.NET Framework 4.6 and C# 7.0

Lesson 17 : Encrypting and Decrypting Data

Capgemini

Module Overview

- Implementing Symmetric Encryption
- Implementing Asymmetric Encryption

What is Encryption & Decryption?

- *Encryption* is the process of transforming plain data in a way that makes it harder for an unauthorized person to make sense of it. The encrypted data is called *ciphertext*.
- *Decryption* is the reverse process, meaning that having the ciphertext, you must apply a transformation to it to get back the original information.
- *Cryptography* is the practice and study of encryption and decryption techniques.

Algorithm Implementation – Different Ways

- Microsoft implemented some of the existing algorithms in .NET, which are implemented in three ways:
- **1. Managed classes (in .NET)**
  - The class name for those is the algorithm name suffixed with Managed,
  - for instance RijndaelManaged is the managed class that implements the Rijndael algorithm.
- **2. Wrapper classes around the native Cryptography API (CAPI) implementation**
  - The class name for those is the algorithm name suffixed with CryptoServiceProvider,
  - for instance DESCryptoServiceProvider is the wrapper class that implements the Data Encryption Standard (DES) algorithm.

The managed implementations are somewhat slower than the other implementations and are not certified by the Federal Information Processing Standards (FIPS).

The CAPI implementations are suitable for older systems, but they are no longer being developed.

CNG algorithms require a Windows Vista or newer operating system for CNG API

Algorithm Implementation – Different Ways

- **3. Wrapper classes around the native Cryptography Next Generation (CNG) API implementation**
  - The class name for those is the algorithm name suffixed with CNG,
  - for instance ECDiffieHellmanCng is the wrapper class that implements the Elliptic Curve Diffie-Hellman (ECDH) algorithm..

- All cryptography classes are defined in the **System.Security.Cryptography** namespace and are part of the core .NET library.

Symmetric Encryption

- Symmetric encryption is also known as shared secret encryption, and that is because the encryption of the data is done with an encryption key, a byte array, and the same key is used to decrypt the data.

The symmetric algorithms rely on the fact that only an authorized person has access to the encryption key. The main drawback of the symmetric encryption is that if the encryption key becomes compromised, the data will not be secured

The Microsoft® .NET Framework includes managed implementation of the following encryption algorithms:

- Data Encryption Standard (DES)
- AES
- Rivest Cipher 2 (RC2)
- Rijndael
- TripleDES

The **System.Security.Cryptography** namespace includes other cryptographic

classes that you can use to derive secret keys and IVs and write cryptographically transformed data.

Workflow of Encrypting

1. Create a symmetric algorithm object by calling the Create method of the SymmetricAlgorithm class setting the optional string parameter to the name of the wanted algorithm.
2. If you want you can set a key and an IV, but this is not necessary because they are generated by default.
3. Create an encryptor object by calling the CreateEncryptor method. Again, you can choose to send the key and the IV as parameters to this method or use the default, generated one.
4. Call the TransformFinalBlock method on the encryptor, which takes as input a byte array, representing the plain data, the offset where to start the encryption from, and the length of the data to encrypt. It returns the encrypted data back.

Code Sample- Encryption

The code should look like this:

```
byte[] EncryptData(byte[] plainData, byte[] IV, byte[] key) {

    SymmetricAlgorithm cryptoAlgorythm = SymmetricAlgorithm.Create();
    ICryptoTransform encryptor = cryptoAlgorythm.CreateEncryptor(key, IV);
    byte[] cipherData = encryptor.TransformFinalBlock(plainData, 0,
                                                      plainData.Length);

    return cipherData;
}
```

The workflow of decrypting chipper text to get back the plain text is

1. Create a symmetric algorithm object by calling the Create method of the SymmetricAlgorithm class setting the optional string parameter to the name of the same algorithm used for encryption.
2. If you want you can set a key and an IV, but this is not necessary now because you can set them on the next step.
3. Create a decryptor object by calling CreateDecryptor method. You must now set the key and the IV by sending them as parameters to this method, if you didn't do it in the previous step. The key and the IV must be the same as the ones used for encryption.
4. Call the TransformFinalBlock method on the decryptor, which takes as input a byte array, which is the chipper data, the offset where to start the decryption from, and the length of the data to decrypt, and it returns the plain data back.

