PHP AES encrypt / decrypt

Asked 13 years, 9 months ago Modified 2 years, 3 months ago Viewed 321k times Part of PHP Collective

69

I found an example for en/decoding strings in PHP. At first it looks very good but it wont work :-(

Does anyone know what the problem is?

```
$Pass = "Passwort";
$Clear = "Klartext";

$crypted = fnEncrypt($Clear, $Pass);
echo "Encrypted: ".$crypted."</br>";

$newClear = fnDecrypt($crypted, $Pass);
echo "Decrypted: ".$newClear."</br>";

function fnEncrypt($sValue, $sSecretKey) {
    return trim(base64_encode(mcrypt_encrypt(MCRYPT_RIJNDAEL_256, $sSecretKey,
$sDecrypted, MCRYPT_MODE_ECB,
mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_RIJNDAEL_256, MCRYPT_MODE_ECB),
MCRYPT_RAND))));
}

function fnDecrypt($sValue, $sSecretKey) {
    return trim(mcrypt_decrypt(MCRYPT_RIJNDAEL_256, $sSecretKey,
base64_decode($sEncrypted), MCRYPT_MODE_ECB,
mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_RIJNDAEL_256, MCRYPT_MODE_ECB),
MCRYPT_RAND)));
}
```

The result is:

Encrypted: `boKRNTYYNp7AiOvY1CidqsAn9wX4ufz/D9XrpjAOPk8=`

Decrypted: `–,(ÑÁ    ^ yË~F'¸®Ó–í    œð2Á_B‰Â–`

php encryption cryptography aes encryption-symmetric

Share Edit Follow

edited Dec 20, 2015 at 6:44
Scott Arciszewski 34k 16 91 212

asked Aug 6, 2010 at 10:08
Andreas Prang 2,207 4 23 33

6 ECB is insecure (so is CBC for communication protocols). `MCRYPT_RIJNDAEL_256` is not AES. – Maarten Bodewes Dec 17, 2014 at 15:27

Here is a good example explaining how to encrypt/decrypt data in PHP using MCrypt Library code-epicenter.com/how-to-use-mcrypt-library-in-php – MrD Apr 9, 2016 at 22:05

2 Also see Upgrading my encryption library from Mcrypt to OpenSSL, Replace Mcrypt with OpenSSL and Preparing for removal of Mcrypt in PHP 7.2 – jww Jul 10, 2017 at 6:15

As the checked answer is considered badly broken and insecure, please move the accepted answer on this question. – Yakk - Adam Nevraumont Jul 10, 2017 at 23:32

10 Answers

Sorted by: Highest score (default)

97

+300

Please use an existing secure PHP encryption library

It's generally a bad idea to write your own cryptography unless you have experience breaking other peoples' cryptography implementations.

None of the examples here authenticate the ciphertext, which leaves them vulnerable to bit-rewriting attacks.

If you can install PECL extensions, libsodium is even better

```
<?php
// PECL libsodium 0.2.1 and newer

/**
 * Encrypt a message
 *
 * @param string $message - message to encrypt
 * @param string $key - encryption key
 * @return string
 */
function safeEncrypt($message, $key)
{
    $nonce = \Sodium\randombytes_buf(
        \Sodium\CRYPTO_SECRETBOX_NONCEBYTES
    );

    return base64_encode(
        $nonce.
        \Sodium\crypto_secretbox(
            $message,
            $nonce,
            $key
        )
    );
}

/**
 * Decrypt a message
 *
 * @param string $encrypted - message encrypted with safeEncrypt()
 * @param string $key - encryption key
 * @return string
 */
function safeDecrypt($encrypted, $key)
{
    $decoded = base64_decode($encrypted);
    $nonce = mb_substr($decoded, 0, \Sodium\CRYPTO_SECRETBOX_NONCEBYTES, '8bit');
    $ciphertext = mb_substr($decoded, \Sodium\CRYPTO_SECRETBOX_NONCEBYTES, null,
'8bit');

    return \Sodium\crypto_secretbox_open(
        $ciphertext,
        $nonce,
        $key
    );
}
```

