

dotNet Protector® Activation System

(verrou matériel)

Comment ça marche ?

Quand vous activez le verrou matériel, chaque méthode est cryptée, mais la clé de décryptage n'est pas enregistrée dans l'assembly protégé. La runtime dotNet Protector a besoin d'une clé de licence pour décrypter les méthodes.

Le verrou matériel est un processus en 3 étapes :

1. Vous protégez votre assembly en activant le verrou matériel. Les méthodes sont cryptées, seule votre clé publique est enregistrée dans l'assembly protégé – pas de clé de décryptage.
2. Votre assistant d'activation, protégé avec dotNet Protector construit une 'configuration matérielle' à partir du matériel sur lequel il s'exécute. Cette config est cryptée avec votre clé publique et envoyée à votre serveur d'activation.
3. Votre générateur de licences (serveur d'activation) décrypte la config et construit une clé de licence. Le générateur de licence s'exécute de votre côté et contient votre clé privée, permettant le décryptage de la 'config matérielle' et le cryptage de la clé de licence.

Quand votre assembly s'exécute, la runtime dotNet Protector cherche la clé de licence, la décrypte avec votre clé publique et essaie de décrypter les méthodes.

Les paires de clés publique/privée

dotNet Protector enregistre des clés symétrique et paires publique/privée (il y en a plusieurs) dans un jeu de clés. Le jeu de clés est construit la première fois que vous exécutez dotNet Protector, à l'aide d'un générateur aléatoire cryptographique (il y a vraiment peu de chance que le même jeu de clés soit généré deux fois).

Dans la mesure où une infrastructure de clé publique est très impliquée dans le processus d'activation, il est obligatoire que votre assembly et votre assistant d'activation soient protégés avec le même jeu de clés. Le générateur de licences utilisera la partie privée de ce jeu de clés pour générer les licences.

Activation de Composants (Dll)

dotNet Protector propose 2 modes d'activation de composants

Activer pour Exécuter : Dans ce mode, une licence est requise chaque fois que la dll s'exécute. Vous vendez votre dll suivant un modèle de redevance : chaque utilisateur final devra acquérir une licence pour l'utiliser.

Activer pour Referencer : Dans ce mode, une licence n'est requise qu'au moment de la compilation (licence design-time). Lors de l'exécution, une licence run-time est générée à partir de la licence design-time. Vous vendez votre dll suivant un modèle sans-redevance. Une licence est nécessaire pour le développeur qui utilise votre dll, mais pas pour le client final.

Le 3^{ème} choix proposé par dotNet Protector est finalement seulement un choix hybride : si votre dll est consommée par une application windows, elle fonctionne en mode 'activer pour référencer', si elle est consommée par une application ASPNET, elle fonctionne en mode 'activer pour exécuter'.

La solution exemple ActivationSample

Lockonly : Ce projet n'est pas impliqué dans l'activation. C'est un exemple bidon de programme à activer.

Productkey : Ce projet construit une clé produit (25 lettre et chiffres) à partir d'un entier 32 bits (pourrait être un compteur entier dans votre base de données de licences). La clé produit est construite à partir de votre jeu de clé privé ; cela rend difficile, connaissant un ID de licence, de deviner le suivant, empêchant les pirates de trouver les licences non encore activées en essayant tous les nombres de 32 bits.

Wizard : C'est un assistant d'activation. Il construit une chaîne de config à partir de

1. Une clé produit
2. Le logiciel à activer
3. Le matériel (ordinateur ou clé USB)

Webservice : C'est un générateur de licences, sous forme de service web ASPNET. A partir d'une config, il retourne la clé de licence correspondante. Pour faire une solution d'activation commerciale à partir de cet exemple, vous devrez le connecter à une base de données et vérifier la validité des licences et gérer les activation précédentes.

Keygen : C'est un générateur de licences 'tout intégré'. Il peut être utilisé comme outil pour générer des licences à usage interne, et vérifier votre système d'activation. Il construit une clé produit à partir d'un nombre, essaie de retrouver le numéro d'après la clé produit, construit une config à partir de cette clé, le logiciel à activer et le matériel, et finalement génère une clé de licence.

ActivationEngine : Ce projet interface les méthodes d'activation de dotNet Protector pour le service web.

Compiler la solution

1. Exportez votre jeu de clés (outils/exporter un jeu de clés du menu dotNet Protector) vers les répertoires ActivationEngine et Keygen.
2. Reconstituez la solution
3. Protégez wizard, keygen et lockonly (projets wizard.dpp, keygen.dpp et lockonly.dpp respectivement)
4. Essayez d'exécuter lockonly sans licence. (il doit se planter)
5. Lancez keygen sur lockonly (keygen 1000 lockonly.exe). Il doit générer lockonly.exe.license contenant la clé produit et la clé de licence.
6. Exécutez lockonly.exe. Il doit s'exécuter sur le même ordinateur et échouer sur un autre.

Compiler l'assistant:

Compilez le service web et déployez-le sur un serveur IIS.

