Hashing your passwords

If you are using external services in your app, you probably will at some point have to hash a password and send in the hash. Why? Well, it isn’t good practice to send passwords over the wire, even under SSL or TLS, because man-in-the-middle attacks are so common. For this reason, sending a hash of the password, while it won’t prevent the attacker from using it later, will at least protect the password from being exposed for use elsewhere.

What’s a hash?

A hash is one-way encryption of some string. You can take a string like “This is my password” and hash it to “Z7463462c34b0c5e017bb95b4aabcd85” but you can’t change it back. This isn’t like symmetrical encryption that you use for encrypting something that you will need later – it is just used to make a value that is compared to some database field in order to prevent the service from having to store actual passwords.

Making a hash in WinRT

WinRT has built in encryption in Windows.Security.Cryptography. Do not, under any circumstances, try to write your own encryption algorithms. Use what is built in. I don’t care if you don’t trust Microsoft – they did a better job than you will, and if something goes wrong, you won’t go to jail.

Using the cryptography classes seems harder than it should because it does a lot more than hash passwords. Because of this flexibility, there are a few steps to getting what you want out of it.


2) Convert the string in question to UTF8, using convertStringToBinary.

3) Get a reference to the SHA512 algorithm using the openAlgorithm method. Right now, that’s the best they have – if you need more call Redmond!

4) Use hashData to do the actual hashing work.

5) Encode the result to hex using encodeToHexString.

Here is my little hashWithSha512 function – feel free to use it as is:

```javascript
function hashWithSha512(value) {
    var crypto = Windows.Security.Cryptography;
    var encodedString = crypto.CryptographicBuffer.convertStringToBinary(value, crypto.BinaryStringEncoding.utf8);
    var algorithmProvider = new crypto.Core.HashAlgorithmProvider.openAlgorithm(crypto.Core HashAlgorithmNames.sha512);
    var hashedValue = algorithmProvider.hashData(encodedString);
    return crypto.CryptographicBuffer.encodeToHexstring(hashedValue);
}
```

Many times, it is a requirement to hash multiple times, and you can implement that as a simple for loop in JavaScript. Know, however, that it is much slower than what you would expect in C# or C++. Make sure you follow the Windows Store guidelines for processes that will take over 50 milliseconds – make it asynchronous, and use the progress tag to show the user that the app is still working.

About the Author

Bill Sempf is a seasoned programmer and .NET evangelist specializing in .NET applications.

Available in print and e-book formats.