Signer nos composants logiciels

Pour des questions de sécurité, il est aujourd'hui nécessaire de s'assurer que les composants externes liés à SQL Server sont authentifiés et réputés fiables.

Vous allez découvrir ici comment signer vos composants logiciels à l'aide d'un certificat d'authentification. Cette approche n'est pas uniquement destinée à l'usage d'un composant au sein du serveur SQL mais devrait être la règle pour tout composant déployé sur un poste de travail ou serveur d'entreprise.

Vous allez ici signer vos assemblages avec un certificat d'authentification généré en local pour des tests. Dans la vraie vie, il serait nécessaire de recourir à un certificat fourni par une autorité de certification (Verisign par exemple) après dépôt de documents permettant de vous authentifier.

Les différentes étapes :

- Créez un certificat, si vous n'en avez pas déjà un.
- Créez un fichier .pfx à partir du certificat. Pfx stocke les clés qui vous permettront, associées à un mot de passe, de signer votre assembly.
- Signez la dll avec le fichier .pfx. A partir des propriétés du projet dans Visual Studio ou avec l'outil en mode de commande.

3 outils du SDK Windows seront ici utilisés.

Ils se trouvent sur ma machine à l'emplacement C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64.

- MakeCert : création d'un certificat
- PVK2PFX : création d'un fichier pfx pour transporter les données du certificat. Ce fichier sera ensuite utilisé pour signer nos composants
- Signtool : signature de nos composants avec un fichier contenant les infos d'un certificat

A noter : Pour simplifier l'encodage des commandes, utilisez comme base les fichiers de commandes batch (.bat) fournis. Toutes les commandes doivent être exécutées en tant qu'administrateur.

Produire le certificat (CertificatCles.bat)

Ici, nous allons donc auto-générer un certificat de test pour nous permettre de signer nos composants et respecter les contraintes imposées en matière de sécurité avec l'outil **MakeCert**.

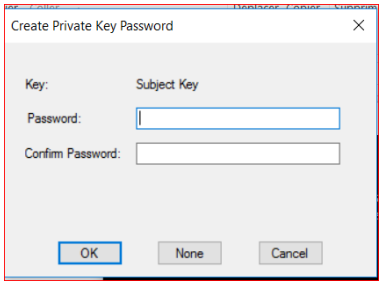
Rechercher l'outil MakeCert qui permet de générer un certificat

A partir du prompt de la ligne de commande.

```
makecert -r -pe -n "CN=Bost Certificat Test"
      -a sha256 -sky signature -cy authority
      -sv NielsTestPvk.pvk -len 2048
      -m 144 BostCertificat.cer
```

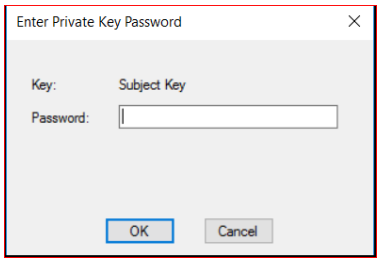
-r - Crée un certificat auto-signé.
-pe - Rendre la clé privée générée exportable.
-n - Le nom du certificat (CN pour certificate name suivi de = et du nom)
-a - L'algorithme de synthèse de la signature.
-sv - Le fichier PVK à créer.
-m - Nombre de mois pour lesquels le certificat sera valide.

A l'exécution, vous obtenez le prompt suivant :



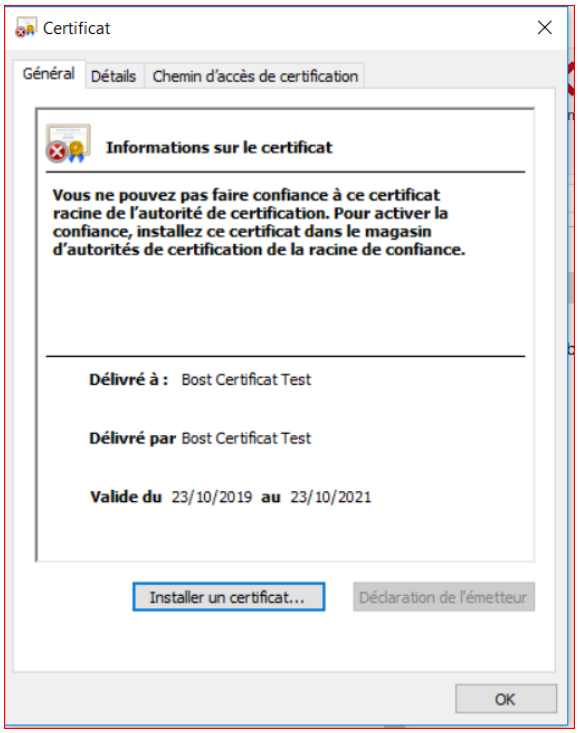
Définissez un mot de passe pour usage de votre clé privée

Puis



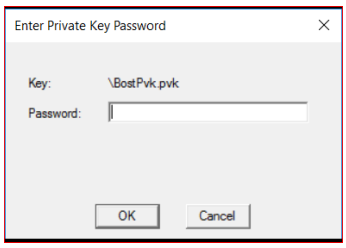
Renseignez ici la clé privée définie précédemment pour la création du certificat

Consultez le répertoire. Vous devriez y trouver un certificat (extension cert). Ouvrez le fichier par double click.



