RSA ANAHTAR DAĞITIMI VE RSA İLE DİJİTAL İMZA OLUŞTURMA

İlk defa 1977 yılında Ron Rivest, Adi Shamir ve Leonard Adleman tarafından oluşturulan RSA algoritması geliştiricilerinin soyisimlerinin ilk harfleriyle anılmaktadır. Bu yazımızda RSA'da yer alan anahtar dağıtımı algoritması ve dijital imza oluşturmayı inceleyeceğiz.

Anahtar Sözcükler: Şifreleme Algoritması, RSA, Dijital İmza, Anahtar Dağıtımı, Microsoft .NET, Kriptoloji

RSAOAEPKeyExchangeFormatter sınıfı, RSA kullanarak Optimal Asymmetric Encryption Padding (OAEP) anahtar değişimi verisini oluşturur. AsymmetricKeyExchangeFormatter sınıfından türeyen bu sınıfın şu erişilebilir özellikleri vardır:

- **Parameters**, RSA tarafından anahtar değişimi sırasında kullanılacak parametreleri belirtir.
- **Rng**, anahtar değişimi sırasında kullanılacak Rastgele Sayı Üretici (Random Number Generator) algoritmayı belirler. Dışarıdan değiştirerek kendi özel algoritmanızı da kullanabilirsiniz.

Bu sınıfın şu erişilebilir metotları vardır:

- **CreateKeyExchange**, şifrelenmiş anahtar değişimi verisini oluşturur.
- **Equals**, iki nesnenin birbirine eşit olup olmadığını test eder.
- **GetHashCode**, bellekteki o nesneye özgü bir hash kodu oluşturur.
- **GetType**, bu nesnenin tipini verir.
- **SetKey**, anahtar değişimi sırasında kullanılacak olan şifreleme algoritmasının genel anahtarını ayarlamanıza izin verir.
- **ToString**, şu an ki nesneyi ifade eden bir metin oluşturur.

RSAOAEPKeyExchangeDeformatter sınıfı, RSA kullanarak Optimal Asymmetric Encryption Padding (OAEP) anahtar değişimi verisini deşifreler. AsymmetricKeyExchangeDeformatter sınıfından türeyen bu sınıfın tek bir erişilebilir özelliği vardır:

- **Parameters**, RSA tarafından anahtar değişimi sırasında kullanılan parametreleri verir.

Bu sınıfın şu erişilebilir metotları vardır:

- **DecryptKeyExchange**, şifrelenmiş anahtar değişimi verisinden gizli bilgiyi elde etmek deşifreleme yapar.
- **Equals**, iki nesnenin birbirine eşit olup olmadığını test eder.
- **GetHashCode**, bellekteki o nesneye özgü bir hash kodu oluşturur.
- **GetType**, bu nesnenin tipini verir.
- **SetKey**, anahtar değişimi sırasında kullanılacak olan deşifreleme algoritmasının özel anahtarını ayarlamanıza izin verir.
- **ToString**, şu an ki nesneyi ifade eden bir metin oluşturur.

RSAPKCS1KeyExchangeFormatter sınıfı, RSA kullanarak PKCS#1 anahtar değişimi verisini oluşturur. AsymmetricKeyExchangeFormatter sınıfından türeyen bu sınıfın şu erişilebilir özellikleri vardır:

- **Parameters**, RSA tarafından PKCS #1 anahtar değişimi sırasında kullanılacak parametreleri belirtir.
- **Rng**, anahtar değişimi sırasında kullanılacak Rastgele Sayı Üretici (Random Number Generator) algoritmayı belirler. Dışarıdan değiştirerek kendi özel algoritmanızı da kullanabilirsiniz.

Bu sınıfın şu erişilebilir metotları vardır:

- **CreateKeyExchange**, şifrelenmiş anahtar değişimi verisini oluşturur.
- **Equals**, iki nesnenin birbirine eşit olup olmadığını test eder.
- **GetHashCode**, bellekteki o nesneye özgü bir hash kodu oluşturur.
- **GetType**, bu nesnenin tipini verir.
- **SetKey**, anahtar değişimi sırasında kullanılacak olan şifreleme algoritmasının genel anahtarını ayarlamanıza izin verir.
- **ToString**, şu an ki nesneyi ifade eden bir metin oluşturur.

