Esta irreversibilidad significa que un valor hash puede ser libremente distribuido o almacenado, **ya que sólo se utiliza para fines de comparación.** SHA significa [algoritmo de hash seguro](http://es.wikipedia.org/wiki/Secure_Hash_Algorithm).

La seguridad proporcionada por un algoritmo hash es sumamente dependiente de su capacidad de producir un único valor para un conjunto de datos dados. **Cuando una función hash produce el mismo valor para dos conjuntos de datos distintos, entonces se dice que se ha producido una colisión.** Una colisión aumenta la posibilidad de que un atacante pueda elaborar computacionalmente conjuntos de datos que proporcionen acceso a información segura o para alterar ficheros de datos informáticos.

# FUNCIONALIDAD

[**http://www.arumeinformatica.es/blog/encriptar-y-guardar-contrasenas-en-base-de-datos/**](http://www.arumeinformatica.es/blog/encriptar-y-guardar-contrasenas-en-base-de-datos/)

En realidad, aunque seguiremos llamándole "encriptado" por comodidad, lo que se suele hacer es pasar las contraseñas por una [función resumen](http://es.wikipedia.org/wiki/Hash) como MD5, SHA, guardar en la base de datos el "resumen" generado y descartar la contraseña.

Más adelante, para comprobar si un usuario ha puesto bien su contraseña, lo que haremos será encriptar lo que el usuario nos envía y comparar el resultado del cifrado con el que tenemos en nuestra base de datos.

// $db\_hash será lo que tenemos guardado en la base de datos

if ($db\_hash === sha1($password))

{

// Contraseña correcta

}

# AÑADIENDO SEMILLAS

Otra forma de ponérselo difícil al atacante es añadir una semilla a la contraseña antes de pasarla por la función resumen. De esta forma, aún tiene más complicado utilizar las tablas rainbow, al hacer la cadena de la contraseña más larga y más improbable que esté en la tabla rainbow.

Para ponérselo aún peor, lo mejor es que la semilla sea única para cada contraseña, de forma que no pueda hacerse una tabla rainbow con tu semilla incorporada, sino que tendría que hacerse una tabla para cada contraseña.

La forma de hacerlo sería así:

$salt = md5(uniqid(rand(), true)); // O incluso mejor si tuviese mayúsculas, minúsculas, caracteres especiales...

$hash = hash('sha512', $salt.$password); // Puede ponerse delante o detrás, es igual

unset($password);

// Guardar en base de datos el $hash y $salt

Luego, cuando queramos autenticar un usuario:

// $db\_salt será la semilla que tendremos en base de datos

// $db\_hash será el resumen que tenemos guardado en la base de datos

if ($db\_hash === hash('sha512', $db\_salt.$password))

{

// Contraseña correcta

}

[**http://www.dinvaders.com/como-encriptar-contrasenas-y-hacerlas-mas-seguras-con-php/**](http://www.dinvaders.com/como-encriptar-contrasenas-y-hacerlas-mas-seguras-con-php/)

# ¿Y cómo compruebo la contraseña?

Cuando el usuario se registra se guarda en la base de datos su nombre de usuario y su contraseña encriptada. Cuando éste va a loguearse completará el campo nombre de usuario y contraseña, si el hash de esta contraseña ingresada en el formulario de login coincide con el hash asociado al nombre de usuario guardado en la base de datos en el momento del registro, el usuario iniciará sesión.

Con la función crypt() reconocemos que el h

ash en la base de datos fue creado con Blowfish.

$passwordenBD = '$2a$07$yMoJrJpwEPrmVnZx4KIyNuOAiOMQksjkV1EW0YRgVe33eYe/yT60y';

if( crypt('micontraseña', $passwordenBD) == $passwordenBD) {

echo 'Es igual';

}

¿QUÉ ES SALT Y PARA QUE SIRVE?

Salt en resumen sería una línea de texto que se añade — en este caso — a una contraseña para que sea más compleja de descifrar. Les dejo un ejemplo simple así les queda más gráfico:

$salt = '34a@$#aA9823$';

$password = 'dinvaders1234';

$password = hash('sha256', $salt . $password);

El resultado sería un hash que contiene el salt que creamos más la contraseña