Produire le fichier pfx (CertificatClesPfx.bat)

Vous allez maintenant générer un fichier pfx à partir de ce certificat. Le fichier pfx servira pour signer vos composants. Il permet de stocker dans un format binaire le certificat avec sa clé privée. Le fichier .pfx contenant le certificat est protégé par mot de passe. Ce mot de passe sera exigé pour la signature de vos composants.



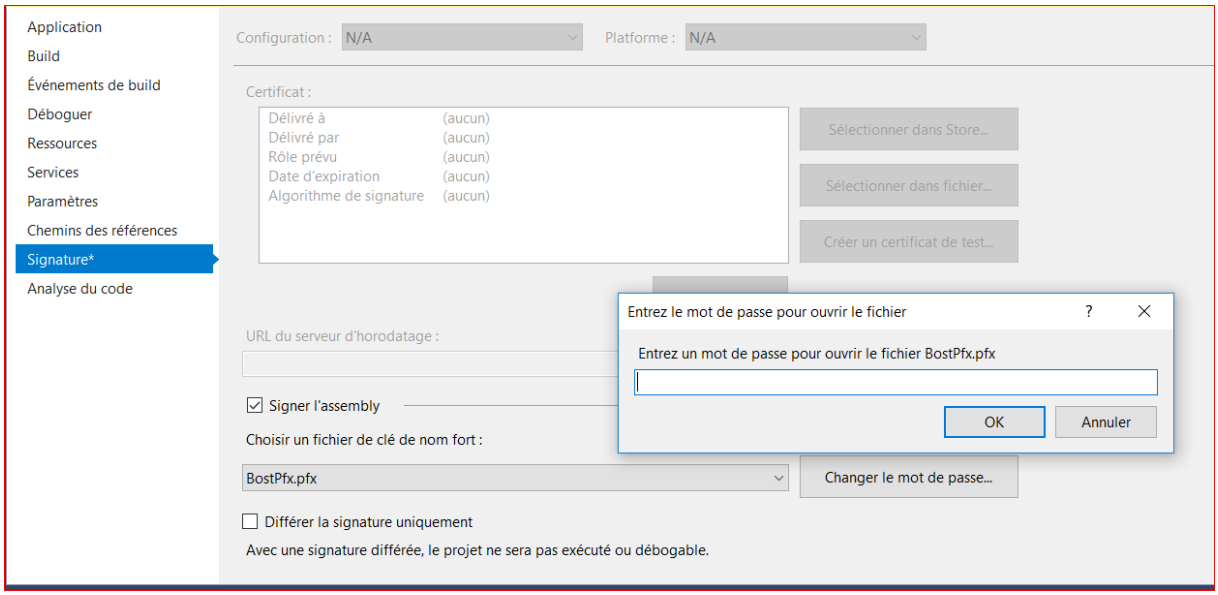
Entrez votre clé privée (celle communiquée à l'étape précédente)

Signer ses composants logiciels (SignatureDll.bat)

Vous pouvez maintenant utiliser le fichier pfx créé pour signer vos assemblages.

Vous pouvez le faire via l'interface de Visual Studio ou avec un outil en ligne de commande (fichier bat)

Par le biais de Visual Studio



.

En ligne de commande (SignatureDll.bat) Extrait

```
cd "C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64"
signtool sign /f "C:\Certificat\BostPfx3.pfx" /p vincent "C:\01 CDA\M3
```

Au niveau du fichier de commande, l'option /p vincent permet de définir le mot de passe de votre clé privée.

Vous avez maintenant un composant (une bibliothèque de fonctions signée)

Pour que celle-ci puisse fonctionner avec SQL serveur, il reste quelques étapes à réaliser…

Inscription de la Dll au niveau de SQL Server

Au niveau de SQL Server, il vous faut encore :

- Créez un certificat SQL Server à partir du certificat d'origine.
- Créez un identifiant (login) à partir du certificat.
- Accordez les autorisations de connexion nécessaires (UNSAFE ASSEMBLY dans notre exemple).
- Créez la DLL dans la base de données.
- Activer la prise en charge du clr
- Créez un wrapper pour les fonctions externes.