RSAPKCS1KeyExchangeDeformatter sınıfı, RSA kullanarak PKCS #1 anahtar değişimi verisini deşifreler. AsymmetricKeyExchangeDeformatter sınıfından türeyen bu sınıfın erişilebilir özellikleri şunlardır:

- **Parameters**, RSA tarafından anahtar değişimi sırasında kullanılan parametreleri verir.
- **Rng**, anahtar değişimi sırasında kullanılan Rastgele Sayı Üretici (Random Number Generator) algoritmasını verir.

Bu sınıfın şu erişilebilir metotları vardır:

- **DecryptKeyExchange**, şifrelenmiş anahtar değişimi verisinden gizli bilgiyi elde etmek deşifreleme yapar.
- **Equals**, iki nesnenin birbirine eşit olup olmadığını test eder.
- **GetHashCode**, bellekteki o nesneye özgü bir hash kodu oluşturur.
- **GetType**, bu nesnenin tipini verir.
- **SetKey**, anahtar değişimi sırasında kullanılacak olan deşifreleme algoritmasının özel anahtarını ayarlamanıza izin verir.
- **ToString**, şu an ki nesneyi ifade eden bir metin oluşturur.

RSAPKCS1SignatureFormatter sınıfı, RSA kullanarak PKCS #1 version 1.5 imzasını oluşturur. AsymmetricSignatureFormatter sınıfından türeyen bu sınıfın erişilebilir bir özelliği bulunmamaktadır. Bu sınıfın erişilebilir metotları ise şunlardır:

- **CreateSignature**, imzayı oluşturur. Bir imza oluşturmadan önce SetHashAlgorithm metotunun çağırarak bir algoritma belirtmeniz gerekmektedir.
- **Equals**, iki nesnenin birbirine eşit olup olmadığını test eder.
- **GetHashCode**, bellekteki o nesneye özgü bir hash kodu oluşturur.
- **GetType**, bu nesnenin tipini verir.
- **SetHashAlgorithm**, imzanın oluşturulması sırasında kullanılacak olan algoritmayı değiştirmenize izin verir. "SHA1" gibi algoritmanın ismini vermeniz yeterli olacaktır.
- **SetKey**, anahtar değişimi sırasında kullanılacak olan şifreleme algoritmasının özel

anahtarını ayarlamanıza izin verir.

- **ToString**, şu an ki nesneyi ifade eden bir metin oluşturur.

RSAPKCS1SignatureDeformatter sınıfı, RSA kullanarak PKCS #1 version 1.5 imzasını doğrular. AsymmetricSignatureDeformatter sınıfından türeyen bu sınıfın erişilebilir bir özelliği bulunmamaktadır. Bu sınıfın erişilebilir metotları ise şunlardır:

- **Equals**, iki nesnenin birbirine eşit olup olmadığını test eder.
- **GetHashCode**, bellekteki o nesneye özgü bir hash kodu oluşturur.
- **GetType**, bu nesnenin tipini verir.
- **SetHashAlgorithm**, imzanın oluşturulması sırasında kullanılacak olan algoritmayı değiştirmenize izin verir. "SHA1" gibi algoritmanın ismini vermeniz yeterli olacaktır.
- **SetKey**, anahtar değişimi sırasında kullanılacak olan şifreleme algoritmasının genel anahtarını ayarlamanıza izin verir.
- **ToString**, şu an ki nesneyi ifade eden bir metin oluşturur.
- **VerifySignature**, SetHashAlgorithm ile belirtilmiş algoritmayı kullanarak imzayı doğrular.

RSA kullanarak imza oluşturma ve imzayı doğrulama işlemi ile ilgili basit bir konsol uygulaması geliştirelim:

