

How to Encrypt User Passwords Using a Decorator

Most user management systems encrypt user passwords. This guide will show you how to quickly and easily take advantage of [Transfer Decorators](#) to handle encrypting of your user passwords and performing checks against them.

Why do you want to encrypt your passwords? To keep the honest, honest. It only takes one disgruntled employee with access to your database to grab user passwords and wreak havoc on your system, and potentially, to your user's lives. Many administrators and/or managers will suggest that you use `hash()` instead of `encrypt()` to do password encryption. Since a `hash()` is a one-way encryption, and therefore the resultant string cannot be decoded back to its original value, your user accounts are provided with more safety than a simple `encrypt()` offers.

Okay, let's get started. Time to get your favourite editor fired up and we want to create our `UserDecorator.cfc`. Be sure to use the `extends` attribute on your `<cfcomponent>` tag, and extend [transfer.com.TransferDecorator](#):

Fig. 1: Extend the TransferDecorator

```
<cfcomponent displayname="UserTransferObjectDecorator"
              extends="transfer.com.TransferDecorator"
              output="false">
```

Next on the agenda is to tell Transfer to use the decorator for our User. This is done in your [transfer.xml config file](#).

Fig. 2: Define the Decorator

```
<object name="User" table="tblUser" decorator="model.UserDecorator">
```

Okay, buckle up, because it's going to get hairy for a moment. We're now ready to write our method that handles hashing of the password. You ready? Alrighty then, let's see it...

Fig. 3: setPassword

```
<cffunction name="setPassword"
            hint="I set the hashed password"
            returntype="void"
            output="no"
            access="public">
    <cfargument name="password"
                hint="I am the password to hash/set"
                type="string"
                required="yes">

        <cfset getObject().setPassword(hash(arguments.password, "SHA-512")) />
</cffunction>
```

In your controller simply call `User.setPassword(form.password)` as you normally would, and it's now hashed.

That's it. Ridiculously simple. That's all that is required to hash your passwords with Transfer.

For those of you wanting to know more of the nitty-gritty behind what we've done, here's a more detailed explanation:

1. By adding a 'setPassword' method on our decorator, we are "overwriting" the publicly exposed generated method on the Transfer Object. What this means is that any time our application calls `setPassword()` on our User object, the method on the decorator is actually the method that answers the call. Now we can do whatever we need to do with the password before it becomes part of (is set on) the object, e.g., `hash()` it.
2. Our decorator method calls `getObject()`, which returns the actual TransferObject generated by Transfer. We're chaining this method call with the `setPassword()` method on the TransferObject, which is where we are setting the hashed (encrypted) password to.

3. By setting the password on the TransferObject, it is automatically saved (committed) behind the scenes by Transfer.
4. Because of the way the Transfer internals work, we also don't need to worry about having the Password re-encrypted when it gets pulled from the database. Transfer handles that for us!

Okay, one more quick example. Let's say that your user wants to change their password. We want to make sure that they supply their current password before allowing them to change it, but how do we know if they supplied the correct password?

Fig. 4: Check Password

```
<cffunction name="isPassword"
    hint="I return true if the provided password matches, false otherwise"
    returntype="boolean"
    output="no"
    access="public">
    <cfargument name="password"
        hint="I am the password to check"
        type="string"
        required="yes">

        <cfreturn (compare(hash(arguments.password, "SHA-512"), getPassword()) EQ 0) />
</cffunction>
```

With this method in place, our controller can now very easily check the password and perform the appropriate task:

Fig. 5: Controller Code (Change Password)

```
<cfif User.isPassword(form.password)>
    <!-- change the password -->
<cfelse>
    <!-- give them an error message -->
</cfif>
```

There you have it. Simple and easy. What other uses can you think of for a Transfer Decorator?

Categories: • [Decorators](#)