Then to test it out:

```
<?php
// This refers to the previous code block.
require "safeCrypto.php";

// Do this once then store it somehow:
$key = \Sodium\randombytes_buf(\Sodium\CRYPTO_SECRETBOX_KEYBYTES);
$message = 'We are all living in a yellow submarine';

$ciphertext = safeEncrypt($message, $key);
$plaintext = safeDecrypt($ciphertext, $key);

var_dump($ciphertext);
var_dump($plaintext);
```

This can be used in any situation where you are passing data to the client (e.g. encrypted cookies for sessions without server-side storage, encrypted URL parameters, etc.) with a reasonably high degree of certainty that the end user cannot decipher or reliably tamper with it.

Since libsodium is cross-platform, this also makes it easier to communicate with PHP from, e.g. Java applets or native mobile apps.

Note: If you specifically need to add encrypted cookies powered by libsodium to your app, my employer Paragon Initiative Enterprises is developing a library called Halite that does all of this for you.

Share Edit Follow edited Feb 15, 2016 at 8:55 answered May 11, 2015 at 11:06
Scott Arciszewski
34k 16 91 212

the library you provided, returns encrypted message in binary format - correct? Is it possible to return in a simple string format? Thanks – Andrew Dec 28, 2015 at 15:32

A C# .NET port is available in this GitHub repo, in case anyone needs it : github.com/mayerwin/SaferCrypto. Thank you @ScottArciszewski. – Erwin Mayer Jan 3, 2017 at 4:23

No @Andrew, it returns the message with base64 encoding, which is a simple string – Riking Jul 12, 2017 at 22:13

80

If you **don't want to use a heavy dependency** for something solvable in 15 lines of code, use the built in **OpenSSL** functions. Most PHP installations come with OpenSSL, which provides fast, compatible and secure AES encryption in PHP. Well, it's secure as long as you're following the best practices.

The following code:

- uses AES256 in CBC mode
- is compatible with other AES implementations, but **not mcrypt**, since mcrypt uses PKCS#5 instead of PKCS#7.
- generates a key from the provided password using SHA256
- generates a hmac hash of the encrypted data for integrity check
- generates a random IV for each message
- prepends the IV (16 bytes) and the hash (32 bytes) to the ciphertext
- should be pretty secure

IV is a public information and needs to be random for each message. The hash ensures that the data hasn't been tampered with.

```
function encrypt($plaintext, $password) {
    $method = "AES-256-CBC";
    $key = hash('sha256', $password, true);
    $iv = openssl_random_pseudo_bytes(16);

    $ciphertext = openssl_encrypt($plaintext, $method, $key, OPENSSL_RAW_DATA,
$iv);
    $hash = hash_hmac('sha256', $ciphertext . $iv, $key, true);

    return $iv . $hash . $ciphertext;
}

function decrypt($ivHashCiphertext, $password) {
    $method = "AES-256-CBC";
    $iv = substr($ivHashCiphertext, 0, 16);
    $hash = substr($ivHashCiphertext, 16, 32);
    $ciphertext = substr($ivHashCiphertext, 48);
    $key = hash('sha256', $password, true);

    if (!hash_equals(hash_hmac('sha256', $ciphertext . $iv, $key, true), $hash))
return null;

    return openssl_decrypt($ciphertext, $method, $key, OPENSSL_RAW_DATA, $iv);
}
```

Usage:

```
$encrypted = encrypt('Plaintext string.', 'password'); // this yields a binary
string

echo decrypt($encrypted, 'password');
// decrypt($encrypted, 'wrong password') === null
```

edit: Updated to use `hash_equals` and added IV to the hash.

