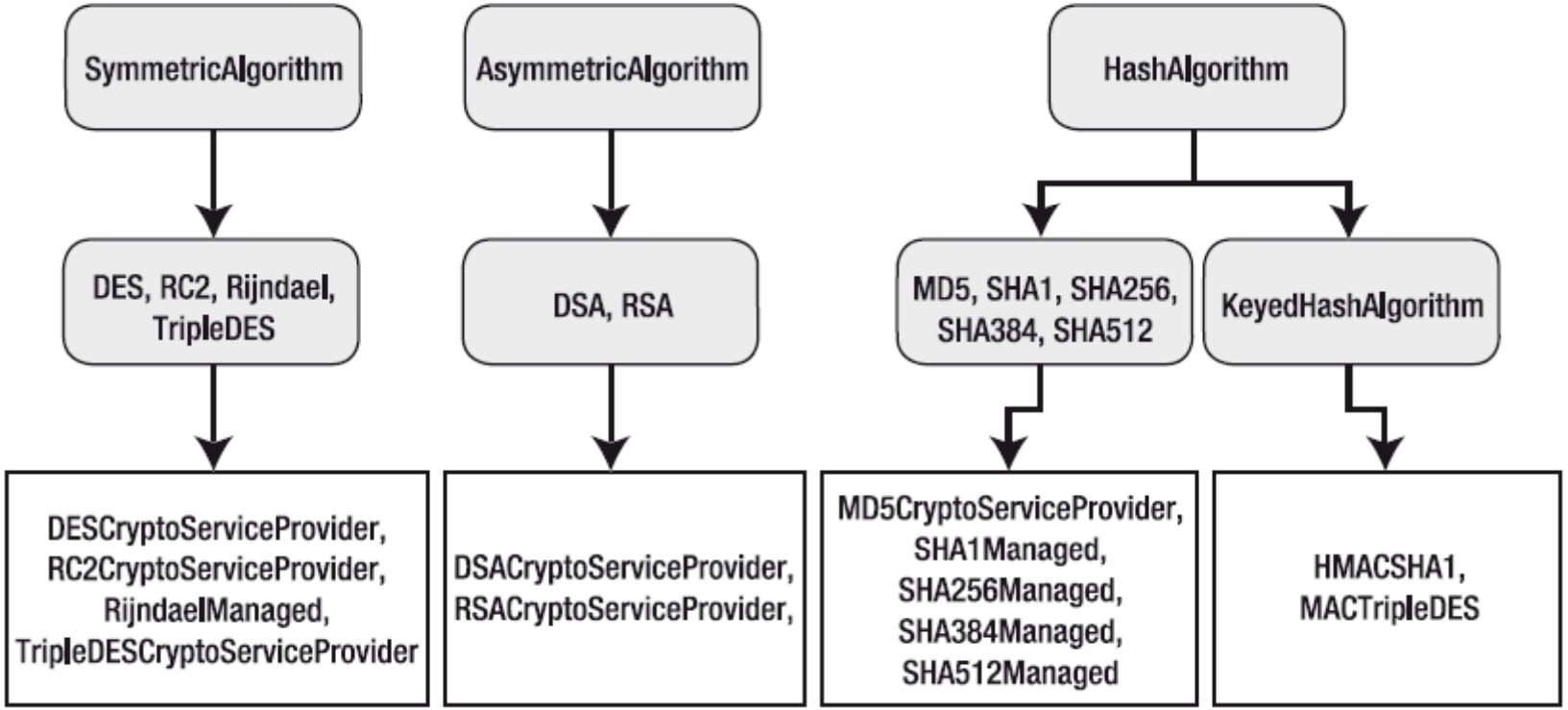
Cryptographie .NET

Algorithmes symétriques
Algorithmes asymétriques
Algorithmes de hachage
Signatures digitales

System.Security.Cryptography



Algorithmes à clés symétriques

Classe	Taille de la clé	Description
RijndaelManaged	De 128 à 256 bits par incréments de 32 bits	Implémentation de l'algorithme AES La classe est complètement Managée.
AesManaged	Similaire à RijndaelManaged, mais limite les blocs à 128 bits. (à partir de la version 3.5)	
RC2	variable	conçu par Ronald Rivest en 1987
DES	56	
TripleDES	156 (112 réellement utilisés).	

