

Encriptar contraseñas

Porqué encriptar contraseñas

El **hash** de contraseñas es el método más elemental de seguridad que hay que tener en cuenta para asegurar las contraseñas de nuestros usuarios.

Se trata de un algoritmo matemático de un solo sentido (no permite la descryptación), transforma cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija. Independientemente de la longitud de los datos de entrada, el valor hash de salida tendrá siempre la misma longitud.

Nos protege tanto en el caso de que la BD se vea atacada como de los mismos empleados de la aplicación. Además de esta tendremos que aconsejar a nuestros usuarios el uso de contraseñas seguras y el cambio de estas cada cierto tiempo.

No todos los algoritmos son iguales

Cada hash utiliza un algoritmo para realizar la encriptación. Los dos más usados en el pasado son **MD5** y **SHA-1**. Pero actualmente no son recomendables porque los ordenadores de hoy en día pueden **romper** fácilmente estos **algoritmos**. Dependiendo de la **complejidad y longitud de la contraseña**, se puede **romper** en menos de una hora con los dos algoritmos nombrados (los ratios son 3650 millones de cálculos por segundo con MD5 y 1360 millones por segundo con SHA-1).

Por eso es importante usar **algoritmos complejos**, tardan algo más en realizar el encriptado pero es conveniente ya que se consigue que se tarde muchísimo más tiempo en crackearlo, en caso de que sea posible.

Otro problema es la existencia de **Rainbow Tables**, son **tablas de búsqueda inversa para hashes**. El creador de las tablas precalculó los hashes MD5 para palabras comunes, frases, palabras modificadas y strings aleatorios. La **facilidad de crackear un algoritmo MD5** lo hace posible la existencia de este tipo de tablas.

Generar este tipo de tablas para un **algoritmo complejo** tarda mucho más, pero es posible también. Una medida apropiada es **añadir un salt al hash**. En este contexto, salt es cualquier frase que se añade a la contraseña antes de crear el hash. Usando un salt se gana mucho terreno frente a este tipo de tablas. Se debería crear una **Rainbow Table** específica para tu **salt** y averiguar cual es el salt en tu aplicación, incluso utilizar diferentes **salt**.

Actualmente se aconseja blowfish.

Cómo encriptar en PHP

Podemos utilizar dos funciones que nos realizan la encriptación `password_hash` y `crypt`. Nosotros utilizaremos `password_hash`.

`password_hash`

`password_hash(string $password, integer $algo, array $options = ?): string`

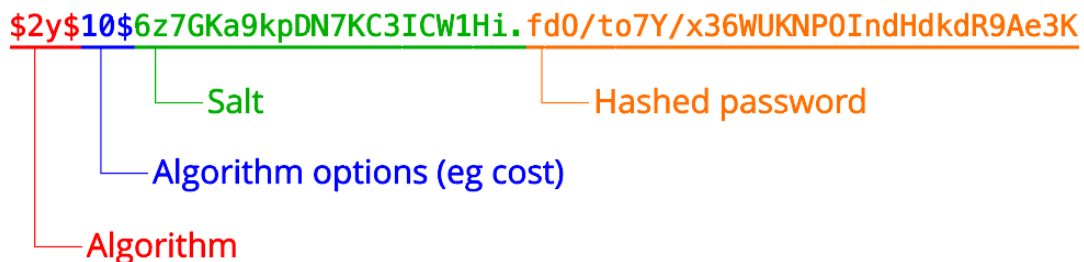
Crea un password muy complejo, incluyendo la generación de salts aleatorios.

Es necesario facilitar también el algoritmo que se desea emplear (parámetro \$algo). La mejor opción de momento es especificar **PASSWORD_DEFAULT** (se actualiza siempre que se añada un algoritmo nuevo más fuerte), aunque también es posible elegir otro concreto.

Tercer parámetro, opciones. Actualmente sólo permite el coste, el coste indica cuánto de complejo debe ser el algoritmo y por lo tanto cuánto tardará en generarse el hash. El número se puede considerar como el número de veces que el algoritmo hashea la contraseña, sino ponemos nada por defecto toma el 10.

La cadena obtenida como resultado (la llamaremos hash) contiene información del propio método de encriptación lo que nos permitirá realizar la verificación.

\$2y\$10\$6z7GKa9kpDN7KC3ICW1Hi.f**d0/to7Y/x36WUKNP0IndHdkdR9Ae3K**



```
<?php
/**
 * En este caso, queremos aumentar el coste predeterminado de BCrypt a 12.
 * Observe que también cambiamos a BCrypt, que tendrá siempre 60 caracteres.
 */
$opciones = [
    'cost' => 12,
];
echo password_hash("rasmuslerdorf", PASSWORD_BCRYPT, $opciones)."\n";
?>
```

Verificar contraseña

La librería hash de PHP incluye la función password_verify que pasándole la contraseña que introduce el usuario y el hash (contraseña encriptada) almacenado en la BD nos devolverá true si coinciden. Es la mejor forma de verificarlas.

```
If (password_verify($pass, $passBD))
    echo "Contraseña correcta";
```