Code Sample - Decryption

The code should look like this:

```
byte[] DecryptData(byte[] cipherData, byte[] IV, byte[] key) {

    SymmetricAlgorithm cryptoAlgorythm = SymmetricAlgorithm.Create();
    ICryptoTransform decryptor = cryptoAlgorythm.CreateDecryptor(key, IV);
    byte[] plainData = decryptor.TransformFinalBlock(cipherData, 0,
                                                     cipherData.Length);

    return plainData;
}
```

Hashing

- *Hashing* is the process of mapping binary data of a variable length to a fixed size binary data.

ALGORITHM SHORT NAME	DESCRIPTION
SHA1	Implementation of SHA algorithm with a resulting hash size of 160 bits
SHA256	Implementation of SHA algorithm with a resulting hash size of 256 bits
SHA512	Implementation of SHA algorithm with a resulting hash size of 512 bits
SHA384	Implementation of SHA algorithm with a resulting hash size of 384 bits
MD5	Implementation of MD5 hash algorithm
RIPEMD160	Implementation of RIPEMD hash algorithm

A hash is a numerical representation of a piece of data and can be thought of as a digital fingerprint.

Hashing Steps

1. Create a hashing algorithm object.
2. Set the hashing key if the algorithm used is a keyed one.
3. Call the ComputeHash method.
4. Save the hash of the data.

The code should look something like this:

```
string ComputeHash(string input)
{
    HashAlgorithm sha = SHA256.Create();
    byte[] hashData = sha.ComputeHash(Encoding.Default.GetBytes(input));
    return Convert.ToBase64String(hashData);
}
```

Steps for Verifying Hash

1. Create a hashing algorithm object using the same algorithm you used for hashing the data.
2. If a hashing keyed algorithm was used, set the key to the same value used for hashing.
3. Extract the original hash of the data.
4. Call the ComputeHash method.
5. Compare the extracted hash with the computed one. If they are the same, it means that the data wasn't changed.

```
bool VerifyHash(string input, string hashValue) {

    HashAlgorithm sha = SHA256.Create();
    byte[] hashData = sha.ComputeHash(Encoding.Default.GetBytes(input));
    return Convert.ToBase64String(hashData) == hashValue;
}
```

Steps for Verifying Hash

```
bool VerifyHash(string input, string hashValue) {

    HashAlgorithm sha = SHA256.Create();
    byte[] hashData = sha.ComputeHash(Encoding.Default.GetBytes(input));
    return Convert.ToBase64String(hashData) == hashValue;
}
```

Asymmetric Encryption

- The main reason to use asymmetric encryption is to avoid sharing the encryption key, which is considered a vulnerability.
- Asymmetric encryption uses two mathematically related keys that complement each other, such as whatever is encrypted with one key can be decrypted only with the other key.
- One key is made public, and is known as the *public key*, by the receiving party, so whoever wants to transmit secured data can encrypt the data.

The main disadvantage of the asymmetric encryption is that it is slower than the symmetric encryption, but the biggest advantage is that there is no need to have a shared secret for the algorithm to work.

.

Symmetric algorithms implemented in .NET Framework

ALGORITHM SHORT NAME	DESCRIPTION
`DSA`	Digital Signature algorithm. Used to create digital signatures that help protect the integrity of data. There is one class implementing this algorithm: `DSACryptoServiceProvider`. Use DSA only for compatibility with legacy applications and data.
`ECDiffieHellman`	Elliptic Curve Diffie-Hellman algorithm implemented by `ECDiffieHellmanCng`.
`ECDsa`	Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm implemented by `ECDsaCng`.
`RSA`	RSA algorithm implemented by `RSACryptoServiceProvider`.

Workflow of encrypting data using asymmetric algorithm

- In .NET all classes that implement an asymmetric algorithm inherit from **System.Security.Cryptography.AsymmetricAlgorithm.**
- The workflow of encrypting data using asymmetric encryption follows:
  1. Obtain the public key of the receiver.
  2. Create a new asymmetric encryption object.
  3. Set the public key.
  4. Encrypt the data.
  5. Send the data to the receiver.