- Classe de base: System.Security.Cryptography.SymetricAlgorithm
- Propriétés
  - BlockSize
  - IV: vecteur d'initialisation
  - Key: si aucune valeur n'est assignée à cette propriété lors d'une opération de cryptage, alors une clé est générée aléatoirement. La clé peut aussi générée à partir d'un mot de passe à l'aide de la classe Rfc2898DeriveBytes.
  - KeySize: par défaut le framework .Net utilise la valeur maximale autorisée.
  - Mode:une valeur de type CipherMode (une énumération), par défaut: CBC.
- Méthodes:
  - CreateDecryptor
  - CreateEncryptor
  - GenerateIV
  - GenerateKey

Cryptage / Décryptage d'un message

- Etapes
  1. Créer les flux de données vers les fichier ou les données stockés en mémoire (en lecture et écriture)
  2. Créer un objet de type SymmetricAlgorithm.
  3. Renseigner la clé et le vecteur d'initialisation de l'algorithme
  4. Créer un objet de type ICryptoTransform par appel de la méthode SymmetricAlgorithm.CreateEncryptor() pour crypter ou bien SymmetricAlgorithm.CreateDecryptor() pour décrypter
  5. Créer un flux de cryptage (décryptage) de type CryptoStream
  6. Lire à partir du flux de cryptage pour décrypter, et écrire dans le flux pour crypter.

Cryptage d'un fichier

```
string inChemin = "d:/message.txt";
 string outChemin = "d:/cryptogramme.txt";
 string mdp = "123456@abcdef";

 //Créer la clé à partir du mot de passe
byte[] salt  = Encoding.ASCII.GetBytes("Un peu de sel");
Rfc2898DeriveBytes cle = new Rfc2898DeriveBytes(mdp, salt);

//Créer l'algorithme
RijndaelManaged algoAES= new RijndaelManaged();

//Initialiser la clé et le vecteur d'initialisation
algoAES.Key = cle.GetBytes(algoAES.KeySize/8);
 algoAES.IV = cle.GetBytes(algoAES.BlockSize/8);

 // Lire le fichier à crypter et stocker le message dans le tableau
      données
//Modes d'ouverture de fichier: Open, Create, CreateNew, Append,
      Truncate,OpenOrCreate
//Modes d'accès: Read, Write, ReadWrite
FileStream inF = new FileStream(inChemin, FileMode.Open,
      FileAccess.Read);
    byte[] donnees = new byte[inF.Length];
    inF.Read(donnees, 0, (int)inF.Length);


    // Créer la fonction de cryptage
    ICryptoTransform crypter = algoAES.CreateEncryptor();
    //Créer le flux de sortie vers le fichier crypté
    FileStream outF = new FileStream(outChemin,
FileMode.OpenOrCreate, FileAccess.Write);
    //Créer l'objet CryptoSystem
    CryptoStream cs = new CryptoStream(outF, crypter,
CryptoStreamMode.Write);

    // Ecrire le contenu du message dans le flux de cryptage cs.
    cs.Write(donnees, 0, donnees.Length);

    // Fermer les flux
    cs.Close();
    inF.Close();
    outF.Close();
```

Décryptage d'un fichier

```
string inChemin = "d:/cryptogramme.txt";
string outChemin = "d:/decrypte.txt";
tring mdp = "123456@abcdef";
byte[] salt = Encoding.ASCII.GetBytes("Un peu de sel");
Rfc2898DeriveBytes cle = new Rfc2898DeriveBytes(mdp,
    salt);
RijndaelManaged algoAES = new RijndaelManaged();
algoAES.Key = cle.GetBytes(algoAES.KeySize / 8);
algoAES.IV = cle.GetBytes(algoAES.BlockSize / 8);

// Lecture du fichier crypté
FileStream inF = new FileStream(inChemin, FileMode.Open,
    FileAccess.Read);

//Créer la focnction de décryptage
ICryptoTransform decrypter = algoAES.CreateDecryptor();

//Créer le flux de décryptage
CryptoStream cs = new CryptoStream(inF, decrypter,
    CryptoStreamMode.Read);
```

```
    //Créer le tableau données
    byte[] donnees = new byte[inF.Length];

    // Stocker dans le tableau données le message
décrypté
    cs.Read(donnees, 0, (int)inF.Length);

    // Créer le flux en sortie vers le fichier qui doit
contenir le message décrypté
    FileStream outF = new FileStream(outChemin,
FileMode.OpenOrCreate, FileAccess.Write);
    outF.Write(donnees, 0, donnees.Length);

    // Fermeture des flux
    cs.Close();
    inF.Close();
    outF.Close();
```