Share Edit Follow edited Oct 5, 2019 at 9:43 answered Oct 22, 2017 at 9:31
blade
12.6k 7 39 40

4 It's not a good practice to compare hashes using the equality operator, You should use `hash_equals()` instead, as it's vulnerable to the timing attack, more information here – Shahin Mar 25, 2019 at 22:13

2 This answer is almost there, but needs... 1) A better KDF, SHA-256 is a very poor KDF. Use PBKDF2 at the very least, but Argon2/bcrypt would be better. 2) The IV needs to be included in the HMAC - the point of an HMAC is to ensure decryption will either result in the plaintext or fail - not including the IV gives rise to a situation where the user *thinks* they're getting the original plaintext, but aren't. 3) Use a time safe comparison when comparing the hashes, otherwise this code *could* be vulnerable to timing attacks. 4) Don't use the same key for HMAC as you do for AES. – Luke Joshua Park Jul 4, 2019 at 20:59

2 @MikkoRantalainen We can't assume the key is secure, specifically because it isn't a key, it's a human-selected, low-entropy password. If we use SHA-256 to derive our encryption key, which takes a very negligible amount of time, brute force attacks on passwords are quite easy. However, if we use PBKDF2 or Argon2, where we can fine-tune the time it takes to derive a password (think a few hundred milliseconds), brute forcing becomes far less feasible. Pretty much the exact same reason we wouldn't use SHA-256 as a password hash. – Luke Joshua Park Apr 2, 2020 at 10:13

2 If you want to store this in a database you should do base64_encode on the return value of the encrypt function and base64_decode on the $ivHashCiphertext value of the decrypt function. If not you might run into encoding problems with the database. – Chris May 26, 2021 at 22:39

2 @naman1994 The output a binary blob, so you need to encode it into plain text - ie. using base64_encode, bin2hex, etc. – blade Jun 15, 2021 at 15:45

59

`$sDecrypted` and `$sEncrypted` were undefined in your code. See a solution that works (**but is not secure!**):

STOP!

> This example is **insecure!** Do not use it!

```
$Pass = "Passwort";
$Clear = "Klartext";

$crypted = fnEncrypt($Clear, $Pass);
echo "Encryped: ".$crypted."</br>";

$newClear = fnDecrypt($crypted, $Pass);
echo "Decryped: ".$newClear."</br>";

function fnEncrypt($sValue, $sSecretKey)
{
    return rtrim(
        base64_encode(
            mcrypt_encrypt(
                MCRYPT_RIJNDAEL_256,
                $sSecretKey, $sValue,
                MCRYPT_MODE_ECB,
                mcrypt_create_iv(
                    mcrypt_get_iv_size(
                        MCRYPT_RIJNDAEL_256,
                        MCRYPT_MODE_ECB
                    ),
                    MCRYPT_RAND)
                )
            ), "\0"
        );
}

function fnDecrypt($sValue, $sSecretKey)
{
    return rtrim(
        mcrypt_decrypt(
        MCRYPT_RIJNDAEL_256,
        $sSecretKey,
        base64_decode($sValue),
        MCRYPT_MODE_ECB,
        mcrypt_create_iv(
            mcrypt_get_iv_size(
                    MCRYPT_RIJNDAEL_256,
                    MCRYPT_MODE_ECB
                ),
                MCRYPT_RAND
            )
        ), "\0"
    );
}
```

But there are other problems in this code which make it insecure, in particular the use of ECB (which is not an *encryption* mode, only a building block on top of which encryption modes can be defined). See Fab Sa's answer for a quick fix of the worst problems and Scott's answer for how to do this right.

Share Edit Follow

edited Jul 10, 2017 at 5:49
Clonkex
3,491 8 41 58

answered Aug 6, 2010 at 10:12
zz1433
3,558 2 30 37

17 I was using this code, and I found a bug. The trim should NOT be used!!! it should be a rtrim() with a second parameter "\0". In rare cases the first or last character of the encrypted value could be a space or return, the decryption goes wrong... – Paul Jacobse May 23, 2012 at 11:07

