La cryptographie en C#

Par Daniel MINKO FASSINOU

Date de publication : 22 novembre 2016

Le but de cet article est de présenter les classes implémentant la cryptographie en C#. Découvrons comment la cryptographie, et le hachage sont définis dans ce langage.

Pour réagir au contenu de ce tutoriel, un espace de dialogue vous est proposé sur le forum.
Commentez

I - Généralités

Petit rappel : la cryptographie désigne l'ensemble des techniques utilisées pour chiffrer un message. Il existe deux types d'algorithmes de chiffrement :

- **symétrique** : une même clé est utilisée pour chiffrer et déchiffrer les messages ;
- **asymétriques** : il existe une clé publique pour le chiffrement et une clé privée pour déchiffrer les messages.

En .NET, trois grandes familles de classes implémentent la cryptographie :

- **Les classes managées**. Leurs noms se terminent en **Managed**. Par exemple, la classe **RijndaelManaged** implémente l'algorithme Rijndael.
- Les classes issues de l'API de cryptographie de **Microsoft CAPI**. Leur suffixe est **CryptoServiceProvider**. Exemple : la classe **DESCryptoServiceProvider** implémente l'algorithme DES. Cette API n'évolue plus et est présente uniquement sur les anciens systèmes.
- Les classes de l'API CNG (**Cryptography Next Generation**). C'est la nouvelle API de cryptographie disponible à partir de Windows Vista. Ses classes se terminent en **Cng**. Exemple, la classe **ECDiffieHellmanCng** qui implémente l'algorithme ECDH (*Elliptic Curve Diffie-Hellman*).

II - Le chiffrage symétrique

Toutes les classes implémentant les algorithmes symétriques héritent de la classe **System.Security.Cryptography.SymmetricAlgorithm**.

II-A - Principales propriétés de cette classe

- **IV** (Byte[]) : le vecteur d'initialisation utilisée pour chiffrer et déchiffrer les données.
- **Key** (Byte[]) : la clé utilisée pour le chiffrage et le déchiffrage.

Le chiffrage symétrique nécessite une clé et un vecteur d'initialisation (IV) pour chiffrer et déchiffrer les données. En utilisant le constructeur par défaut, ces deux propriétés sont automatiquement créées.

II-B - Principales méthodes de cette classe

- **void Clear()** : libère les ressources utilisées par l'objet ;
- **ICryptoTransform CreateDecryptor(Byte[], Byte[])** : crée l'objet de déchiffrage à l'aide de la clé et du IV ;
- **ICryptoTransform CreateEncryptor(Byte[], Byte[])** : crée l'objet de chiffrage à l'aide de la clé et du IV ;
- **void GenerateIV()** : génère un nouvel IV ;
- **void GenerateKey()** : génère une nouvelle clé.

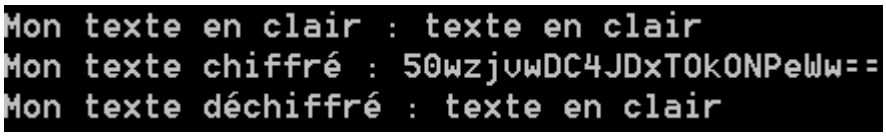
Le framework .NET implémente cinq algorithmes symétriques :

Algorithmes	Classes
AES	AesManaged, AesCryptoServiceProvider
DES	AesManaged, AesCryptoServiceProvider
RC2	RC2CryptoServiceProvider
Rijndael	RijndaelManaged
TripleDES	TripleDESCryptoServiceProvider