C#
```
using System;
using System.Security.Cryptography;

namespace TestConsoleApplication
{
  public  class TestUygulaması
  {
    static void Main()
    {
      try
      {
         //RSACryptoServiceProvider nesnesi oluşturalım
        RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();
        //İmzalamak için kullanacağımız bir hash tanımlayalım.
        byte[] Hash = {59,4,248,102,77,97,142,
        201,210,12,224,93,25,41,100,197,213,134,130,135};
        //RSAOPKCS1SignatureFormatter nesnesi oluşturalım,
        //Anahtar bilgisini transfer edecek olan
        //RSACryptoServiceProvider nesnesiyle ilişkilkendirelim.
        RSAPKCS1SignatureFormatter RSAFormatter = new
        RSAPKCS1SignatureFormatter(RSA);
        //İmza oluşturmak için SHA1 algoritmasını
        //kullanacağımızı belirtelim.
        RSAFormatter.SetHashAlgorithm("SHA1");
        //Hash değeri için imzayı oluşturalım.
        byte[] SignedHash = RSAFormatter.CreateSignature(Hash);
        //RSAPKCS1SignatureDeformatter nesnesi oluşturalım,
        //Anahtar bilgisini transfer edecek olan
        //RSACryptoServiceProvider nesnesiyle ilişkilkendirelim.
        RSAPKCS1SignatureDeformatter RSADeformatter = new
        RSAPKCS1SignatureDeformatter(RSA);
        //SHA1 ile imzanın oluşturulduğunu biliyoruz.
        //İmzayı doğrulamak için SHA1 kullanılacağını belirtelim.
        RSADeformatter.SetHashAlgorithm("SHA1");
        //Hash değerini doğrulayalım,
```

```
        if(RSADeformatter.VerifySignature(Hash, SignedHash))
        {
           Console.WriteLine("İmza Doğrulandı.");
        }
        else
        {
           Console.WriteLine("İmza Doğrulanamadı.");
        }
     }
     catch(CryptographicException e)
     {
        Console.WriteLine("Şifreleme Hatası Oluştu: "+e.Message);
     }
     Console.Read();
   }
 }
}
```

VB.NET

```
Imports System
Imports System.Security.Cryptography
Module TestConsoleApplication
    Sub Main()
        Try
            'RSACryptoServiceProvider nesnesi oluşturalım
            Dim RSA As New RSACryptoServiceProvider
            'İmzalamak için kullanacağımız bir hash tanımlayalım.
            Dim Hash As Byte() = {59, 4, 248, 102, 77, 97, 142,
            201,210,12, 224, 93, 25, 41, 100, 197, 213, 134, 130,135}
            'RSAOPKCS1SignatureFormatter nesnesi oluşturalım,
            'Anahtar bilgisini transfer edecek olan
            'RSACryptoServiceProvider nesnesiyle ilişkilkendirelim.
            Dim RSAFormatter As New
            RSAPKCS1SignatureFormatter(RSA)
            'SHA1 algoritmasını kullanacağımızıbelirtelim.
            RSAFormatter.SetHashAlgorithm("SHA1")
            'Hash değeri için imzayı oluşturalım.
            Dim SignedHash As Byte()= RSAFormatter.CreateSignature(Hash)
            'RSAPKCS1SignatureDeformatter nesnesi oluşturalım,
            'Anahtar bilgisini transfer edecek olan
            'RSACryptoServiceProvider nesnesiyle ilişkilkendirelim.
            Dim RSADeformatter As New RSAPKCS1SignatureDeformatter(RSA)
            'SHA1 ile imzanın oluşturulduğunu biliyoruz.
            'İmzayı doğrulamak için SHA1 kullanılacağını belirtelim.
            RSADeformatter.SetHashAlgorithm("SHA1")
            If (RSADeformatter.VerifySignature(Hash, SignedHash)) Then
                Console.WriteLine("İmza Doğrulandı.")
            Else
                Console.WriteLine("İmza Doğrulanamadı.")
            End If
        Catch e As CryptographicException
            Console.WriteLine("Şifreleme Hatası Oluştu: " + e.Message)
        End Try
        Console.Read()
    End Sub

End Module
```

