.NET Envelope API

Protection settings for an application are typically specified by using the Sentinel LDK Envelope user interface.

For certain settings, the .NET Envelope API provides an alternate method for applying protection. Using this API, the developer can specify protection settings for classes and methods directly in the application's source code.

This is accomplished by employing a feature of the .NET framework known as *custom attributes*.

The .NET Envelope API provides custom attributes that simplify the protection process.

To use the.NET Envelope API, include the **Aladdin.HASP.Envelope** namespace in your source code.

For unprotected assemblies, the **Aladdin.HASP.Envelope.dll** assembly is required at compile-time and protection-time. This DLL is *not* required for a protected assembly.

Using the custom attributes, you can control the settings listed in the table that follows.

Name	Description	Type	Default
Protect	Protect this method	bool	true
FeatureId	Feature ID to use for protection	int	-1 (indicates global)
Encrypt	Encrypt this method	bool	true
CodeObfuscation	Obfuscate the entire code of this method	bool	false (see note below)
TreatCheckOnlyAsUnprotectable	If true, treat as "unprotectable" any weak method for which Sentinel LDK cannot provide full protection (but can offer only weak "Check-Only" protection). Show only fully-protected methods as protected, and mark all other methods as unprotectable.	bool	true
Frequency	When the license for this method is checked. Possible values are: • CheckOncePerApplication (0) • CheckOncePerInstance (1): Check once for each class instance. • CheckEveryTime (2): Check every time this method is invoked (this may result in a reduction in performance if the method is called frequently).	EnvelopeMethodProtectionFrequency	CheckOncePerApplication

Name	Description	Type	Default
MinCodeSizeForProtection	Minimum IL code size (in bytes) of methods that Envelope will protect by default. To improve performance, only methods with IL code size greater than the specified value will be protected by default. This enables you to set the protection criteria for the methods if you do not want small methods to be protected by default or there are many small methods in the application.	int	30
SymbolObfuscation	Obfuscate all symbols. Possible values are: • ObfuscateDefault (0): Obfuscate all except public names and virtual protected. • ObfuscateSkip (1): Turn off symbol obfuscation. • ObfuscateForce (2): Force to obfuscate all symbols.	EnvelopeSymbolObfuscation	ObfuscateDefault (0)

Note: Enabling code obfuscation imposes a significant load on the system, resulting in a reduction in performance. For this reason, the default for **CodeObfuscation** is **false**. SafeNet recommends that you obfuscate only algorithms or other key modules that represent important intellectual property.

You can specify these settings at assembly-level, class-level, and method-level. Class-settings overrule assembly-settings, and method-settings overrule class-settings.

The following is an example with source-code comments:

```
using Aladdin.HASP.Envelope;
using System;

namespace MyProgram
{
    /// Methods in this class do not get protected, because protection
    /// settings are inherited from assembly
    class MyExample
    {
        /// This method does not get protected
        static void Main(string[] args)
        {
            Console.WriteLine("{0} + {1} = {2}", 3, 4, new MyClass().Add(3, 4));
            Console.WriteLine("{0} * {1} = {2}", 3, 4, new MyClass().Multiply(3,
4));
        }
    }
```

```
    /// Protect all methods in this class with Feature ID 0 and Minimum code
        size for methods as 8
    [EnvelopeMethodProtectionAttributes(Protect = true, FeatureId = 0,
    MinCodeSizeForProtection = 8)]
    class MyClass
    {
        /// Protect the Add method using Feature ID 1, obfuscate the code,
        /// check the license every time this method is invoked
        [EnvelopeMethodProtectionAttributes(Protect = true, FeatureId = 1,
Encrypt = true,
        CodeObfuscation = true, Frequency =
EnvelopeMethodProtectionFrequency.CheckEveryTime)]
        public int Add(int a, int b)
        {
            return a + b;
        }

        /// protecting settings for this method are inherited from the settings
of the class
        public int Multiply(int a, int b)
        {
            return a * b;
        }
    }
}
```

Custom attributes in nested structures are inherited from the higher levels. However, when a lower level specified a different attribute than what is inherited from the higher-level attribute, the lower-level attributes overrides the higher-level attribute.

When it applies protection to an application, Sentinel LDK Envelope recognizes code that has been protected using custom attributes. Sentinel LDK Envelope allows the protection settings from the custom attributes to override settings specified in the Envelope user interface.