user supplied one would not be stored. Then, you'd do something like this:

   ```
   $key = $userKey . $serverKey . $userSuppliedKey;
   ```

   The benefit there, is that any two of the keys can be compromised without the data being compromised. If there's a SQL injection attack, they can get the `$userKey`, but not the other two. If there's a local server exploit, they can get `$userKey` and `$serverKey`, but not the third `$userSuppliedKey`. If they go beat the user with a wrench, they can get the `$userSuppliedKey`, but not the other two (but then again, if the user is beaten with a wrench, you're too late anyway).

4. Instead of storing the private key, is it a good idea to require users to enter the private key any time they need a password decrypted? (Users of this application can be trusted)

Absolutely. In fact, that's the only way I would do it. Otherwise you'd need to store an unencrypted version in a durable storage format (shared memory, such as APC or Memcached, or in a session file). That's exposing yourself to additional compromises. Never store the unencrypted version of the password in anything except a local variable.

5. In what ways can the password be stolen and decrypted? What do I need to be aware of?

   Any form of compromise of your systems will let them view encrypted data. If they can inject code or get to your filesystem, they can view decrypted data (since they can edit the files that decrypt the data). Any form of replay or MITM attack will also give them full access to the keys involved. Sniffing the raw HTTP traffic will also give them the keys.

   Use SSL for all traffic. And make sure nothing on the server has any kind of vulnerabilities (CSRF, XSS, SQL injection, privilege escalation, remote code execution, etc.).

Here's a PHP class implementation of a strong encryption method:

```
/**
 * A class to handle secure encryption and decryption
 *
 * Note that this is not just straight encryption.  It
 *  features in it to make the encrypted data far more
 *  other implementations used to decrypt data will ha
 *  operations.
 *
```

```
 * Security Benefits:
 *
 * - Uses Key stretching
 * - Hides the Initialization Vector
 * - Does HMAC verification of source data
 *
 */
class Encryption {

    /**
     * @var string $cipher The mcrypt cipher to use fo
     */
    protected $cipher = '';

    /**
     * @var int $mode The mcrypt cipher mode to use
     */
    protected $mode = '';

    /**
     * @var int $rounds The number of rounds to feed i
generation
     */
    protected $rounds = 100;

    /**
     * Constructor!
     *
     * @param string $cipher The MCRYPT_* cypher to us
     * @param int    $mode   The MCRYPT_MODE_* mode to
     * @param int    $rounds The number of PBKDF2 roun
     */
    public function __construct($cipher, $mode, $round
        $this->cipher = $cipher;
        $this->mode = $mode;
        $this->rounds = (int) $rounds;
    }

    /**
     * Decrypt the data with the provided key
     *
     * @param string $data The encrypted datat to decr
     * @param string $key  The key to use for decrypti
     *
```

```
     * @returns string|false The returned string if de
     *                       false if it is not
     */
    public function decrypt($data, $key) {
        $salt = substr($data, 0, 128);
        $enc = substr($data, 128, -64);
        $mac = substr($data, -64);

        list ($cipherKey, $macKey, $iv) = $this->getKe

        if (!hash_equals(hash_hmac('sha512', $enc, $ma
            return false;
        }

        $dec = mcrypt_decrypt($this->cipher, $cipherKe
$iv);

        $data = $this->unpad($dec);

        return $data;
    }

    /**
     * Encrypt the supplied data using the supplied ke
     *
     * @param string $data The data to encrypt
     * @param string $key  The key to encrypt with
     *
     * @returns string The encrypted data
     */
    public function encrypt($data, $key) {
        $salt = mcrypt_create_iv(128, MCRYPT_DEV_URAND
        list ($cipherKey, $macKey, $iv) = $this->getKe

        $data = $this->pad($data);

        $enc = mcrypt_encrypt($this->cipher, $cipherKe
$iv);

        $mac = hash_hmac('sha512', $enc, $macKey, true
        return $salt . $enc . $mac;
    }

    /**
```

```
     * Generates a set of keys given a random salt and
     *
     * @param string $salt A random string to change t
     * @param string $key  The supplied key to encrypt
     *
     * @returns array An array of keys (a cipher key,
     */
    protected function getKeys($salt, $key) {
        $ivSize = mcrypt_get_iv_size($this->cipher, $t
        $keySize = mcrypt_get_key_size($this->cipher,
        $length = 2 * $keySize + $ivSize;

        $key = $this->pbkdf2('sha512', $key, $salt, $t

        $cipherKey = substr($key, 0, $keySize);
        $macKey = substr($key, $keySize, $keySize);
        $iv = substr($key, 2 * $keySize);
        return array($cipherKey, $macKey, $iv);
    }

    /**
     * Stretch the key using the PBKDF2 algorithm
     *
     * @see http://en.wikipedia.org/wiki/PBKDF2
     *
     * @param string $algo   The algorithm to use
     * @param string $key    The key to stretch
     * @param string $salt   A random salt
     * @param int    $rounds The number of rounds to d
     * @param int    $length The length of the output
     *
     * @returns string The derived key.
     */
    protected function pbkdf2($algo, $key, $salt, $rou
        $size   = strlen(hash($algo, '', true));
        $len    = ceil($length / $size);
        $result = '';
        for ($i = 1; $i <= $len; $i++) {
            $tmp = hash_hmac($algo, $salt . pack('N',
            $res = $tmp;
            for ($j = 1; $j < $rounds; $j++) {
                 $tmp  = hash_hmac($algo, $tmp, $key,
                 $res ^= $tmp;
            }
```

```
            $result .= $res;
        }
        return substr($result, 0, $length);
    }

    protected function pad($data) {
        $length = mcrypt_get_block_size($this->cipher,
        $padAmount = $length - strlen($data) % $length
        if ($padAmount == 0) {
            $padAmount = $length;
        }
        return $data . str_repeat(chr($padAmount), $pa
    }

    protected function unpad($data) {
        $length = mcrypt_get_block_size($this->cipher,
        $last = ord($data[strlen($data) - 1]);
        if ($last > $length) return false;
        if (substr($data, -1 * $last) !== str_repeat(c
            return false;
        }
        return substr($data, 0, -1 * $last);
    }
}
```