C++.NET

```
#include "stdafx.h"
```

```
#using <mscorlib.dll>

using namespace System;
using namespace System::Security::Cryptography;

int _tmain()
{
      try
     {
         //RSACryptoServiceProvider nesnesi oluşturalım
        RSACryptoServiceProvider __gc* RSA = __gc
        new RSACryptoServiceProvider();
        //İmzalamak için kullanacağımız bir hash tanımlayalım.
        Byte Hash[]= {59,4,248,102,77,97,142,
        201,210,12,224,93,25,41,100,197,213,134,130,135};
        //RSAOPKCS1SignatureFormatter nesnesi oluşturalım,
        //Anahtar bilgisini transfer edecek olan
        //RSACryptoServiceProvider nesnesiyle ilişkilkendirelim.
        RSAPKCS1SignatureFormatter __gc* RSAFormatter = __gc
        new RSAPKCS1SignatureFormatter(RSA);
        //İmza oluşturmak için SHA1 algoritmasını
        //kullanacağımızı belirtelim.
        RSAFormatter->SetHashAlgorithm("SHA1");
        //Hash değeri için imzayı oluşturalım.
        Byte SignedHash[] = RSAFormatter->CreateSignature(Hash);
        //RSAPKCS1SignatureDeformatter nesnesi oluşturalım,
        //Anahtar bilgisini transfer edecek olan
        //RSACryptoServiceProvider nesnesiyle ilişkilkendirelim.
        RSAPKCS1SignatureDeformatter __gc* RSADeformatter = __gc
        new RSAPKCS1SignatureDeformatter(RSA);
        //SHA1 ile imzanın oluşturulduğunu biliyoruz.
        //İmzayı doğrulamak için SHA1 kullanılacağını belirtelim.
        RSADeformatter->SetHashAlgorithm("SHA1");
        //Hash değerini doğrulayalım,
        if(RSADeformatter->VerifySignature(Hash, SignedHash))
        {
            Console::WriteLine("İmza Doğrulandı.");
        }
        else
        {
            Console::WriteLine("İmza Doğrulanamadı.");
        }
     }
     catch(CryptographicException* e)
     {
         Console::WriteLine(String::Concat(S"Şifreleme Hatası Oluştu: "
         ,e->Message));
     }
    Console::Read();
    return 0;
}

J#
package TestConsoleApplication;

import System.*;
import System.Security.Cryptography.*;

public class TestUygulaması
{
    public TestUygulaması()
```

```
    {
    }

    /** @attribute System.STAThread() */
    public static void main(String[] args)
    {
        try
        {
            //RSACryptoServiceProvider nesnesi oluşturalım
            RSACryptoServiceProvider RSA = new
            RSACryptoServiceProvider();
            //İmzalamak için kullanacağımız bir hash tanımlayalım.
            ubyte[] Hash = {59,4,248,102,77,97,142,
            201,210,12,224,93,25,41,100,197,213,134,130,135};
            //RSAOPKCS1SignatureFormatter nesnesi oluşturalım,
            //Anahtar bilgisini transfer edecek olan
            //RSACryptoServiceProvider nesnesiyle ilişkilkendirelim.
            RSAPKCS1SignatureFormatter RSAFormatter = new
            RSAPKCS1SignatureFormatter(RSA);
            //İmza oluşturmak için SHA1 algoritmasını
            //kullanacağımızı belirtelim.
            RSAFormatter.SetHashAlgorithm("SHA1");
            //Hash değeri için imzayı oluşturalım.
            ubyte[] SignedHash = RSAFormatter.CreateSignature(Hash);
            //RSAPKCS1SignatureDeformatter nesnesi oluşturalım,
            //Anahtar bilgisini transfer edecek olan
            //RSACryptoServiceProvider nesnesiyle ilişkilkendirelim.
            RSAPKCS1SignatureDeformatter RSADeformatter = new
            RSAPKCS1SignatureDeformatter(RSA);
            //SHA1 ile imzanın oluşturulduğunu biliyoruz.
            //İmzayı doğrulamak için SHA1 kullanılacağını belirtelim.
            RSADeformatter.SetHashAlgorithm("SHA1");
            //Hash değerini doğrulayalım,
            if(RSADeformatter.VerifySignature(Hash, SignedHash))
            {
                Console.WriteLine("İmza Doğrulandı.");
            }
            else
            {
                Console.WriteLine("İmza Doğrulanamadı.");
            }
        }
        catch(CryptographicException e)
        {
            Console.WriteLine("Şifreleme Hatası Oluştu: "+
            e.get_Message());
        }
        Console.Read();
    }
}
```

RSA anahtarlarının dağıtımı ve RSA kullanarak dijital imza oluşturmayı örnek kodlarıyla incelediğimiz bu makalemizin de sonuna geldik. RSA algoritmasını uygulamalarınızda birebir kullanabilirsiniz. Bir sonraki makalemizde görüşünceye kadar güvende kalın...

Yunus Emre ALPÖZEN

25.03.2005