3 Could you obfuscate that anymore please? I'm afraid I can, with some effort, still read what your code does. </sarcasm mode> Code should be self-descriptive. Give things like the output of mcrypt_get_iv_size an appropriate variable name and then use it. This kind of indentation is hard to read. Unless you're used to Lisp, maybe, but most PHP programmers aren't I'd say. I'm not saying the OP's (question asker's) code was any better, but as a good answer you might improve the code too. – Luc Sep 16, 2013 at 19:36

6 -1 for ECB. See the Wikipedia entry for details on why it's a poor choice of block cipher mode. – Polynomial Mar 16, 2014 at 1:26

9 -1: Passing different IVs in the `encrypt` and `decrypt` functions makes no sense at all. The only reason why this works is because the ECB mode does not use an initialization vector at all, so any value would do and produce the same output. – Clément May 3, 2014 at 14:44

4 I don't know why this answer is so up voted. Just because 'it works' it doesn't mean it secured and should be used on production environments. MCRYPT_MODE_ECB use is highly descurged and even PHP mcrypt_ecb function has been DEPRECATED as of PHP 5.5.0. Relying on this function is highly discouraged. Instead you should use MCRYPT_MODE_CBC mode: wpy.me/blog/15-encrypt-and-decrypt-data-in-php-using-aes-256 – wappy Feb 27, 2015 at 15:04

27

For information `MCRYPT_MODE_ECB` doesn't use the IV (initialization vector). ECB mode divide your message into blocks and each block is encrypted separately. I really *don't recommended it*.

CBC mode use the IV to make each message unique. CBC is recommended and should be used instead of ECB.

Example :

```php
<?php
$password = "myPassword_!";
$messageClear = "Secret message";

// 32 byte binary blob
$aes256Key = hash("SHA256", $password, true);

// for good entropy (for MCRYPT_RAND)
srand((double) microtime() * 1000000);
// generate random iv
$iv = mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_RIJNDAEL_256, MCRYPT_MODE_CBC),
MCRYPT_RAND);


$crypted = fnEncrypt($messageClear, $aes256Key);

$newClear = fnDecrypt($crypted, $aes256Key);

echo
"IV:        <code>".$iv."</code><br/>".
"Encrypred: <code>".$crypted."</code><br/>".
"Decrypred: <code>".$newClear."</code><br/>";

function fnEncrypt($sValue, $sSecretKey) {
    global $iv;
    return rtrim(base64_encode(mcrypt_encrypt(MCRYPT_RIJNDAEL_256, $sSecretKey,
$sValue, MCRYPT_MODE_CBC, $iv)), "\0\3");
}

function fnDecrypt($sValue, $sSecretKey) {
    global $iv;
    return rtrim(mcrypt_decrypt(MCRYPT_RIJNDAEL_256, $sSecretKey,
base64_decode($sValue), MCRYPT_MODE_CBC, $iv), "\0\3");
}
```

You have to stock the IV to decode each message (IV are *not* secret). Each message is unique because each message has an unique IV.

- More informations about mode of operation (wikipedia).

Share Edit Follow edited Nov 18, 2014 at 12:55 answered Sep 9, 2012 at 13:21
Fabien Sa
9,272 4 38 45

1 You should add an example to clearify the usage of `aes256Key`. This example shows how to use it: php.net/manual/fr/book.mcrypt.php#107483 – mgutt Mar 17, 2013 at 16:15

1 It's pretty the same but my example doesn't use salt for clarity. – Fabien Sa Mar 20, 2013 at 9:10

2 +1. The top answer generates random IVs to feed a system (ECB) that doesn't need any. – Clément May 3, 2014 at 14:47

If I have understood it properly, the $password = "myPassword_!" becomes part of the encryption algorithm, right? – user3746998 Jul 2, 2014 at 15:34

4 Be warned that the code above does not use AES nor does it use PKCS#7 padding, which means it will be incompatible with any other system out there. I'm the guy that fixed that example code for `mcrypt_encrypt`. – Maarten Bodewes Dec 17, 2014 at 15:25