Note that I'm using a function added in PHP 5.6: `hash_equals`. If you're on lower than 5.6, you can use this substitute function which implements a timing-safe comparison function using double HMAC verification:

```
function hash_equals($a, $b) {
    $key = mcrypt_create_iv(128, MCRYPT_DEV_URANDOM);
    return hash_hmac('sha512', $a, $key) === hash_hmac
}
```

Usage:

```
$e = new Encryption(MCRYPT_BLOWFISH, MCRYPT_MODE_CBC);
$encryptedData = $e->encrypt($data, $key);
```

Then, to decrypt:

```
$e2 = new Encryption(MCRYPT_BLOWFISH, MCRYPT_MODE_CBC)
$data = $e2->decrypt($encryptedData, $key);
```

Note that I used `$e2` the second time to show you different instances will still properly decrypt the data.

Now, how does it work/why use it over another solution:

1. Keys

- The keys are not directly used. Instead, the key is stretched by a standard PBKDF2 derivation.
- The key used for encryption is unique for every encrypted block of text. The supplied key therefore becomes a "master key". This class therefore provides key rotation for cipher and auth keys.
- ***Important note***, the `$rounds` parameter is configured for true random keys of sufficient strength (128 bits of cryptographically secure random at a minimum). If you are going to use a password, or non-random key (or less random then 128 bits of CS random), you **must** increase this parameter. I would suggest a minimum of 10000 for passwords (the more you can afford, the better, but it will add to the runtime)...

2. Data Integrity

- The updated version uses ENCRYPT-THEN-MAC, which is a far better method for ensuring the authenticity of the encrypted data.

3. Encryption:

- It uses mcrypt to actually perform the encryption. I would suggest using either `MCRYPT_BLOWFISH` or `MCRYPT_RIJNDAEL_128` cyphers and `MCRYPT_MODE_CBC` for the mode. It's strong enough, and still fairly fast (an encryption and decryption cycle takes about 1/2 second on my machine).

Now, as to point 3 from the first list, what that would give you is a function like this:

```
function makeKey($userKey, $serverKey, $userSuppliedKe
    $key = hash_hmac('sha512', $userKey, $serverKey);
    $key = hash_hmac('sha512', $key, $userSuppliedKey)
    return $key;
}
```

You could stretch it in the `makeKey()` function, but since it's going to be stretched later, there's not really a huge point to doing so.

As far as the storage size, it depends on the plain text. Blowfish uses a 8 byte block size, so you'll have:

- 16 bytes for the salt
- 64 bytes for the hmac

- data length
- Padding so that data length % 8 == 0

So for a 16 character data source, there will be 16 characters of data to be encrypted. So that means the actual encrypted data size is 16 bytes due to padding. Then add the 16 bytes for the salt and 64 bytes for the hmac and the total stored size is 96 bytes. So there's at best a 80 character overhead, and at worst a 87 character overhead...