Exemple 1 :

```
1. static void Main(string[] args)
2. {
3.     // À la création de l'instance de chiffrage, la clé et le IV sont également créés
4.     TripleDESCryptoServiceProvider TDES = new TripleDESCryptoServiceProvider();
5.
6.     byte[] iv = TDES.IV;
7.     byte[] key = TDES.Key;
8.
9.     string text = "texte en clair";
10.
11.     Console.WriteLine("Mon texte en clair : {0}", text);
12.
13.     // La même clé et le même IV sont utilisés pour le chiffrage et le déchiffrage
14.     byte[] cryptedTextAsByte = Crypt(text, key, iv);
15.
16.     Console.WriteLine("Mon texte chiffré : {0}", Convert.ToBase64String(cryptedTextAsByte));
17.
18.     String decryptedText = Decryp(cryptedTextAsByte, key, iv);
19.
20.     Console.WriteLine("Mon texte déchiffré : {0}", decryptedText);
21. }
22.
23. static byte[] Crypt(string text, byte[] key, byte[] iv)
24. {
25.     byte[] textAsByte = Encoding.Default.GetBytes(text);
26.
27.     TripleDESCryptoServiceProvider TDES = new TripleDESCryptoServiceProvider();
28.
29.     // Cet objet est utilisé pour chiffrer les données.
30.     // Il reçoit en entrée les données en clair sous la forme d'un tableau de bytes.
31.     // Les données chiffrées sont également retournées sous la forme d'un tableau de bytes
32.     var encryptor = TDES.CreateEncryptor(key, iv);
33.
34.     byte[] cryptedTextAsByte = encryptor.TransformFinalBlock(textAsByte, 0,
 textAsByte.Length);
35.
36.     return cryptedTextAsByte;
37. }
38.
39. static string Decryp(byte[] cryptedTextAsByte, byte[] key, byte[] iv)
40. {
41.     TripleDESCryptoServiceProvider TDES = new TripleDESCryptoServiceProvider();
42.
43.     // Cet objet est utilisé pour déchiffrer les données.
44.     // Il reçoit les données chiffrées sous la forme d'un tableau de bytes.
45.     // Les données déchiffrées sont également retournées sous la forme d'un tableau de bytes
46.     var decryptor = TDES.CreateDecryptor(key, iv);
47.
48.     byte[] decryptedTextAsByte = decryptor.TransformFinalBlock(cryptedTextAsByte, 0,
 cryptedTextAsByte.Length);
49.
50.     return Encoding.Default.GetString(decryptedTextAsByte);
51. }
```

Résultat de l'exécution :



III - Le chiffrage asymétrique

Les algorithmes asymétriques sont moins rapides que les algorithmes symétriques. Les classes implémentant les algorithmes asymétriques héritent de la classe **System.Security.Cryptography.AsymmetricAlgorithm**.

Algorithmes	Classes
DSA	DSACryptoServiceProvider
ECDiffieHellman Elliptic Curve Diffie-Hellman	ECDiffieHellmanCng
ECDsa Elliptic Curve Digital Signature Algorithm (ECDSA)	ECDsaCng
RSA	RSACryptoServiceProvider

Exemple 2 :

```
1. static void Main(string[] args)
2. {
3.     // Cet objet est utilisé par le service de chiffrement
4.     // pour créer les clés
5.     CspParameters cspParams = new CspParameters();
6.
7.     // Les clés publique et privée
8.     RSAParameters privateKeys;
9.     RSAParameters publicKeys;
10.
11.     using (var rsa = new RSACryptoServiceProvider(cspParams))
12.     {
13.         // Génère la clé publique et la clé privée
14.         privateKeys = rsa.ExportParameters(true);
15.         publicKeys = rsa.ExportParameters(false);
16.
17.         rsa.Clear();
18.     }
19.
20.     string texte = "Allo le monde";
21.
22.     Console.WriteLine("Texte  en clair {0}", texte);
23.
24.     // La clé publique est utilisée pour chiffrer les données
25.     byte[] cryptedBytes = Encrypt(texte, publicKeys);
26.
27.      Console.WriteLine("Texte chiffré {0}", Convert.ToBase64String(cryptedBytes));
28.
29.     // La clé privée est utilisée pour déchiffrer les données
30.     Console.WriteLine("Text déchiffré {0}", Decrypt(cryptedBytes, privateKeys));
31. }
32.
33. static byte[] Encrypt(string value, RSAParameters rsaKeyInfo)
34. {
35.     using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
36.     {
37.         // Récupère les infos de la clé publique
38.         rsa.ImportParameters(rsaKeyInfo);
39.
40.         byte[] encodedData = Encoding.Default.GetBytes(value);
41.
42.         // Chiffre les données.
43.         // Les données chiffrées sont retournées sous la forme d'un tableau de bytes
44.         byte[] encryptedData = rsa.Encrypt(encodedData, true);
45.
46.         rsa.Clear();
47.
48.         return encryptedData;
49.     }
50. }
51.
52. static string Decrypt(byte[] encryptedData, RSAParameters rsaKeyInfo)
53. {
54.     using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())
55.     {
56.         // Récupère les infos de la clé privée
57.         rsa.ImportParameters(rsaKeyInfo);
58.
59.         // Déchiffre les données.
```

```
60.         // Les données déchiffrées sont retournées sous la forme d'un tableau de bytes
61.         byte[] decryptedData = rsa.Decrypt(encryptedData, true);
62.
63.         string decryptedValue = Encoding.Default.GetString(decryptedData);
64.
65.         rsa.Clear();
66.
67.         return decryptedValue;
68.     }
69. }
```