6

These are compact methods to encrypt / decrypt strings with PHP using **AES256 CBC**:

```
function encryptString($plaintext, $password, $encoding = null) {
    $iv = openssl_random_pseudo_bytes(16);
    $ciphertext = openssl_encrypt($plaintext, "AES-256-CBC", hash('sha256', 
$password, true), OPENSSL_RAW_DATA, $iv);
    $hmac = hash_hmac('sha256', $ciphertext.$iv, hash('sha256', $password, true), 
true);
    return $encoding == "hex" ? bin2hex($iv.$hmac.$ciphertext) : ($encoding == 
"base64" ? base64_encode($iv.$hmac.$ciphertext) : $iv.$hmac.$ciphertext);
}

function decryptString($ciphertext, $password, $encoding = null) {
    $ciphertext = $encoding == "hex" ? hex2bin($ciphertext) : ($encoding == 
"base64" ? base64_decode($ciphertext) : $ciphertext);
    if (!hash_equals(hash_hmac('sha256', substr($ciphertext, 
48).substr($ciphertext, 0, 16), hash('sha256', $password, true), true), 
substr($ciphertext, 16, 32))) return null;
    return openssl_decrypt(substr($ciphertext, 48), "AES-256-CBC", hash('sha256', 
$password, true), OPENSSL_RAW_DATA, substr($ciphertext, 0, 16));
}
```

Usage:

```
$enc = encryptString("mysecretText", "myPassword");
$dec = decryptString($enc, "myPassword");
```

EDIT: This is a new version of functions that use **AES256 GCM** and **PBKDF2** as key derivation, more secure.

```
function str_encryptaesgcm($plaintext, $password, $encoding = null) {
    if ($plaintext != null && $password != null) {
        $keysalt = openssl_random_pseudo_bytes(16);
        $key = hash_pbkdf2("sha512", $password, $keysalt, 20000, 32, true);
        $iv = openssl_random_pseudo_bytes(openssl_cipher_iv_length("aes-256-
gcm"));
        $tag = "";
        $encryptedstring = openssl_encrypt($plaintext, "aes-256-gcm", $key,
OPENSSL_RAW_DATA, $iv, $tag, "", 16);
        return $encoding == "hex" ? bin2hex($keysalt.$iv.$encryptedstring.$tag) :
($encoding == "base64" ? base64_encode($keysalt.$iv.$encryptedstring.$tag) :
$keysalt.$iv.$encryptedstring.$tag);
    }
}

function str_decryptaesgcm($encryptedstring, $password, $encoding = null) {
    if ($encryptedstring != null && $password != null) {
        $encryptedstring = $encoding == "hex" ? hex2bin($encryptedstring) :
($encoding == "base64" ? base64_decode($encryptedstring) : $encryptedstring);
        $keysalt = substr($encryptedstring, 0, 16);
        $key = hash_pbkdf2("sha512", $password, $keysalt, 20000, 32, true);
        $ivlength = openssl_cipher_iv_length("aes-256-gcm");
        $iv = substr($encryptedstring, 16, $ivlength);
        $tag = substr($encryptedstring, -16);
        return openssl_decrypt(substr($encryptedstring, 16 + $ivlength, -16),
"aes-256-gcm", $key, OPENSSL_RAW_DATA, $iv, $tag);
    }
}
```

Usage:

```
$enc = str_encryptaesgcm("mysecretText", "myPassword", "base64"); // return a
base64 encrypted string, you can also choose hex or null as encoding.
$dec = str_decryptaesgcm($enc, "myPassword", "base64");
```

Share Edit Follow edited Sep 1, 2021 at 15:23 answered Jun 3, 2020 at 14:21

Marco Concas
1,778 22 27

I found that this worked, while the others above produced strange characters and didn't decrypt to anything legible. – WilliamK Jul 22, 2020 at 7:54

I am finding that this works intermittently. Each time it encrypts it produces a different answer. Sometimes it does not decrypt. – WilliamK Jul 22, 2020 at 8:37

