**Password Storage Research**

<https://www.mking.net/blog/password-security-best-practices-with-examples-in-csharp>

## **DO NOT: store passwords in a reversible format (ie. don't encrypt your passwords)**

I know the word 'encrypted' sounds secure, but you really shouldn't store passwords in any form that is reversible. If your system gets compromised, your 'secure' key will probably be taken just as quickly and easily as your password database is. End result: your 'encrypted' passwords are just as insecure as plain text ones.

## **DO NOT: hash your passwords with a general purpose hash function (such as MD5, SHA, etc.)**

Hashing your passwords is the best thing to do. Using a general purpose hash function (such as MD5, SHA, etc.) is NOT. In the past, 'conventional wisdom' was that hashing+salting your passwords was the way to go. These days, if you use a general-purpose hash function, you're doomed. Modern GPUs can calulate hashes very, very fast. Anyone who wants to throw some GPUs at your password database will be able to brute force billions of passwords per second. All the salt in the world won't save you at this point - nobody bothers with rainbow tables any more. Brute force will have you cracked in no time.

<https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rfc2898derivebytes?redirectedfrom=MSDN&view=netframework-4.8>

BCRYPT, SCRYPT, or PBKDF are fine. I strongly recommend using PBKDF, as it has an implementation built in to the .NET base class library ([Rfc2898DeriveBytes](https://msdn.microsoft.com/en-us/library/system.security.cryptography.rfc2898derivebytes%28v=vs.110%29.aspx)).

Store the following in your user database (alongside any additional data you need):

* Password salt
* Password hash
* Iterations / work factor

When a user creates an account:

* Generate a new salt.
* Generate a hash using the generated salt and the provided password.
* Save the salt, hash, and work factor in the database.

When a user tries to log in:

* Generate a hash using the provided password and the stored salt and work factor.
* If the hash generated above matches the stored hash, the password was correct; otherwise, the password was incorrect!

If you want to increase the work factor at a later date, write a script that will run on user login to:

* Verify the user's password by comparing the hash generated using the provided password, the stored salt, and the stored work factor with the stored hash.
* Generate a new password hash using the provided password, the stored salt, and the new (increased) work factor, and store the new password hash and the new work factor in the database.