Share Improve this answer

Follow

edited Jun 2, 2022 at 15:29

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Feb 23, 2011 at 16:04

ircmaxell

165k ● 35 ● 268 ● 315

3 Somebody doesn't understand what it means "break". @IRC nice job on the class, that's pretty damned nice code. – jcolebrand Mar 4, 2011 at 15:43

1 The following returns false. Any idea why? $x = new Encryption(MCRYPT_BlOWFISH, MCRYPT_MODE_CBC); $test = $x->encrypt("test", "a"); echo var_dump($x->decrypt($test, "a")); – The Wavelength Dec 14, 2012 at 21:52

2 Oh and again in the decrypt function changing the two `-64` s to `-128` helped (so you get `$enc = substr($data, 128, -128)` and `$mac = substr($data, -128);` – cosmorogers Dec 17, 2012 at 14:51

4 @ircmaxell It's been quite a while since the code was last revised so I'm wondering if it's up to date. I need to use something similar for a financial application and it would be nice if you gave an okay with this class :-) – nt.bas Mar 22, 2015 at 11:14

2 Warning! The mcrypt extension has been abandonware for nearly a decade now, and was also fairly complex to use. It has therefore been deprecated in favour of OpenSSL, where it will be removed from the core and into PECL in PHP 7.2. th1.php.net/manual/en/migration71.deprecated.php – vee Dec 18, 2016 at 3:57

15

> How do I encrypt and decrypt a password in PHP?

By implementing one of many encryption algorithms (or using one of many libraries)

> What is the safest algorithm to encrypt the passwords with?

There are tons of different algorithms, none of which are 100% secure. But many of them are secure enough for commerce and even military purposes

> Where do I store the private key?

If you have decided to implement a public key - cryptography algorithm (e.g., RSA), you don't store the

private key. The user has a private key. Your system has a public key which could be stored anywhere you wish.

> Instead of storing the private key, is it a good idea to require users to enter the private key any time they need a password decrypted? (Users of this application can be trusted)

Well, if your user can remember ridiculously long prime numbers then - yes, why not. But generally you would need to come up with the system which will allow user to store their key somewhere.

> In what ways can the password be stolen and decrypted? What do I need to be aware of?

This depends on the algorithm used. However, always make sure that you don't send password unencrypted to or from the user. Either encrypt/decrypt it on the client side, or use HTTPS (or user other cryptographic means to secure connection between server and client).

However if all you need is to store passwords in encrypted way, I would suggest you to use a simple XOR cipher. The main problem with this algorithm is that it could be easily broken by frequency analysis. However, as generally passwords are not made from long paragraphs of English text I don't think you should worry about it. The second problem with an XOR cipher is that if you have a message in both encrypted and decrypted

form you could easily find out password with which it was encrypted. Again, not a big problem in your case as it only affects the user who already was compromised by other means.

Share Improve this answer

Follow

edited Jun 2, 2022 at 15:37

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Feb 23, 2011 at 11:37

Ivan

3,645 ● 19 ● 25

On answer 3, when you say users have private key, I don't understand what that means. You don't recommend passing private keys into the application manually by the user, so how else are private keys passed to the application? – HyderA Feb 23, 2011 at 11:52

Well that's a bit of a problem. Private key could be stored in the text file and then copy pasted to the app. Key could also be stored on the server but in this case it still should be encrypted with some other encryption algorithm like XOR. Using XOR here in this case is secure enough as there is only one password-message pair and message is quite random so frequency analysis cold not be used. – Ivan Feb 23, 2011 at 12:00

4 I certainly wouldn't recommend implementing an encryption algorithm yourself, there's too many potential pitfalls and the existing libraries have been tested and analysed by many people. – Long Ears Feb 23, 2011 at 12:29

The main problem with XOR is that if someone steals your application data and knows just one of a user's passwords, they can decrypt all the other passwords for that user.
– Long Ears Feb 23, 2011 at 12:31

1 @Ivan: yes, but this is one of the cases when I think DIY is really **really** bad unless you REALLY understand cryptography. There are strong ciphers that exist, why not use them? – ircmaxell Mar 3, 2011 at 21:41

13

1. The PHP module you are after is Mcrypt.

   The example from the manual is slightly edited for this example):

```
<?php
    $iv_size = mcrypt_get_iv_size(MCRYPT_BLOWFISH,
    $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
    $key = "This is a very secret key";
    $pass = "PasswordHere";
    echo strlen($pass) . "\n";

    $crypttext = mcrypt_encrypt(MCRYPT_BLOWFISH, $
$iv);
    echo strlen($crypttext) . "\n";
?>
```

   You would use **mcrypt_decrypt** to decrypt your password.