Try this: encryptString("mysecretText", "myPassword", "hex") | decryptString($enc, "myPassword", "hex") @WilliamK – Marco Concas Aug 26, 2020 at 22:31

I encrypt the string in Javascript using crypto-es and want to decrypt it in PHP using your function, but it returns null. The passphrase are the same in JS and PHP. I already set encoding param using 'base64', but no luck. What am I possibly missing here? – Jeaf Gilbert Jul 25, 2021 at 6:03

I should check the function you use on JS to understand the problem, anyway I have tested this with C# and everything works perfectly. – Marco Concas Jul 25, 2021 at 17:22

6

This is a working solution of `AES encryption` - implemented using `openssl`. It uses the Cipher Block Chaining Mode (CBC-Mode). Thus, alongside `data` and `key`, you can specify `iv` and `block size`

```
<?php
     class AESEncryption {

           protected $key;
           protected $data;
           protected $method;
           protected $iv;

           /**
            * Available OPENSSL_RAW_DATA | OPENSSL_ZERO_PADDING
            *
            * @var type $options
            */
           protected $options = 0;

           /**
            *
            * @param type $data
            * @param type $key
            * @param type $iv
            * @param type $blockSize
            * @param type $mode
            */
           public function __construct($data = null, $key = null, $iv = null,
$blockSize = null, $mode = 'CBC') {
               $this->setData($data);
               $this->setKey($key);
               $this->setInitializationVector($iv);
               $this->setMethod($blockSize, $mode);
           }

           /**
            *
            * @param type $data
            */
           public function setData($data) {
               $this->data = $data;
           }

           /**
            *
            * @param type $key
            */
           public function setKey($key) {
               $this->key = $key;
           }

           /**
            * CBC 128 192 256
             CBC-HMAC-SHA1 128 256
             CBC-HMAC-SHA256 128 256
             CFB 128 192 256
             CFB1 128 192 256
             CFB8 128 192 256
             CTR 128 192 256
             ECB 128 192 256
             OFB 128 192 256
             XTS 128 256
            * @param type $blockSize
            * @param type $mode
            */
           public function setMethod($blockSize, $mode = 'CBC') {
               if($blockSize==192 && in_array('', array('CBC-HMAC-SHA1','CBC-
HMAC-SHA256','XTS'))){
                   $this->method=null;
                   throw new Exception('Invalid block size and mode
combination!');
                }
                $this->method = 'AES-' . $blockSize . '-' . $mode;
            }

            /**
             *
             * @param type $data
             */
            public function setInitializationVector($iv) {
                $this->iv = $iv;
            }

            /**
             *
             * @return boolean
             */
            public function validateParams() {
                if ($this->data != null &&
                        $this->method != null ) {
                    return true;
                } else {
                    return FALSE;
                }
            }

            //it must be the same when you encrypt and decrypt
            protected function getIV() {
                return $this->iv;
            }

             /**
             * @return type
             * @throws Exception
             */
            public function encrypt() {
                if ($this->validateParams()) {
                    return trim(openssl_encrypt($this->data, $this->method,
$this->key, $this->options,$this->getIV()));
                } else {
                    throw new Exception('Invalid params!');
                }
            }

            /**
             *
             * @return type
             * @throws Exception
             */
            public function decrypt() {
                if ($this->validateParams()) {
                   $ret=openssl_decrypt($this->data, $this->method, $this->key,
$this->options,$this->getIV());

                   return   trim($ret);
                } else {
                    throw new Exception('Invalid params!');
                }
            }

        }
```

Sample usage:

```
<?php
        $data =
json_encode(['first_name'=>'Dunsin','last_name'=>'Olubobokun','country'=>'Nigeria'])
        $inputKey = "W92ZB837943A711B98D35E799DFE3Z18";
        $iv = "tuqZQhKP48e8Piuc";
        $blockSize = 256;
        $aes = new AESEncryption($data, $inputKey, $iv, $blockSize);
        $enc = $aes->encrypt();
        $aes->setData($enc);
        $dec=$aes->decrypt();
        echo "After encryption: ".$enc."<br/>";
        echo "After decryption: ".$dec."<br/>";
```

