Criptografia

# Definicion

La criptografía nos permite proteger los datos de forma tal, que la visualización o modificación de los mismos solo sea posible para aquellos que conozcan la forma en que han sido encriptados. Esto es aplicable no solo a la comunicación entre dos puntos, sino también al contenido de nuestros ficheros.Además de proteger la información,también podemos utilizar la criptografía para garantizar que los datos no han sido alterados y que provienen de una fuente fiable.

Para realizar las tareas criptográficas, .NET Framework nos ofrece una serie de clases en las que se implementan algoritmos criptográficos estandarizados; en este artículo veremos algunos de esos algoritmos y cómo usarlos desde C#.

**Primitivas Criptografías.-**

Existen diferentes formas de encriptación, conocidas como primitivas criptográficas. A continuación enumeramos esas primitivas y el uso que podemos darle.

• **Cifrado de clave secreta (criptografía simétrica)** Realiza la transformación de los datos, impidiendo que terceros los lean. Este tipo de cifrado utiliza una clave secreta compartida para cifrar y descifrar los datos.

• **Cifrado de clave pública (criptografía asimétrica)** Realiza la transformación de los datos, impidiendo que terceros los lean. Este tipo de cifrado utiliza un par de claves pública y privada para cifrar y descifrar los datos.

• **Firmas criptográficas** Ayudan a comprobar que los datos se originan en una parte específica mediante la creación de una firma digital única para esa parte. En este proceso también se usan funciones hash.

• **Valores hash** criptográficos Asigna datos de cualquier longitud a una secuencia de bytes de longitud fija. Los valores hash son únicos estadísticamente; el valor hash de una secuencia de bytes distinta no será el mismo.

# Cifrado y descifrar datos (dsa)

Podemos realizar el cifrado de datos de dos formas distintas, según sea grande o pequeño el tamaño de los mismos. En el primer caso, podemos usar el cifrado simétrico, en el cual intervienen secuencias (streams), en particular una de tipo CryptoStream, que será la que pasemos a las clases que utilizan este tipo de cifrado. La segunda forma de cifrado normalmente lo aplicaremos a cantidades pequeñas de datos, los cuales estarán almacenados habitualmente en un array de bytes.

Pero para poder realizar el cifrado (y posteriormente el descifrado) necesitamos crear las claves para cifrar y escifrar esos datos. En el caso de los algoritmos simétricos, además de una clave también hay que generar un vector de inicialización (IV - Initialization Vector); ambos hay que mantenerlos en secreto y solo deben conocerlos las dos partes interesadas. Las clases utilizadas en este tipo de algoritmos incluyen métodos para generar esas claves, en particular los métodos GenerateKey() y GenerateIV(), aunque también podemos generarlas de forma manual, por ejemplo basándonos en una cadena; en cualquier caso, al instanciar la clase, se generan tanto la clave como el IV de forma automática. La formas de generar las claves para usar con la clase DESCryptoServiceProvider. Si usamos una cadena a partir de la que generar tanto la clave como el vector de inicialización, debemos asegurarnos de que tiene los bytes necesarios; ese valor lo obtenemos por medio de la propiedad KeySize, que devuelve el número de bits necesarios (1 byte = 8 bits).

Por otra parte, los algoritmos asimétricos deben crear una clave pública y otra privada; la clave pública es la que se usará para cifrar el contenido y la privada la usaremos para descifrarlo. Si no indicamos lo contrario en el constructor de las clases que utilizan este tipo de algoritmos, se generarán automáticamente claves con una longitud de 1024 bits, (128 bytes); el rango puede ser de 512 a 1024 en incrementos de 64 bits. Por supuesto, solo debemos exponer la clave pública, que será la usada para cifrar el mensaje, mientras que la clave privada la usaremos para descifrarlo. Estas clases implementan métodos para exportar e importar las claves públicas y/o privadas. Por ejemplo, si queremos generar una cadena en formato XML con la clave pública podemos usar el método ToXmlString() pasándole un valor falso es permitirnos generar un valor hash, por ejemplo del tipo SHA1, con el contenido del texto a cifrar. Mediante ese valor hash podemos comprobar si el mensaje (o texto) ha sido modificado. De esta forma, el mensaje realmente no se cifra, sino solo los valores de comprobación, ya que una de las funcionalidades de este tipo de algoritmos es obtener firmas digitales de mensajes, de forma que podamos comprobar la autenticidad del mismo. Si quisiéramos cifrar el mensaje, podremos usar cualquiera de las clases de cifrado simétrico o bien usar los como parámetro; si queremos exportar las dos claves, tendremos que usar un valor verdadero como parámetro de esa llamada. En metodo ToXmlString() se utiliza para guardar las claves usadas al crear la instancia de la clase DSACryptoServiceProvider.

# Cifrado y descifrar asimetrico (rsa)

Podemos usar la clase RSACrypto-ServiceProvider para cifrar y descifrar textos de forma fácil ya que, a diferencia de la clase DSACryptoServiceProvider, expone métodos para realizar esas tareas. Para cifrar un texto (en realidad un array de bytes), usaremos el método Encrypt(), que devolverá un array con el texto cifrado. Para descifrar un texto previamente cifrado, usaremos el método Decrypt(). En ambos casos debemos usar las claves públicas y privadas. Si lo que queremos es cifrar los datos, solo necesitamos la clave pública; pero para descifrar esos datos, necesitaremos también la clave privada.

# Comprobar Firma Digital (dsa)

Una de las características de las clases que utilizan cifrado asimétrico métodos que nos permiten tanto el cifrado como el descifrado. A la hora de usar la firma digital, el que la genera debe disponer de las dos claves, (pública y privada), pero el que comprueba si es correcta solo necesita la clave pública. Podemos ver cómo generar el valor hash cifrado a partir de una cadena de texto; también usamos ese valor hash para comprobar que el texto coincide con el valor hash usado para verificarlo. Para generar el valor hash SHA1 utilizamos un objeto del tipo SHA1CryptoServiceProvider, para asegurarnos de que el valor hash generado tiene la longitud adecuada (20 bytes).

# Generar Numero Aleatorio Criptografico (rng)

Las clases de .NET Framework utilizan generadores de números aleatorios para generar las claves criptográficas. Nosotros podemos generar también esos números aleatorios por medio de la clase RNGCryptoServiceProvider, que está basada en la clase abstracta RandomNumberGenerator. La forma de generar esos números aleatorios es por medio de un array de tipo byte en el que indicamos cuántas cifras debe generar, ya que en cada elemento del array se incluirá una cifra. El método usado para generar los números aleatorios es GetBytes(), el cual incluirá todos los valores, incluso el cero.