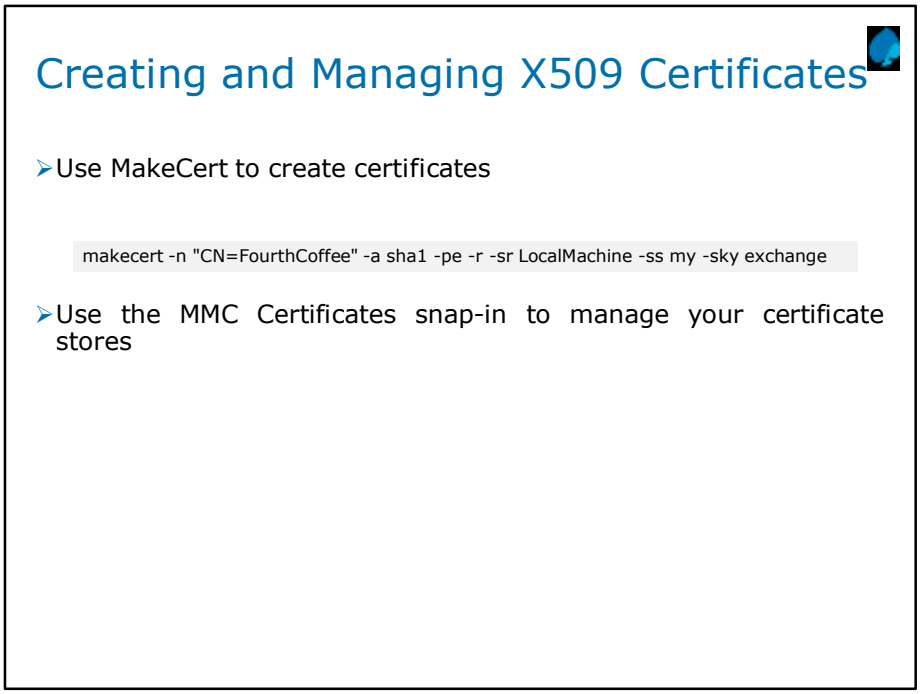
Workflow of decrypting data using asymmetric algorithm

1. Get the data from the sender.
2. Create a new asymmetric encryption object.
3. Set the private key.
4. Decrypt the data.

- If the data were changed or it not encrypted using the corresponding public key, a CryptographicException will be thrown

Code Sample - Asymmetric Encryption

```
var rawBytes = Encoding.Default.GetBytes("hello world..");
var decryptedText = string.Empty;
using (var rsaProvider = new RSACryptoServiceProvider())
{
   var useOaepPadding = true;
   var encryptedBytes =  rsaProvider.Encrypt(rawBytes, useOaepPadding);
   var decryptedBytes = rsaProvider.Decrypt(encryptedBytes, useOaepPadding);
    decryptedText = Encoding.Default.GetString(decryptedBytes);
}
// decryptedText == hello world..
```



- You can use MakeCert, which is a command-line certificate creation tool, to create your own X509 certificates for development and testing.

Choosing when to use which?

- Microsoft recommends the following algorithms to be used in the following situations:
  - For data privacy, use Aes.
  - For data integrity, use HMACSHA256 or HMACSHA512.
  - For digital signatures, use RSA or ECDsa.
  - For key exchange, use RSA or ECDiffieHellman.
  - For random number generation, use RNGCryptoServiceProvider.
  - For generating a key from a password, use Rfc2898DeriveBytes.

Demo

- Walkthrough: Creating a Cryptographic Application.

Summary

- Choosing an encryption algorithm
  - If you need to encrypt data that is used locally, or you have a secure way to distribute the encryption key, use the symmetric encryption
  - If you don't have a secure way to send the encryption key data between parties, then asymmetric encryption is recommended
  - If you need only to ensure integrity of the data, use a hashing algorithm
  - If you need to ensure both integrity and authenticity, choose a MAC algorithm

Summary

- Symmetric encryption
  - Based on a common key called shared secret
  - It needs an initialization vector (IV) that doesn't need to be secret but is used to encrypt the first block of data
  - You use it by instantiating a symmetric algorithm object and then calling CreateEncryptor or CreateDecryptor
  - The encryptor/decryptor is then used with either by calling directly the TransformFinalBlock method or by sending it to a CryptoStream

Summary

- Asymmetric encryption
  - It is based on a pair of complementary keys
  - Encrypted data with one key can be decrypted only with the other key
  - One key is kept secret and is called a *private key*; the other one is made available to anyone that wants to encrypt data, or verify encrypted data, and it is called a *public key*
- Key management
  - Symmetric keys can be exchanged using asymmetric algorithms
  - Asymmetric private keys can be secured either by using certificates or by using Crypto Service Providers containers

Summary

- Hashing
  - Mapping binary data of a variable length to a fixed size binary data, called *hash*
  - When you need to make sure that data is not modified while transferred, you can calculate the cryptographic hash and send it together with the data to be verified by the receiving party
  - The two commonly used algorithms are SHA256 and SHA512 with resulting hashes of 256 and 512 bits, respectively (32 and 64 bytes)