Share Edit Follow edited Jul 3, 2019 at 7:28 answered Jul 2, 2019 at 16:44

Dunsin Olubobokun
882 10 18

1 This code leaves IV handling to the user (who *will* do it poorly) and also doesn't include any integrity checks. Not good crypto code. – Luke Joshua Park Jul 4, 2019 at 21:01

3

Few important things to note with AES encryption:

1. Never use plain text as encryption key. Always hash the plain text key and then use for encryption.
2. Always use Random IV (initialization vector) for encryption and decryption. **True randomization** is important.
3. As mentioned above, don't use ecb mode, use `CBC` instead.

Share Edit Follow edited Jul 28, 2017 at 11:45 answered Jun 12, 2015 at 22:58

Naveen DA
4,246 6 39 60

Navneet Kumar
919 7 6

1 It isn't enough just to hash a password to use as an encryption key, see comments on blade's answer. – Luke Joshua Park Apr 5, 2020 at 3:05

1

If you are using MCRYPT_RIJNDAEL_128, try `rtrim($output, "\0\3")`. If the length of the string is less than 16, the decrypt function will return a string with length of 16 characters, adding 03 at the end.

You can easily check this, e.g. by trying:

```
$string = "TheString";
$decrypted_string = decrypt_function($stirng, $key);

echo bin2hex($decrypted_string)."=".bin2hex("TheString");
```

Share Edit Follow

edited Aug 28, 2012 at 17:03
ЯegDwight
25k 10 46 52

answered Aug 28, 2012 at 16:56
Kamen
11 1

Using MCRYPT_RIJNDAEL_128 this worked for me: rtrim($output, "\x00..\x1F") – Sergio Viudes Jul 16, 2014 at 15:30

1

Here's an improved version based on code written by blade

- add comments
- overwrite arguments before throwing to avoid leaking secrets with the exception
- check return values from openssl and hmac functions

The code:

```
class Crypto
{
    /**
     * Encrypt data using OpenSSL (AES-256-CBC)
     * @param string $plaindata Data to be encrypted
     * @param string $cryptokey key for encryption (with 256 bit of entropy)
     * @param string $hashkey key for hashing (with 256 bit of entropy)
     * @return string IV+Hash+Encrypted as raw binary string. The first 16
     *     bytes is IV, next 32 bytes is HMAC-SHA256 and the rest is
     *     $plaindata as encrypted.
     * @throws Exception on internal error
     *
     * Based on code from: https://stackoverflow.com/a/46872528
     */
    public static function encrypt($plaindata, $cryptokey, $hashkey)
    {
        $method = "AES-256-CBC";
        $key = hash('sha256', $cryptokey, true);
        $iv = openssl_random_pseudo_bytes(16);

        $cipherdata = openssl_encrypt($plaindata, $method, $key,
OPENSSL_RAW_DATA, $iv);

        if ($cipherdata === false)
        {
            $cryptokey = "**REMOVED**";
            $hashkey = "**REMOVED**";
            throw new \Exception("Internal error: openssl_encrypt()
failed:".openssl_error_string());
        }

        $hash = hash_hmac('sha256', $cipherdata.$iv, $hashkey, true);

        if ($hash === false)
        {
            $cryptokey = "**REMOVED**";
            $hashkey = "**REMOVED**";
            throw new \Exception("Internal error: hash_hmac() failed");
        }

        return $iv.$hash.$cipherdata;
    }
    /**
    * Decrypt data using OpenSSL (AES-256-CBC)
     * @param string $encrypteddata IV+Hash+Encrypted as raw binary string
     *     where the first 16 bytes is IV, next 32 bytes is HMAC-SHA256 and
     *     the rest is encrypted payload.
     * @param string $cryptokey key for decryption (with 256 bit of entropy)
     * @param string $hashkey key for hashing (with 256 bit of entropy)
     * @return string Decrypted data
     * @throws Exception on internal error
     *
     * Based on code from: https://stackoverflow.com/a/46872528
     */
    public static function decrypt($encrypteddata, $cryptokey, $hashkey)
    {
        $method = "AES-256-CBC";
        $key = hash('sha256', $cryptokey, true);
        $iv = substr($encrypteddata, 0, 16);
        $hash = substr($encrypteddata, 16, 32);
        $cipherdata = substr($encrypteddata, 48);

        if (!hash_equals(hash_hmac('sha256', $cipherdata.$iv, $hashkey, true),
$hash))
        {
            $cryptokey = "**REMOVED**";
            $hashkey = "**REMOVED**";
            throw new \Exception("Internal error: Hash verification failed");
        }

        $plaindata = openssl_decrypt($cipherdata, $method, $key,
OPENSSL_RAW_DATA, $iv);

        if ($plaindata === false)
        {
            $cryptokey = "**REMOVED**";
            $hashkey = "**REMOVED**";
            throw new \Exception("Internal error: openssl_decrypt()
failed:".openssl_error_string());
        }

        return $plaindata;
    }
}
```

