##### 绝不要以明文存储密码。

永远使用 **哈希算法** 来处理密码。

##### 绝不要使用 Base64 或其他编码方式来存储密码。

这和以明文存储密码是一样的，使用 **哈希** ，而不要使用 **编码** 。

编码以及加密，都是双向的过程，而密码是保密的，应该只被它的所有者知道， 这个过程必须是单向的。哈希正是用于做这个的，从来没有解哈希这种说法， 但是编码就存在解码，加密就存在解密。

##### 绝不要使用弱哈希或已被破解的哈希算法，像 MD5 或 SHA1 。

这些算法太老了，而且被证明存在缺陷，它们一开始就并不是为了保存密码而设计的。

另外，绝不要自己发明算法。

只使用强密码哈希算法，例如 BCrypt ，在 PHP 自己的 [密码哈希](http://php.net/password) 函数中也是使用它。

即使你的 PHP 版本不是 5.5+ ，也请使用它们，CodeIgniter 为你提供了这些算法。

如果你连升级 PHP 也无法做到，那么使用 *hash\_pbkdf() <http://php.net/hash\_pbkdf2>* 吧， 为实现兼容性我们提供了这个函数。

##### 绝不要以明文形式显示或发送密码。

即使是对密码的所有者也应该这样。如果你需要 "忘记密码" 的功能，可以随机生成一个新的 一次性的（这点很重要）密码，然后把这个密码发送给用户。

##### 绝不要对用户的密码做一些没必要的限制。

如果你使用除 BCrypt（它有最多 72 字符的限制）之外的其他哈希算法，你应该设置一个相对长一点的密码长度（例如 1024 字符），这样可以缓解 DoS 攻击。

但是除此之外，对密码的其他限制诸如密码中只允许使用某些字符，或者密码中不允许包含某些字符，就没有任何意义了。

这样做不仅不会提高安全性，反而 **降低了** 安全性，而且真的没有任何理由需要这样做。 只要你对密码进行哈希处理了，那么无论是技术上，还是在存储上都没有任何限制。

首先来看看各算法对比

必须明确一点：  
Bcrypt是单向Hash加密算法，类似Pbkdf2算法 不可反向破解生成明文。



# 使用 Bcrypt 代替 MD5/SHA1

如果数据库被“拖库”明文存储的密码就变得不安全。之前的做法是使用 md5 散列的方式，因为 md5 不可逆，无法从密文推出原文。

但是 HASH 算法最大的问题是，会发生撞库，也就是说，有可能出现多个原文得到同一个密码。

下面这个式子是存在的，如果原文是 M1，只需要另外一个同样 HASH 值的密码即可登录。

MD5(M1) = MD5(M2) = MD5(M3)

一种攻击方法是，攻击者记录了一张巨大的密码库，预先计算了常用密码的 hash 值，这样只需要搜索 hash 值就能寻找到一个合适的密码用于登录。

这就是被彩虹表攻击。

解决彩虹表的问题是加盐，在加密之前，对原文混入其他信息，混入的信息不存放到数据库中。实际寻找到其他原文也无法登录。

第二中攻击方法是王小云教授寻找到的一种新的方法，通过算法快速的找到 M2，这样不依赖彩虹表就可以实施攻击。

MD5(M1) = MD5(M2)

当被攻击者价值非常大，攻击者获取足够多的撞库原文，还是能分析盐值。

## Bcrypt

Bcrypt 有两个特点

每一次 HASH 出来的值不一样

计算非常缓慢

因此使用 Bcrypt 进行加密后，攻击者想要使用算出 M2 成本变得不可接受。但代价是应用自身也会性能受到影响，不过登录行为并不是随时在发生，因此能够忍受。对于攻击者来说，需要不断计算，让攻击变得不太可能。

因此推荐使用 Bcrypt 进行密码加密。

##### Php Bcrypt 进行密码加密。

每一次 password\_hash 运行结果都不一样，因此需要使用 password\_verify 函数进行验证。

|  |  |
| --- | --- |
| 1  2  3 | $password = '123456';  $hash = password\_hash($password, PASSWORD\_DEFAULT);  var\_dump(password\_verify($password, $hash)); |

password\_hash($password, $algo, $options) 的第三个参数 $options 支持设置至少 22 位的 salt。但仍然强烈推荐使用 PHP 默认生成的 salt，不要主动设置 salt。

$options = array('cost' => 11);

$newHash = password\_hash($password, PASSWORD\_DEFAULT, $options);

password\_verify($password, $newHash)

##### 例子： PHP> = 5.5-DEV

密码散列函数[现在已直接构建到PHP> = 5.5中](http://php.net/password_hash)。您现在可以使用[password\_hash()](http://php.net/password_hash)创建bcrypt任何密码的哈希值：

<?php

// Usage 1:

$hash =password\_hash('rasmuslerdorf', PASSWORD\_DEFAULT)."\n";

echo $hash;

// $2y$10$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

// For example:

// $2y$10$.vGA1O9wmRjrwAVXD98HNOgsNpDczlqm3Jq7KnEd1rVAGv3Fykk1a

// Usage 2:默认加盐方式

$options = [

'cost' => 11

];

$hash =password\_hash('rasmuslerdorf', PASSWORD\_BCRYPT, $options)."\n";

echo $hash;

// $2y$11$6DP.V0nO7YI3iSki4qog6OQI5eiO6Jnjsqg7vdnb.JgGIsxniOn4C

要根据现有的散列验证用户提供的密码，可以使用以下[password\_verify()](http://php.net/password_verify)方式：

<?php

// See the password\_hash() example to see where this came from.

$hash = '$2y$07$BCryptRequires22Chrcte/VlQH0piJtjXl.0t1XkA8pw9dMXTpOq';

if (password\_verify('rasmuslerdorf', $hash)) {

echo 'Password is valid!';

} else {

echo 'Invalid password.';

}