Some choices for encrypting data so that it can be decrypted only by the same user or computer

devblogs.microsoft.com/oldnewthing/20240327-00

March 27, 2024

Raymond Chen

One of the functions that Windows makes available to you for encrypting data is `CryptProtectData`. (Previously.) This function gives you some options for who can decrypt the data:

- The same user on the same machine.
- Any user on the same machine.

The resulting binary blob can be saved anywhere, and can later be loaded back into memory by any process running within the same scope¹ and passed to `CryptUnprotectData` to recover the original data.

(Here's some sample code.)

Alternatively, you can use `Windows.Security.Cryptography.DataProtection.DataProtectionProvider`. When encrypting, you construct the `DataProtectionProvider` with a string that describes who can decrypt the data.

- C++/WinRT: `auto protector = DataProtectionProvider(scope);`
- C++/WinRT: `DataProtectionProvider protector{ scope };`
- C++/CX: `auto protector = ref new DataProtectionProvider(scope);`
- C#: `var protector = new DataProtectionProvider(scope);`
- JavaScript: `var protector = new DataProtectionProvider(scope);`

The `DataProtectionProvider` gives you a wider choice of scopes than `CryptProtectData`.

- Current user.
- Current machine.
- Specific Active Directory user or group. (Requires enterpriseData capability.)
- Specific Web site password.

Once you've created a `DataProtectionProvider`, you can use the `ProtectAsync` and `ProtectStreamAsync` methods to encrypt a binary blob or data stream (respectively). In both cases, the resulting binary blob or stream can be saved anywhere.

To decrypt, create a `DataProtectionProvider` using the default (0-parameter) constructor, and then call `UnprotectAsync` or `UnprotectStreamAsync` to recover the original data.

(Here's some sample code.)

Really, the `DataProtectionProvider` is two different features glued together. First is encryption, which requires a scope. Second is decryption, which rejects a scope. Clearer would have to been to keep the separate functionality in separate objects.

```
runtimeclass DataProtectionEncryptionProvider
{
    DataProtectionEncryptionProvider(String scope);

    IAsyncOperation<IBuffer> ProtectAsync(IBuffer data);
    IAsyncAction ProtectStreamAsync(IInputStream src, IOutputStream dest);
}

runtimeclass DataProtectionDecryptionProvider
{
    DataProtectionDecryptionProvider();

    IAsyncOperation<IBuffer> UnprotectAsync(IBuffer data);
    IAsyncAction UnprotectStreamAsync(IInputStream src, IOutputStream dest);
}
```

With this alternate design, you can't accidentally try to encrypt with an unscoped provider, and it removes the confusion over whether the scope you use for decryption needs to match the one that was used for encryption.

But really, the decryption provider is stateless, so the decryption methods can just be static, and that also avoids the confusion.

```
runtimeclass DataProtectionProvider
{
    DataProtectionProvider(String scope);

    IAsyncOperation<IBuffer> ProtectAsync(IBuffer data);
    IAsyncAction ProtectStreamAsync(IInputStream src, IOutputStream dest);

    static IAsyncOperation<IBuffer> UnprotectAsync(IBuffer data);
    static UnprotectStreamAsync(IInputStream src, IOutputStream dest);
}
```

Bonus chatter: The `DataProtectionProvider` is just a wrapper around the `NCryptProtect-Secret` and `NCryptUnprotectSecret` functions (for buffers) and te `NCryptStreamOpenToProtect` and `NCryptStreamOpenToUnprotect` functions (for streams).

Bonus bonus chatter: `CryptProtectData` and `DataProtectionProvider` are not interoperable. Data encrypted by one cannot be decrypted by the other.

[1] Sometimes people ask whether the memory must be decrypted from the same buffer returned by `CryptProtectData`. No, it doesn't. You can save the bytes to a file or copy them to another buffer. Just pass those same bytes back when you're ready to decrypt. (After all, if the memory must be decrypted from literally the same buffer, that would prevent cross-process decryption and render moot all of the discussion in the documentation about roaming users, since roaming users are decrypting from another computer entirely.)