2. The best algorithm is rather subjective - ask five people, and get five answers. Personally, if the default (Blowfish) isn't good enough for you, you probably have bigger problems!

3. Given that it is needed by PHP to encrypt, I am not sure you can hide it anywhere. Standard PHP best

coding practices apply of course!

4. Given that the encryption key will be in your code anyway, I am not sure what you will gain, providing the rest of your application is secure.
5. Obviously, if the encrypted password and the encryption key are stolen, then game over.

I'd put a rider on my answer. I'm not a PHP cryptography expert, but, I think what I have answered is standard practice.

edited Jun 2, 2022 at 16:01

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Feb 23, 2011 at 11:48

Jon Rhoades

496 ● 4 ● 12

`$pass = $text`. I think he changed that to cater to the question, and didn't notice the second occurrence.
– HyderA Feb 23, 2011 at 11:56

3 Two things to note. First, `MCRYPT_MODE_ECB` doesn't use an IV. Second, if it did, you'd need to store the IV as you can't decrypt the data without it... – ircmaxell Mar 3, 2011 at 21:09

"The best algorithm is rather subjective - ask 5 people, get 5 answers. Personally if the the default (Blowfish) isn't good enough for you, you probably have bigger problems!" This is totally wrong. Any crypto expert will more-or-less agree with gist.github.com/tqbf/be58d2d39690c3b366ad which specifically excludes blowfish – Scott Arciszewski Feb 15, 2016 at 8:26

6

A lot of users have suggested using mcrypt... which is correct, but I like to go a step further to make it easily stored and transferred (as sometimes encrypted values can make them hard to send using other technologies like cURL, or JSON).

After you have successfully encrypted using mcrypt, run it through base64_encode and then convert it to hexadecimal code. Once in hexadecimal code it's easy to transfer in a variety of ways.

```
$td = mcrypt_module_open('tripledes', '', 'ecb', '');
$iv = mcrypt_create_iv (mcrypt_enc_get_iv_size($td), M
$key = substr("SUPERSECRETKEY", 0, mcrypt_enc_get_key_
mcrypt_generic_init($td, $key, $iv);
$encrypted = mcrypt_generic($td, $unencrypted);
$encrypted = $ua . "||||" . $iv;
mcrypt_generic_deinit($td);
```

```
mcrypt_module_close($td);
$encrypted = base64_encode($encrypted);
$encrypted = array_shift(unpack('H*', $encrypted));
```

And on the other side:

```
$encrypted = pack('H*', $encrypted);
$encrypted = base64_decode($encrypted);
list($encrypted, $iv) = explode("||||", $encrypted, 2)
$td = mcrypt_module_open('tripledes', '', 'ecb', '');
$key = substr("SUPERSECRETKEY", 0, mcrypt_enc_get_key_
mcrypt_generic_init($td, $key, $iv);
$unencrypted = mdecrypt_generic($td, $encrypted);
mcrypt_generic_deinit($td);
mcrypt_module_close($td);
```

Share Improve this answer

Follow

edited Jun 2, 2022 at 15:42

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Feb 25, 2011 at 23:20

Bradley

2,141 ● 2 ● 21 ● 32

Mcrypt isn't a great option – Scott Arciszewski Feb 15, 2016 at 8:27

2 Well - it was in 2011 :P – Bradley Mar 18, 2017 at 7:15

5

I'd only suggest public key encryption if you want the ability to set a user's password without their interaction (this can be handy for resets and shared passwords).

Public key

1. The OpenSSL extension, specifically `openssl_public_encrypt` and `openssl_private_decrypt`
2. This would be straight RSA assuming your passwords will fit in key size - padding, otherwise you need a symmetric layer
3. Store both keys for each user, the private key's passphrase is their application password

Symmetric

1. The Mcrypt extension
2. AES-256 is probably a safe bet, but this could be a SO question in itself
3. You don't - this would be their application password

Both

`4`. Yes - users would have to enter their application password every time, but storing it in the session would raise other issues

`5`.

