

dotNet Protector® Activation System (hardware lock)

How it works

When you enable hardware lock, every method body is encrypted, but the decryption key isn't stored into the protected assembly. dotNet Protector's runtime needs a license key to decrypt method bodies.

Hardware locking-unlocking is a 3 steps process:

1. You protect your assembly with hardware loc enabled. Bodies are encrypted, only your public key is stored in the protected assembly – no decryption key.
2. Your activation assistant, protected with dotNet Protector builds a 'hardware configuration' from the hardware where it runs. This 'hardware configuration' is encrypted with your public key (stored by dotNet Protector into your assistant).
3. Your license generator (activation server) decrypts the 'hardware config' and builds a license key. The license generator runs on your side, and embeds your private key, allowing 'hardware config' decryption and license key encryption.

When your assembly runs, dotNet Protector Runtime looks for the license key, decrypts it with your public key and tries to decrypt bodies.

The public/private key pairs.

dotNet Protector stores symmetric and public/private key pairs (there are more than one) in a key set. The keyset is built the first time you run dotNet Protector, with help from a cryptographic random generator (there is minimal likelihood the same keyset is produced twice).

Since pki is strongly involved in activation process, this is mandatory that your assembly and your activation assistant are protected with the same keyset. Your license generator will use the private part of this keyset to build licenses.

Component (Dll) licensing

dotNet Protector® offers 2 modes for component licensing

Activate to Run : In this mode, a license is mandatory each time the dll runs. You sell your Dll in a royalty model: every final user needs a license to use it.

Activate to Reference : In this mode, a license is needed only when compiling the client application (this is the design-time license). A run-time license is generated from the design-time license when your dll runs. You sell your dll in a 'royalty free' model. A license is needed to consume your dll, but can be deployed with a client application.

The 3rd choice in dotNet Protector is in fact only a hybrid mode: If your dll is consumed by a windows application, it works in 'activate to reference' mode, when consumed by a ASPNET application, it runs in 'activate to run' mode.

ActivationSample sample solution

Lockonly : this project isn't implied in activation. This is just a dummy sample of some program to activate.

Productkey : this project builds a product key (25 numbers and digits) from a 32 bits ID (can be an integer counter in your license database). The product key is built from your private keyset; this makes difficult, knowing a license ID to know the next one, preventing hacker from finding un-activated licenses by trying each 32 bit number.

Wizard : this is an activation wizard. It builds a configuration string from

1. A product key
2. The software to activate
3. A piece of hardware (computer or USB storage)

Webservice : This a license generator, designed as an ASPNET web service. From a config string, it returns a matching license key. To build a commercial activation service, you'll need to connect it to a database and check license validity and handle previous activations.

Keygen : this is an 'all in one' license generator. It can be used as a tool to build 'internal use' licenses, and to verify your activation system. It builds a product key from a number, tries to retrieve the number from the product key, builds a 'config string' from this key, the software to activate and hardware, and finally builds a license key.

ActivationEngine : This project wraps dotNet Protector Activation Methods for the web service.

Building the solution

1. Export your keyset (from dotNet Protector's menu Tools/Export Keyset) to the ActivationEngine and the Keygen directories.
2. Rebuild all the solution
3. Protect wizard, keygen and lockonly (wizard.dpp, keygen.dpp and lockonly.dpp projects respectively)
4. Try to run lockonly without a license key. (It should crash)
5. Run keygen against lockonly (keygen 1000 lockonly.exe). It should build lockonly.exe.license containing the product key and the license key.
6. Run lockonly.exe. It should run on the same computer and fail on another.

Building the wizard:

Build the webservice and deploy it on a IIS web server.

The wizard references the webservice at 'http://localhost/activationssample/service.asmx' change the url to reflect the working one. Build the wizard and protect it using wizard.dpp.

Protect lockonly using lockwiz.dpp (the wizard will be embedded into your exe).