Cryptage d'un message

```
RijndaelManaged algoAES = new
    RijndaelManaged();
algoAES.GenerateIV();
algoAES.GenerateKey();
cle = algoAES.Key;
IV = algoAES.IV;
//Fonction de cryptage
ICryptoTransform crypter =
    algoAES.CreateEncryptor();

//Flux du message crypté
 MemoryStream mscrypte = new
    MemoryStream();
CryptoStream cs = new
    CryptoStream(mscrypte, crypter,
CryptoStreamMode.Write);

    //Création du flux du message
    StreamWriter sw = new
StreamWriter(cs);
    //Ecriture dans le flux de cryptage.
    sw.Write(message);
    sw.Close();
    cs.Close();
    byte[] cryptogramme =
mscrypte.ToArray();
    mscrypte.Close();

    return cryptogramme;
```

Décryptage d'un message

```
RijndaelManaged algoAES = new RijndaelManaged();
        algoAES.Key = cle;
        algoAES.IV = IV;
        //Fonction de décryptage
        ICryptoTransform decrypter = algoAES.CreateDecryptor();
        MemoryStream ms = new MemoryStream(cryptogramme);
        CryptoStream cs = new CryptoStream(ms, decrypter,
    CryptoStreamMode.Read);
        StreamReader sr = new StreamReader(cs);
        string decrypte = sr.ReadToEnd();
        sr.Close();
        cs.Close();
        ms.Close();
        return decrypte;
```

Algorithmes asymétriques

- Classes
  - Classe de base:
    System.Security.Cryptography.AsymmetricAlgorithm
  - ***RSACryptoServiceProvider***
  - ***DSACryptoServiceProvider***

RSA: Exemple

```
string message = "Cryptez moi!";
        RSACryptoServiceProvider algoRSA = new
    RSACryptoServiceProvider();
        byte[] donnees = Encoding.Unicode.GetBytes(message);
        byte[] cryptogramme = algoRSA.Encrypt(donnees, false);
        byte[] messageDecrypte = algoRSA.Decrypt(cryptogramme, false);

    //Console.WriteLine(Encoding.Unicode.GetString(messageDecrypte));
        textBox1.Text = Encoding.Unicode.GetString(messageDecrypte);
        textBox2.Text = Encoding.Unicode.GetString(cryptogramme);
```

- Propriétés
  - PersistKeyInCsp: (true ou false) définit si la clé doit être enregistrée dans le CSP (Crypto Service Provider) ou non
  - UseMachineKeyStore: (true ou false) définit si la clé doit être enregistrée dans le store de la machine ou bien celui du profile de l'utilisateur.

- Méthodes:
  - Encrypt
  - Decrypt
  - ExportParameters (bool): exporte la structure *RSAParameters* si l'argument vaut true alors les deux clés sont exportées.
    - *Structure* ***RSAParameters***
      - ***D: clé privée***
      - ***Exponent: e***
      - ***Modulus: n***
    - *Exemple:* RSAParameters clePublique = rsa.ExportParameters(false);
  - *importParameters*
  - FromXmlString: importer les clés au format XML
  - ToXmlString(true): exporte les clés au format xml

<RSAKeyValue>
<Modulus>4GfA4JVTnnfLvklM2/jezbokcC8BGEpo2GIIRARTSlN83k5ky2VnpUS2SmPBh7YLCK
jCHXCBSvA6VFY2p6OVw1bXh8g48rh6o5nbqNdW9499xAH7ou553tekr4nZZsZbMkOrxZ5c
gUkAXByAkWrMHKkMKsUm2VgL8WeQ5qrrePk=</Modulus>
<Exponent>AQAB</Exponent>

