[WPF - Using PasswordBox in MVVM](http://briannoyes.net/2015/03/04/wpf-using-passwordbox-in-mvvm/)

One question that often pops up as people start digging into using the MVVM pattern is: “How do I use PasswordBox with MVVM?”

The basis for this question is usually that they tried, but quickly discovered that the Password property on PasswordBox is not a DependencyProperty, therefore it cannot be the target of a Binding, and therefore there is no straightforward way of reading and writing the value through a binding to a property in your ViewModel.

So the first thing you have to ask yourself before you pursue an MVVM-compatible solution is: Am I using PasswordBox just to get the password hiding behavior that is built in from a user experience perspective, or am I using it because I want the password encrypted in memory that the PasswordBox gives me?

If you are not sure why encrypting the value of your password entry in memory is important, I won’t go into the background of what kinds of exploits exist there in detail. The short answer is that if someone (or some code – as in a virus or worm) gets access to your program memory, they could steal your password and impersonate you to that program or system at some point. The chances of that are fairly remote, but bottom line if you want your program to be as secure as possible, it is a fairly easy attack vector to protect against. Let the PasswordBox does what it does, and don’t do something in your code that sits behind it that causes the password to be exposed in memory for long periods of time.

That means even if you could data bind the PasswordBox value into a string property on your ViewModel or Model, you shouldn’t. Because you would be leaving the value just as exposed in memory in your ViewModel property’s backing field as if the PasswordBox did not store it encrypted itself.

So what can you do in the context of MVVM? As long as you are just grabbing a reference to the SecureString that the PasswordBox stores the value in and holding on to that in your ViewModel or Model property, you are still protected. Then the only challenge in terms of MVVM is that there is no straightforward way to do that with a Binding and the properties exposed by the control.

Now you could wire up everything I am going to show here as code behind in the view that has the PasswordBox since this shouldn’t be something you will be doing all over the place in your app. But my intent here is to show you how you can follow the same patterns you do for other fields you are binding in your MVVM screens.

I’m going to show it using a Behavior – a chunk of reusable code that you can wire up easily in XAML. If you want more background on behaviors, check out my Pluralsight course Extending XAML Applications with Behaviors – <http://www.pluralsight.com/courses/extending-xaml-applications-behaviors>. A behavior is a great way to encapsulate code that bridges between an API on a control that is not conducive to data binding, and exposing that information from the behavior in a way that is conducive to data binding.

So the idea is to create a behavior that listens to the PasswordChanged event that PasswordBox will raise as the user types characters into the PasswordBox and use that to set some property on our ViewModel. To make the behavior reusable, we don’t want to couple it to the ViewModel type at all. So what we can do is expose a DependencyProperty on the behavior itself that can be bound to a property on your ViewModel. Then, when the event fires, we go set the property that the binding points to using the value of the SecurePassword property of the control, which is a reference to the SecureString being managed by the PasswordBox.

Here is what that behavior looks like in code:

1: public class PasswordBoxBindingBehavior : Behavior<PasswordBox>

2: {

3: protected override void OnAttached()

4: {

5: AssociatedObject.PasswordChanged += OnPasswordBoxValueChanged;

6: }

7:

8: public SecureString Password

9: {

10: get { return (SecureString)GetValue(PasswordProperty); }

11: set { SetValue(PasswordProperty, value); }

12: }

13:

14: public static readonly DependencyProperty PasswordProperty =

15: DependencyProperty.Register("Password", typeof(SecureString),

16: typeof(PasswordBoxBindingBehavior), new PropertyMetadata(null));

17:

18:

19: private void OnPasswordBoxValueChanged(object sender, RoutedEventArgs e)

20: {

21: var binding = BindingOperations.GetBindingExpression(this, PasswordProperty);

22: if (binding != null)

23: {

24: PropertyInfo property = binding.DataItem.GetType()

25: .GetProperty(binding.ParentBinding.Path.Path);

26: if (property != null)

27: property.SetValue(binding.DataItem, AssociatedObject.SecurePassword, null);

28: }

29: }

30:

31: }

You can see that in the OnAttached override, I hook the PasswordChanged event. I also expose a DependencyProperty of type SecureString named Password. Then, the only trick is how to set the underlying property of a Binding that is set on the Password property. We can’t simply set Password from inside the Behavior because that ends up replacing the current value, which is the reference to the Binding, with the new value, thus breaking the link to the underlying property the binding points to. So you have to obtain the binding expression for the property, and use that to go set the underlying property through the binding. You can see the (somewhat obscure) code in the behavior that does that for us.

Once you have that Behavior created, hooking it and the binding up to the PasswordBox is as simple as this:

1: <PasswordBox Grid.Row="1"

2: Grid.Column="1">

3: <i:Interaction.Behaviors>

4: <local:PasswordBoxBindingBehavior Password="{Binding Password}"/>

5: </i:Interaction.Behaviors>

6: </PasswordBox>

You will have to add an xmlns namespace for Blend SDK to bring in the Interaction.Behaviors attached property. Easiest way to do that is to build the project, go open the solution in Blend, and drag and drop the behavior from the Behaviors collection in the Assets panel onto the PasswordBox in the designer or the Objects and Timeline panel. I show how to do that in my course. You can see then it is just a matter of Binding the Password property on the behavior to the appropriate SecureString property on the ViewModel and let the behavior take care of transferring the value of the SecureString to the ViewModel property whenever it changes.

Because SecureString is a reference type, you could just set it once but doing so on each change ensures that if the binding hookup happens after the behavior is attached it will still work.

At some point you will have to pull the value out of the SecureString to pass it off to a Login API or validate it. To do that you write code that looks like the following:

1: public class AuthenticationService : IAuthenticationService

2: {

3: public bool Login(string username, SecureString password)

4: {

5: IntPtr passwordBSTR = default(IntPtr);

6: string insecurePassword = "";

7: try

8: {

9: passwordBSTR = Marshal.SecureStringToBSTR(password);

10: insecurePassword = Marshal.PtrToStringBSTR(passwordBSTR);

11: }

12: catch

13: {

14: insecurePassword = "";

15: }

16: return MockServiceProxyCall(username, insecurePassword);

17: }

18:

19: private bool MockServiceProxyCall(string username, string password)

20: {

21: if (username == "Brian" && password == "password") return true;

22: else return false;

23: }

24: }

The key thing is to leave it in the SecureString as long as possible while the value is at rest in a member variable or field on an object. Only pull out the value into an insecure string as briefly as possible and only in a stack variable (local variable in a method) so that it only lives in an unprotected state in memory on the stack as long as that stack frame (method) is executing.

So that is it, a somewhat straightforward, reusable way to bind WPF PasswordBox controls to properties on an MVVM ViewModel that still keeps the password value protected until you are really ready to use it.

You can [download the full sample using this behavior here](https://dl.dropboxusercontent.com/u/7366831/downloads/Blog/MvvmPasswordBox.zip).