Run lockonly without a license file. It should run the wizard, activate the software, and finally launch lockonly's entry point if successfully activated.

Preactivator sample solution

This solution shows how to implement a 'usb key licensing' without a webservice. The goal is to send usb keys as license (like a hardware dongle with a standard storage pen).

This solution shows also how to embed a 'full featured licensed' and a 'limited demo' versions in a single executable.

ProgramToActivate : this is the dummy project to activate. It can be compiled as demo or full featured (the DEMO compilation switch turns the program into a demo version).

Preactivator : builds license keys from usb keys.

Building the solution

1. Export your keyset (from dotNet Protector's menu Tools/Export Keyset) to the solution directory.
2. Rebuild preactivator
3. Protect preactivator, demo and lockedprogram (preactivator.dpp, lockedprogram.dpp projects respectively)
4. Run programtoactivate. It should run in demo mode.
5. Activate a storage pen with preactivator and copy the license file in the same directory as lockedprogram.
6. Run lockedprogram with the dongle attached. It should run in full mode. Remove the dongle. It should exit (savage process killing)
7. Run lockedprogram without the dongle attached. It should run in demo mode.

DllActivation sample solution

DllActivation shows how to activate a Dll, and helps understand the 2 dll activation mode available in dotNet Protector: activate to run and activate to reference.

NOTE: Protected dlls need the runtime to work.

1. PvLogiciels.dotNetProtector.Runtime.dll is the common runtime
2. PvLogiciels.dotNetProtector.RuntimeV1 is the v1.1 (x86 only) runtime
3. PvLogiciels.dotNetProtector.RuntimeX86 is the v2/x86 runtime
4. PvLogiciels.dotNetProtector.RuntimeAMD64 is the v2/x64 runtime
5. PvLogiciels.dotNetProtector.RuntimeIA64 is the v2/IA64 runtime

This is a v2 project. 1 and 3 are needed if you run in a 32bit process, 1 and 4 on x64, 1 and 5 on ia64

IMPORTANT: With versions prior to v5.3.2816, dotNet Protector doesn't correctly handle component activation switches. Verify your dotNet Protector version is at least 5.3.2816 (?/about) before you use component activation.

To replace your engine with the latest version replace dotNetProtectorEngine.dll

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

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

DllToActivate : This is the dummy dll we are supposed to protect.

ActivateToRun : a simple exe that consumes the dll. This exe references the 'activate to run' protected version of the dll.

ActivateToReference : the same as above, but references the 'activate to reference' version.

Notice the ActivateToReference.licx file in project: it tells VS to launch LicenseCompiler (LC.EXE) to build a runtime license resource.

The licx syntax for dotNet Protector is straightforward:

The licensed type is always <dotNetProtector>, followed by your component assembly name.

In this case : <dotNetProtector>, DllToProtect

Keygen : the same as in ActivationSample.

ActivateToRun.dpp : dotNet Protector Project that protects the dll with 'activate to run' option

ActivateToReference.dpp : dotNet Protector Project that protects the dll with 'activate to Reference' option

Building the solution

1. Export your keyset (from dotNet Protector's menu Tools/Export Keyset) to the keygen directory.
2. Rebuild keygen and protect it (keygen.dpp)
3. Protect the dll using ActivateToRun.dpp and ActivateToReference.dpp
4. Run Keygen for each protected dll
 - . keygen 1000 ActivateToRun\DllToActivate.dll
 - . keygen 1001 ActivateToReference\DllToActivate.dll
5. Rebuild ActivateToRefence.exe
6. Run ActivateToRun.exe : An exception should occur when you click 'Dll Invoke'
7. Copy ActivateToRun\DllToActivate.dll.license to ActivateToRun\bin\debug. ActivateToRun should run without exception. ('Dll Invoke' should display a dialog saying 'Hello, world')
8. Run ActivateToReference.exe. No license needed at run time. It should work without a license file.