<P>9U4iFwe80VYhoyE2YsC1Nzt/9spBD7zAaWw221gMxsCjHp9CO8xBlbQ+HCQvfXBNox
bj2EnkLuYOhwLQ6sPTlw==</P>

<Q>6jBa3OZum4CB24jMZpF+KcXbIUwR0ZOZPf+qUfZEWCgpsruE5Au5+MqN+Js/z/Z/
7hzlcT4PSuoHSGEigDBp7w==</Q>

<DP>HYskdeubpQaflHsKhsg4ywcieUGQpmmQLuI2lSI8n69Rtf1uR69o8rO7iz4cbhoZb5vxIvp
7Pd69Pytqp+ufzw==</DP>

<DQ>mUBxhZNKGHq8//QChrB5Vk7C+oQ0OOiU5KyVQbOdv+7wcqUh7rX5ymSLCYB
W/vl5eFHyl+ubyFNj1qqyEB8egw==</DQ>

<InverseQ>Jx3/k9SmF+wFl3U0/bgehk3t4yONlELSpdadN07guZjU3mrfQL99nub/Q4N5VD
x5YzrplSrBmQCYyAQlEDwZNA==</InverseQ>

<D>w1kuq5LyrwHEKIsw0FEuy9KeA9a3Ukj8SlnUrjlBtHNHQVLnJbTJ7DILqmh6wcQ0iL439
V7J/s1vSZmfjIG4TSmzJt01l5Z+VuVVbCPJ/GonPUFXsT80BNC8stAW96vDxxIj9dd3T97TC
OnnM+Z5Gcg9QmdWJO84Fmq7vqnRdD0=</D>
</RSAKeyValue>

- Stocker les clés dans un conteneur de clé
  - Créer un objet CspParameters

```
CspParameters mCsp = new CspParameters();
mCsp.KeyContainerName = "Store0";
```

  - Créer une nouvelle instance d'une classe qui dérive de la classe AsymmetricAlgorithm

```
RSACryptoServiceProvider rsa = new
RSACryptoServiceProvider(mCsp);
```

  - rsa.PersistKeyInCsp = true; // par défaut la clé sera enregistrée dans le conteneur
- supprimer les clés d'un conteneur
  - Créer un objet CspParameters

```
CspParameters mCsp = new CspParameters();
mCsp.KeyContainerName = "Store0"; // Nom du conteneur à
supprimer
```

  - Créer une nouvelle instance d'une classe qui dérive de la classe AsymmetricAlgorithm

Algorithmes de hachage

Algorithmes sans clé (classe de base System.Security.Cryptography.HashAlgorithm)		
Classe abstraite	Implémentation	Description
MD5	*MD5CryptoServiceProvider*	Message Digest 5, taille du code hache : 128 bits
RIPEMD160	*RIPEMD160Managed*	Message Digest 160, taille du code hache : 160 bits
SHA1	*SHA1CryptoServiceProvider*	Secure Hash Algorithm 1, taille du code hache: 160 bits
SHA256	*SHA256Managed*	Secure Hash Algorithm 256, taille du code hache: 256 bits
SHA384	*SHA384Managed*	Secure Hash Algorithm 348, taille du code hache: 384 bits
SHA512	*SHA512Managed*	Secure Hash Algorithm 512, taille du code hache: 512 bits

Algorithmes avec clé (classe de base: *System.Security.Cryptography.KeyedHashAlgorithm*)		
	HMACSHA1	Hash-based Message Authentication Code SHA1 accepte des clés d'une taille quelconque, taille du code hache: 160
	MACTripleDES	Message Authentication Code TripleDES, tailles de la clé: 8,16,24 octets taille du code hache: 8 octets.