L'assistant référence le service web à 'http://localhost/activation/sample/service.asmx' changez l'url pour qu'elle pointe sur le service. Compilez l'assistant et protégez-le avec wizard.dpp.

Protégez lockonly avec lockwiz.dpp (l'assistant sera embarqué dans l'exécutif).

Exécutez lockonly sans licence. L'assistant doit se lancer, activer le logiciel, et finalement exécuter le point d'entrée de lockonly si l'activation a réussi

Solution exemple Preactivator

Cette solution montre comment implémenter une solution de licence basée sur une clé usb sans service web (comme un dongle avec une clé usb standard).

Cette solution montre aussi comment embarquer dans un seul exécutable une version complète et une version démo d'un logiciel.

ProgramToActivate : C'est le projet bidon à activer. Il peut être compilé comme demo ou complet (la directive de compilation DEMO génère une version demo).

Preactivator : génère des licences pour clé usb.

Compiler solution

1. Exportez votre jeu de clés (outils/exporter un jeu de clés du menu dotNet Protector) vers le répertoire de la solution.
2. Recompilez preactivator
3. Protégez preactivator, demo et lockedprogram (projets preactivator.dpp, demo.dpp et lockedprogram.dpp respectivement)
4. Exécutez programtoactivate. Il doit s'exécuter en mode démo
5. Activez une clé usb de stockage avec preactivator et copiez le fichier de licence dans le même répertoire que lockedprogram.
6. Exécutez lockedprogram avec le dongle. Il doit s'exécuter en mode complet. Enlevez le dongle. Il doit se fermer (destruction sauvage du process)
7. Exécutez lockedprogram sans dongle. Il doit fonctionner en mode démo.

Solution exemple DllActivation

DllActivation montre comment activer une Dll, et aide à comprendre les 2 modes d'activation de dll fournis par dotNet Protector : activer pour exécuter et activer pour référencer.

NOTE: Les dlls protégées ont besoin de la runtime pour fonctionner.

1. PvLogiciels.dotNetProtector.Runtime.dll est la runtime commune
2. PvLogiciels.dotNetProtector.RuntimeV1 est la runtime v1.1 (x86 seulement)
3. PvLogiciels.dotNetProtector.RuntimeX86 est la runtime v2/x86
4. PvLogiciels.dotNetProtector.RuntimeAMD64 est la runtime v2/x64
5. PvLogiciels.dotNetProtector.Runtimetanium est la runtime v2/IA64

C'est un projet v2 1 et 3 si vous utilisez une version 32bit, 1 et 4 sur x64, 1 et 5 sur ia64

IMPORTANT: Avec les versions antérieures à v5.3.2816, dotNet Protector ne gère pas correctement les commutateurs d'activation de composant. Vérifiez que votre version de dotNet Protector est au minimum 5.3.2816 (?/A propos) avant d'utiliser l'activation de composant.

Pour remplacer votre moteur avec la dernière version, remplacez dotNetProtectorEngine.dll

<http://dotNetProtector.pvlog.com/downloads/dotNetProtector5/dotNetProtectorEngine32.zip> (x86)

<http://dotNetProtector.pvlog.com/downloads/dotNetProtector5/dotNetProtectorEngine64.zip> (x64)

DllToActivate : C'est la dll bidon que l'on est supposé protéger.

ActivateToRun : un exe simple qui consomme le composant. Il référence la version 'activer pour exécuter' de la dll.

ActivateToReference : Idem, mais référence la version 'activer pour référencer'.

Notez le fichier ActivateToReference.licx dans le projet: Il indique à VS de lancer LicenseCompiler (LC.EXE) pour générer une ressource de licence runtime.

La syntaxe du licx pour dotNet Protector est simple:

Le type pour lequel on demande une licence est toujours <dotNetProtector>, suivi par le nom de l'assembly du composant.

Ici : <dotNetProtector>, DllToProtect

Keygen : Le même que dans ActivationSample.

ActivateToRun.dpp : Projet dotNet Protector qui protège la dll avec l'option 'activer pour exécuter'

ActivateToReference.dpp : Projet dotNet Protector qui protège la dll avec l'option 'activer pour référencer'

Compiler la solution

1. Exportez votre jeu de clés (outils/exporter un jeu de clés du menu dotNet Protector) vers le répertoire de keygen.
2. Régénérez keygen et protégez le (keygen.dpp)
3. Protégez la dll avec ActivateToRun.dpp et ActivateToReference.dpp
4. Lancez Keygen pour chacune des dlls protégées
 - . keygen 1000 ActivateToRun\DllToActivate.dll
 - . keygen 1001 ActivateToReference\DllToActivate.dll
5. Régénérez ActivateToRefence.exe
6. Lancez ActivateToRun.exe : Une exception doit être levée en appuyant 'Dll Invoke'
7. Copiez ActivateToRun\DllToActivate.dll.license dans le répertoire ActivateToRun\bin\debug. ActivateToRun doit s'exécuter sans exception. ('Dll Invoke' doit ouvrir une fenêtre disant 'Hello, world')
8. Lancez ActivateToReference.exe. Aucune licence nécessaire à l'exécution. Il doit fonctionner sans fichier de licence.