Résultat de l'exécution :

```
Texte  en clair Allo le monde
Texte chiffré ejTjSlYOO70dEflYyk/kajWnB/qJgKL6E1PeD1XQgfTj0HH3XXYuMBDdEW0PHtRawn
gaztyMEuKC+tka3p+/6sbm7UUnINAUWWFcE6qMO0xGK8v574q2oSppYdfffp2RyxL+C8C0w61PQLoejF
rb+qQIUE67KJ8wsvXk/zgmrc0=
Texte déchiffré Allo le monde
```

En pratique, les algorithmes asymétriques sont utilisés pour échanger les clés de chiffrement symétrique.

IV - Hachage

Impossible de parler de cryptographie, sans parler de hachage. Le hachage est un processus qui, à partir d'une donnée en entrée, produit une chaîne de longueur fixe, l'empreinte.

Les fonctions de hachage sont utilisées par exemple pour permettre l'authentification par mot de passe sans stocker ce dernier. Dans ce cas, on stocke l'empreinte issue du hachage du mot de passe.

En .NET, toutes les classes qui implémentent le hachage héritent de la classe abstraite **System.Security.Cryptography.HashAlgorithm**.

IV-A - Principales méthodes de cette classe

- **Byte[] ComputeHash(Byte[])** : calcule le hachage pour le tableau de bytes en paramètre ;
- **void Clear()** : libère les ressources de l'objet.

IV-A-1 - Algorithmes de hachage implémentés en .NET

Algorithmes	Classes
MD5	MD5CryptoServiceProvider, MD5Cng.
SHA-1	SHA1Managed, SHA1CryptoServiceProvider, SHA1Cng. Proposée en hash de plusieurs tailles : 224, 256, 384 et 512 bits.
SHA-2	Version 512 bits : SHA512Managed, SHA512CryptoServiceProvider, SHA512Cng.

Exemple 3 :

```
1. static void Main(string[] args)
2. {
3.     string
 texte = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
4.
5.     byte[] textAsByte = Encoding.Default.GetBytes(texte);
6.
```

```
7.      SHA512 sha512 = SHA512Cng.Create();
8.
9.      byte[] hash = sha512.ComputeHash(textAsByte);
10.
11.      Console.WriteLine("Hash = {0}", Convert.ToBase64String(hash));
12. }
```

Résultat de l'exécution :

```
Hash = k5OlEe+eUxsmMaewR/tO2Wm2nJTJxrvXt0fHIWpKTGiyt/swWnBTFc9Chx7HE6F3sniz3G0Cd
smJ4MAzCQf5Xw==
```

V - Conclusion

Cet article vous a permis de voir les classes implémentant la cryptographie en .NET. La cryptographie évolue, de nouveaux algorithmes sont régulièrement créés. Microsoft recommande les algorithmes suivants : **AES** pour la protection des données, **HMACSHA256** pour leur intégrité, **RSA** pour les signatures numériques et l'échange de clés.

VI - Remerciements

Nous remercions la société **Soat** qui nous a autorisés à publier ce tutoriel.

Nous tenons également à remercier **f-leb** pour la relecture orthographique, et **Malick SECK** pour la mise au gabarit.