Calcul d'un code de hachage (algorithme sans clé)

- Créer l'objet algorithme

```
MD5 algoHache = new MD5CryptoServiceProvider();
```

- Stocker les données dans un tableau d'octets

```
FileStream fs = new FileStream("fichier.txt", FileMode.Open,
   FileAccess.Read);
BinaryReader reader = new BinaryReader(fs);
byte[] donnees = reader.ReadBytes((int)fs.Length);
```

- Appeler la méthode ComputeHash

```
algoHache.ComputeHash(donnees);
```

- Récupérer le code de hachage (propriété Hash)

```
Console.WriteLine(Convert.ToBase64String(algoHache.Hash));
```

Calcul d'un code de hachage (algorithme avec clé)

- Créer la clé secrète qui doit utilisée pour le calcul et la vérification du code de hachage .

```
byte[] salt = Encoding.ASCII.GetBytes("Un peu de sel");
Rfc2898DeriveBytes pass =new Rfc2898DeriveBytes("mdp", salt);
byte[] cleSecrete = pass.GetBytes(16);
```

- Créer l'objet algorithme

```
HMACSHA1 algoHache = new HMACSHA1(cleSecrete);
```

Remarque: on peut aussi appeler le constructeur par défaut, et dans ce cas là une clé sera créée automatiquement.

- Stocker les données dans un tableau d'octets

```
FileStream fs = new FileStream("fichier.txt", FileMode.Open, FileAccess.Read);
BinaryReader reader = new BinaryReader(fs);
byte[] donnees = reader.ReadBytes((int)fs.Length);
```

- Appeler la méthode ComputeHash

```
algoHache.ComputeHash(donnees);
```

- Récupérer le code de hachage (propriété Hash)

```
Console.WriteLine(Convert.ToBase64String(algoHache.Hash));
```

Signature digitale des fichiers

- Les classes *DSACryptoServiceProvider et RSACryptoServiceProvider sont utilisées pour générer et vérifier les signatures digitales*
- *Méthodes*
  - SignHash: génère une signature digitale à partir d'un code de hachage d'un fichier.
  - SignData: calcule le code de hachage d'un fichier et génère la signature digitale par la suite.
  - VerifyHash
  - VerifyData

Application

- Générer une signature digitale
  - Créer l'objet algorithme

    ```
    DSACryptoServiceProvider algo= new DSACryptoServiceProvider();
    ```
  - Stocker les données à signer dans un tableau binaire.

    ```
    FileStream fs = new FileStream("nom_fichier", FileMode.Open, FileAccess.Read);
    BinaryReader br = new BinaryReader(fs);
    byte[] donnees = br.ReadBytes((int)fs.Length);
    ```
  - Appeler la méthode SignData

    ```
    byte[] signature = algo.SignData(donnees);
    ```
  - Exporter la clé publique.

    ```
    string clePublique = algo.ToXmlString(false);
    ```
- Vérifier la signature digitale
  - Créer l'objet algorithme
  - Importer la signature et la clé publique

    ```
    algo.FromXmlString(clePublique);
    ```
  - Stocker les données à vérifier dans un tableau d'octets
  - Appeler la méthode VerifyData

    ```
    if (algo.VerifyData(donnees, signature))
    ```

- Remarque
  - L'algorithme *RSACryptoServiceProvider peut être aussi utilisé pour signer des documents, mais dans ce cas là il faut calculer le code de hachage lors de la signature et aussi lors de la vérification.*
    - byte[] signature = algo.SignData(donnees, new SHA1CryptoServiceProvider());
    - if (algo.VerifyData(donnees, new SHA1CryptoServiceProvider(), signature)) ;

Exercices

1. Ecrire une application console nommée crypt qui crypte des fichiers à l'aide de l'algorithme DES .
   l'application accepte les paramètres suivants: "nom fichier" et "mot de passe" (le mot de passe est utilisé pour générer la clé et le vecteur d'initialisation).
   Exemple d'utilisation: c:\crypt fichier.dat mdp
2. Ecrire une application console nommée decrypt qui décrypte des fichiers cryptés par l'application de l'exercice précédent. exemple d'utilisation: c:\decrypt fichier_crypte mdp.
3. Ecrire une nouvelle version des programmes crypt et decrypt qui permet à l'utilisateur de choisir le protocole de cryptage à utiliser.