- If someone steals the application data, it's as secure as the symmetric cipher (for the public key scheme, it's used to protect the private key with the passphrase.)
- Your application should definitely be only accessible over SSL, preferably using client certificates.
- Consider adding a second factor for authentication which would only be used once per session, like a token sent via SMS.

Share Improve this answer

Follow

answered Feb 23, 2011 at 11:57

Long Ears

4,896 ● 1 ● 22 ● 16

Avoid mcrypt, be careful with `openssl_private_decrypt()` . – Scott Arciszewski Feb 15, 2016 at 8:25

3

> The passwords are for a hardware device, so checking against hashes are out of the question

Eh? I don't understand. Do you just mean that passwords must be recoverable?

As others have said, the mcrypt extension provides access to lots of cryptographic functions - however you are inviting your users to put all their eggs in one basket - one which will be potentially be a target for attackers - and if you don't even know how to start solving the

problem then you are doing your users a disservice. You are not in a position to understand how to protect the data.

Most security vulnerabilities come about not because the underlying algorithm is flawed or insecure - but because of problems with the way the algorithm is used within the application code.

Having said that, it is **possible** to build a reasonably secure system.

You should only consider asymmetric encryption if you have a requirement for a user to create a secure message which is readable by another (specific) user. The reason being that it’s computationally expensive. If you just want to provide a repository for users to enter and retrieve their own data, symmetric encryption is adequate.

If, however, you store the key for decrypting the message in the same place as the encrypted message (or where the encrypted message is stored) then the system is not secure. Use the same token for authenticating the user as for the decryption key (or in the case of asymmetric encryption, use the token as the private key pass phrase). Since you will need to store the token on the server where the decryption takes place at least temporarily, you might want to consider using a non-searchable session storage substrate, or passing the token directly to a daemon associated with the session

which would store the token in memory and perform the decryption of messages on demand.

Share Improve this answer

Follow

edited Jun 2, 2022 at 15:49

Peter Mortensen

31.6k • 22 • 109 • 133

answered Mar 1, 2011 at 10:33

symcbean

48.3k • 6 • 62 • 97

2

I tried something like this, but please note that I am not cryptographer nor do I hold in-depth knowledge about PHP or any programming language. It's just an idea.

My idea is to store a *key* in some file or *database* (or enter manually) whose (location) cannot be easily predicted (and of course anything will be decrypted some day. The concept is to lengthen the decryption time) and encrypt sensitive information.

Code

```
$iv_size = mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_
$iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
$key = "evenifyouaccessmydatabaseyouwillneverfindmyema
$text = "myemail@domain.com";
echo "Key: " . $key . "<br/>";
echo "Text: " . $text . "<br/>";
echo "MD5: " . md5($text) . "<br/>";
echo "SHA-1: " . sha1($text) . "<br/>";

$crypttext = mcrypt_encrypt(MCRYPT_BLOWFISH, $key, $te
$iv);
```

```
echo "Encrypted Data: " . $crypttext . "<br>";

$base64 = base64_encode($crypttext);
echo "Encoded Data: " . $base64 . "<br/>";
$decode =  base64_decode($base64);

$decryptdata = mcrypt_decrypt(MCRYPT_BLOWFISH, $key, $
MCRYPT_MODE_ECB, $iv);

echo "Decoded Data: " . ereg_replace("?", null,  $decr
// Even if I add '?' to the sting to the text it works
```

Please note that it is just a concept. Any improvement to this code would be highly appreciated.

Share Improve this answer

Follow

edited Jun 2, 2022 at 16:00

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Mar 1, 2011 at 4:29

S L

14.3k ● 18 ● 79 ● 119

1

Use password_hash and password_verify

```
<?php
/**
 * In this case, we want to increase the default cost
 * Note that we also switched to BCRYPT, which will al
 */
$options = [
    'cost' => 12,
];
echo password_hash("rasmuslerdorf", PASSWORD_BCRYPT, $
?>
```

And to decrypt:

```php
<?php
// See the password_hash() example to see where this c
$hash = '$2y$07$BCryptRequires22Chrcte/VlQH0piJtjXl.0t

if (password_verify('rasmuslerdorf', $hash)) {
    echo 'Password is valid!';
} else {
    echo 'Invalid password.';
}
?>
```

Share Improve this answer

Follow

answered Sep 24, 2017 at 22:51

jvitoroc

390 ● 2 ● 4 ● 16

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.