Descubra como utilizar System.Security.Cryptography no ambiente Windows

System.Security.Cryptography é uma biblioteca do .NET que fornece serviços de criptografia, incluindo algoritmos de hash, cifragem e geração de chaves. Esta biblioteca é amplamente utilizada para garantir a segurança de dados em aplicações .NET, e é totalmente aplicável ao ambiente Windows.

Introdução

A criptografia é essencial para proteger dados sensíveis contra acesso não autorizado. No ambiente Windows, a biblioteca System.Security.Cryptography do .NET Framework oferece uma maneira robusta e simples de implementar criptografia em suas aplicações. Este artigo irá guiá-lo através de exemplos práticos de como utilizar esta biblioteca para realizar operações comuns de criptografia.

Exemplo 1: Criptografia Simétrica com AES

A criptografia simétrica utiliza a mesma chave para cifrar e decifrar dados. O algoritmo AES (Advanced Encryption Standard) é um dos mais populares para este propósito.

Código de Exemplo

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

class Program
{
    static void Main()
    {
        string original = "Texto a ser criptografado";

        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Encoding.UTF8.GetBytes("0123456789abcdef012345678
9abcdef");
            aesAlg.IV = Encoding.UTF8.GetBytes("abcdef9876543210");

            byte[] encrypted = EncryptStringToBytes_Aes(original, aesAlg.K
ey, aesAlg.IV);
            string roundtrip = DecryptStringFromBytes_Aes(encrypted, aesAl
g.Key, aesAlg.IV);

            Console.WriteLine($"Texto Original: {original}");
```

```
            Console.WriteLine($"Texto Criptografado: {Convert.ToBase64Stri
ng(encrypted)}");
            Console.WriteLine($"Texto Decifrado: {roundtrip}");
        }
    }

    static byte[] EncryptStringToBytes_Aes(string plainText, byte[] Key, b
yte[] IV)
    {
        if (plainText == null || plainText.Length <= 0)
            throw new ArgumentNullException("plainText");
        if (Key == null || Key.Length <= 0)
            throw new ArgumentNullException("Key");
        if (IV == null || IV.Length <= 0)
            throw new ArgumentNullException("IV");

        byte[] encrypted;

        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Key;
            aesAlg.IV = IV;

            ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key
, aesAlg.IV);

            using (MemoryStream msEncrypt = new MemoryStream())
            {
                using (CryptoStream csEncrypt = new CryptoStream(msEncrypt
, encryptor, CryptoStreamMode.Write))
                {
                    using (StreamWriter swEncrypt = new StreamWriter(csEnc
rypt))
                    {
                        swEncrypt.Write(plainText);
                    }
                    encrypted = msEncrypt.ToArray();
                }
            }
        }

        return encrypted;
    }

    static string DecryptStringFromBytes_Aes(byte[] cipherText, byte[] Key
, byte[] IV)
    {
        if (cipherText == null || cipherText.Length <= 0)
            throw new ArgumentNullException("cipherText");
        if (Key == null || Key.Length <= 0)
```

```
            throw new ArgumentNullException("Key");
        if (IV == null || IV.Length <= 0)
            throw new ArgumentNullException("IV");

        string plaintext = null;

        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Key;
            aesAlg.IV = IV;

            ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key
, aesAlg.IV);

            using (MemoryStream msDecrypt = new MemoryStream(cipherText))
            {
                using (CryptoStream csDecrypt = new CryptoStream(msDecrypt
, decryptor, CryptoStreamMode.Read))
                {
                    using (StreamReader srDecrypt = new StreamReader(csDec
rypt))
                    {
                        plaintext = srDecrypt.ReadToEnd();
                    }
                }
            }
        }

        return plaintext;
    }
}
```

Exemplo 2: Hashing com SHA-256

Hashing é usado para garantir a integridade dos dados. O algoritmo SHA-256 gera um hash de 256 bits, que é único para cada entrada.

Código de Exemplo

```
using System;
using System.Security.Cryptography;
using System.Text;

class Program
{
    static void Main()
    {
        string data = "Texto para hash";
```

```
        using (SHA256 sha256Hash = SHA256.Create())
        {
            byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(d
ata));
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < bytes.Length; i++)
            {
                builder.Append(bytes[i].ToString("x2"));
            }
            Console.WriteLine($"Hash SHA-256: {builder.ToString()}");
        }
    }
}
```

Conclusão

A biblioteca System.Security.Cryptography do .NET Framework oferece uma maneira poderosa e flexível de implementar criptografia e hashing em suas aplicações Windows. Com os exemplos fornecidos, você pode começar a proteger dados sensíveis de maneira eficaz.