If you truly cannot have proper encryption and hash keys but have to use an user entered password as the only secret, you can do something like this:

```
/**
 * @param string $password user entered password as the only source of
 *   entropy to generate encryption key and hash key.
 * @return array($encryption_key, $hash_key) - note that PBKDF2 algorithm
 *   has been configured to take around 1-2 seconds per conversion
 *   from password to keys on a normal CPU to prevent brute force attacks.
 */
public static function generate_encryptionkey_hashkey_from_password($password)
{
    $hash = hash_pbkdf2("sha512", "$password", "salt$password", 1500000);
    return str_split($hash, 64);
}
```

Share Edit Follow

edited Jan 20, 2021 at 10:21

answered Apr 3, 2020 at 16:14

Mikko Rantalainen
15.1k 11 78 119

Nice edits, looks good! The one thing, like we previously discussed, is that this is vulnerable to brute-force through lookup tables because we trust the user to provide a "cryptokey" that has sufficient entropy. This problem could be fixed with a real KDF rather than SHA-256. Otherwise, looks good! – Luke Joshua Park Apr 6, 2020 at 7:18

@LukeJoshuaPark: Yeah, I think these methods would be the low level implementation using a real key. Perhaps I should add a method for using Key derivation function (KDF) to go from user password to encryption key. However, such method should not claim to magically have 256 bit of entropy from low quality user password. Instead, KDF logically is an injection from e.g. 32 bit key to 256 bit keyspace where attacker does not have an easy way to simply enumerate all the 2^32 possible keys out of 256 bit keyspace. – Mikko Rantalainen Apr 6, 2020 at 9:44

Assuming that we have only password (=no storage for salt), the KDF would need to be something like `hash_pbkdf2("sha256", $password, $password, 500000)` . I'm not sure if even that's enough with low quality passwords when we consider SHA-256 performance on GPUs. – Mikko Rantalainen Apr 6, 2020 at 9:57

@LukeJoshuaPark do you think it would be okay to generate hash key and encryption key from the same password? For example `$hash_key = hash_pbkdf2("sha256", "$password", "hash$password", 500000)` and `$encryption_key = hash_pbkdf2("sha256", $password, "enc$password", 500000)` . – Mikko Rantalainen Apr 6, 2020 at 10:03

1 Yes - although I'd recommend running PBKDF2 with SHA-512 rather than SHA-256 if you're going to do that. This allows the first 256 bits of output to be the encryption key and the last 256 bits of output to be the hash key. – Luke Joshua Park Apr 6, 2020 at 23:25

-1

If you are using PHP >= 7.2 consider using inbuilt sodium core extension for encrption.

Find more information here - `http://php.net/manual/en/intro.sodium.php` .

Share Edit Follow

answered Oct 7, 2018 at 13:07

M_R_K
6,116 1 41 43

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.