Création du certificat

A partir du certificat précédemment créé ou un certificat existant.

```
CREATE CERTIFICATE BostTestCert
FROM FILE = 'C:\Certificat\BostCert.cer';
GO
```

Création d'un login

Création d'un login à partir du certificat.

```
CREATE LOGIN login_BostTestCert
FROM CERTIFICATE BostTestCert
GO
```

Accordez les droits au login.

```
GRANT UNSAFE ASSEMBLY TO login_BostTestCert;
```

Approuver le code de la dll

Modification du paramétrage de la base de données

Le paramètre TRUSTWORTHY définit si SQL Server approuve le code dans la base de données particulière pour accéder aux ressources externes (un peu comme : «Faites-moi confiance, je suis un professionnel!»). Pour l'activer, utilisez la syntaxe ALTER DATABASE.

```
ALTER DATABASE StagiaireDB
SET TRUSTWORTHY ON;
GO
```

Activer la prise en charge du clr

```
EXEC sp_configure 'clr enabled' , '1';
RECONFIGURE;
```

Créer (lier) l'assembly au niveau de SQL Server

Toujours avec les permissions UnSafe. From désigne le path de la dll liée.

```
CREATE ASSEMBLY [IntegrationSQL]
FROM 'C:\01 CDA\M3 - Dév Persistance Données\04 Implémenter la DB\Corrections\A
WITH PERMISSION_SET = UNSAFE;
```

Créez un wrapper pour vos fonctions

Du point de vue de SQL, vous donnez l'impression d'exécuter une fonction SQL.

Pourtant, celle-ci est une fonction externe.

C'est le principe du wrapper utilisé dans de nombreux contextes différents. Un wrapper de fonction est une fonction dont l'objectif principale est d'appeler une autre fonction….

```
USE StagiaireDB
GO
-- Suppression si existe
IF  EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[VerifierExpReg]') AND
type in (N'FN', N'IF', N'TF', N'FS', N'FT'))
DROP FUNCTION [dbo].[VerifierExpReg]
GO

-- Création

CREATE FUNCTION [dbo].[VerifierExpReg](@texte [nvarchar](max), @modele [nvarchar](255))
RETURNS [bit] WITH EXECUTE AS CALLER
AS
EXTERNAL NAME [IntegrationSQL].[TypesSQL.ExpressionsRegulieres].[ExpressionRegIsMatch]
GO
```

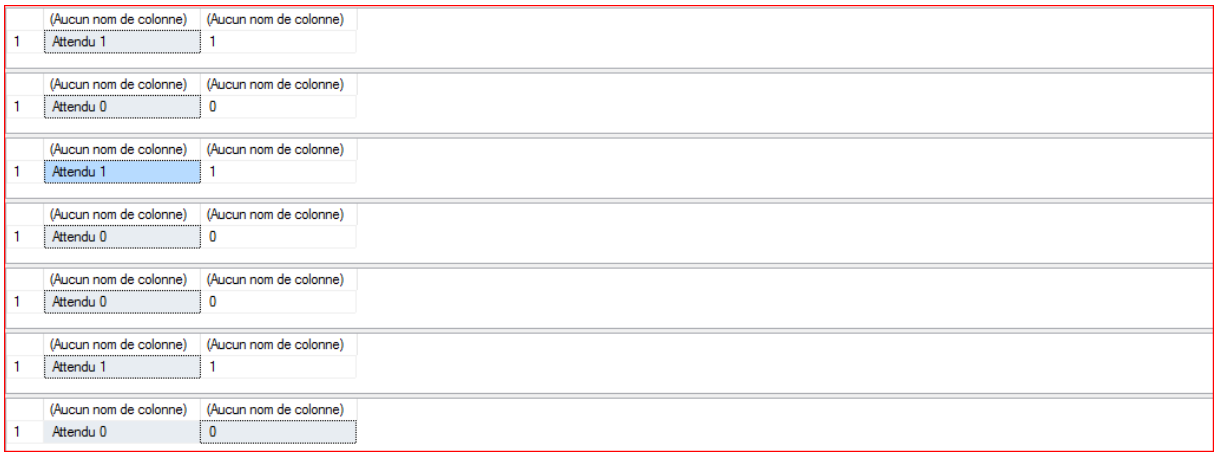
Tests de la fonction

```
select 'Attendu 1',dbo.VerifierExpReg('ab','a')
Select 'Attendu 0',dbo.VerifierExpReg('ab','c')

Select 'Attendu 1',dbo.VerifierExpReg('01346','^[0-9]{5}$')
Select 'Attendu 0',dbo.VerifierExpReg('0A346','^[0-9]{5}$')
Select 'Attendu 0',dbo.VerifierExpReg('0134','^[0-9]{5}$')

Select 'Attendu 1',dbo.VerifierExpReg('01345','^\d{5}$')
Select 'Attendu 0',dbo.VerifierExpReg('0134','^\d{5}$')
```

Résultats



	(Aucun nom de colonne)	(Aucun nom de colonne)
1	Attendu 1	1

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	Attendu 0	0

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	Attendu 1	1

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	Attendu 0	0

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	Attendu 0	0

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	Attendu 1	1

	(Aucun nom de colonne)	(Aucun nom de colonne)
1	Attendu 0	0