RSA ŞİFRELEME ALGORİTMASI

İlk defa 1977 yılında Ron Rivest, Adi Shamir ve Leonard Adleman tarafından oluşturulan RSA algoritması geliştiricilerinin soyisimlerinin ilk harfleriyle anılmaktadır. Bu yazımızda RSA algoritmasını ve bu algoritmanın Microsoft .NET teknolojileri ile nasıl gerçeklenebileceğini inceleyeceğiz.

Anahtar Sözcükler: Şifreleme Algoritması, RSA, Microsoft .NET, Kriptoloji

RSA algoritması, Amerika' da 1983 yılında MIT'ten patent almıştır. Bu patent 21 Eylül 2000 de son bulmuştur. Ancak patenti daha önce bir uygulamaya ait olduğu için bir başka ülkede patent alınamaz.

Bir genel anahtarlı şifreleme tekniği olan RSA, çok büyük tamsayıları oluşturma ve bu sayıları işleminin zorluğu üzerine düşünülmüştür. Anahtar oluşturma işlemi için asal sayılar kullanılarak daha güvenli bir yapı oluşturulmuştur. Anahtar oluşturma algoritması şu şekildedir:

- P ve Q gibi çok büyük iki asal sayı seçilir.
- Bu iki asal sayının çarpımı N = P.Q ve bu bir eksiklerinin φ(N)=(P-1)(Q-1) hesaplanır.
- 1'den büyük φ(N)'den küçük φ(N) ile aralarında asal bir E tamsayısı seçilir.
- Seçilen E tamsayısının mod φ(N)'de tersi alınır, sonuç D gibi bir tamsayıdır.
- E ve N tamsayıları genel anahtarı, D ve N tamsayıları ise özel anahtarı oluşturur.

Genel ve özel anahtarları oluşturduktan sonra gönderilmek istenen bilgi genel anahtar ile şifrelenir. Şifreleme işlemi şu şekilde yapılmaktadır: Şifrelenecek bilginin sayısal karşılığının E' ninci kuvveti alınır ve bunun mod N deki karşılığı şifrelenmiş metni oluşturmaktadır. Genel anahtar ile şifrelenmiş bir metin ancak özel anahtar ile açılabilir. Bu yüzden şifrelenmiş metin, yine aynı yolla, şifrelenmiş metnin sayısal karşılığının D'ninci kuvveti alınır ve bunun mod N deki karşılığı orijinal metni oluşturur.

Basit bir örnek ile algoritmayı tekrar anlatalım. Örneğin basitliği açısından daha küçük asal sayılarla çalışacağız. Öncelikle genel ve özel anahtarlarımızı oluşturalım.

- P=7 ve Q=17 gibi iki asal sayı seçelim.
- Bu iki asal sayının çarpımı N=P.Q; N=119 ve bu iki asal sayının bir eksiklerinin çarpımı φ(N)=(P-1)(Q-1); φ(N)=96 olarak hesaplanır.
- 1'den büyük 96'dan küçük 96 ile aralarında asal bir E=5 tamsayısı seçelim.
- Seçilen E=5 tamsayısının mod 96'da tersi alınır, sonuç D=77 gibi bir tamsayıdır.
- 5 ve 119 tamsayıları genel anahtarı, 77 ve 119 tamsayıları ise özel anahtarı oluşturur.

Bu algoritmada iki asal sayının çarpımını kullanarak anahtar oluşturulmasının sebebi, iki asal sayının çarpımını asal çarpanlarına ayırmak asal olmayan sayıları ayırmaktan daha zorlu olmasıdır. Şimdi oluşturduğumuz {5, 119} ve {77, 119} anahtarlarımızı kullanarak şifreleme yapalım. Örnek olarak, 19 sayısını genel anahtarımızla {5, 119} şifreleyelim. 19 sayısının 5'inci kuvvetinin mod 119 daki karşılığı olan 66, 19 sayısının RSA şifrelenmiş halidir. Özel anahtarımızı {77, 119} kullanarak 66'nın 77'nci kuvvetinin mod 119 daki karşılığı tahmin de edebileceğiniz gibi 19 dur.

İki tamsayının aralarında asal olup olmadığının testi için matematikten de bildiğimiz Öklid algoritması kullanılır. Çok büyük asal sayı oluşturmak oldukça zor bir iştir. RSA ile günümüzde 1024 bitlik bir anahtar (yaklaşık 300 basamaklı bir sayı) basit uygulamalar için yeterli bir şifreleme tekniği olarak kullanılabilir.

RSA algoritması, bir şifreleme algoritması için oldukça basit bir algoritmadır. Buna karşın sürekli çok büyük asal sayı oluşturmak oldukça zor bir işlemdir. Asal sayılarının bilinen bir formülü

yoktur. RSA algoritmasını biz de yazabiliriz, ancak algoritmanın tüm detaylarıyla teker teker uğraşmak zorunda kalırdık. Bunun için, Microsoft .NET Framework' te hali hazırda RSA uygulaması geliştirebilmek için RSA sınıfı bulunmaktadır. Kullanımı oldukça basit olduğu için bir web servisi ile örneklendirebiliriz. Bunun için bir web servisi oluşturalım. Bu web servisine rsa isimli sınıfından bir değişken tanımlayalım. Bu web servisine şifreleme ve deşifreleme için aşağıdaki iki web metodu ekleyelim.

C#
```
[WebMethod]
public string Encrypt(string stringToEncrypt)
{
        byte[] buffer = Encoding.UTF8.GetBytes(stringToEncrypt);
        return Encoding.UTF8.GetString(rsa.Encrypt(buffer,false));
}

[WebMethod]
public string Decrypt(string stringToDecrypt)
{
        byte[] buffer = Encoding.UTF8.GetBytes(stringToDecrypt);
        return Encoding.UTF8.GetString(rsa.Decrypt(buffer,false));
}
```

VB.NET
```
<WebMethod()> _
Public Function Encrypt(ByVal stringToEncrypt As String) As String
        Dim buffer As Byte() = Encoding.UTF8.GetBytes(stringToEncrypt)
        Return Encoding.UTF8.GetString(rsa.Encrypt(buffer, False))
End Function

<WebMethod()> _
Public Function Decrypt(ByVal stringToDecrypt As String) As String
        Dim buffer As Byte() = Encoding.UTF8.GetBytes(stringToDecrypt)
        Return Encoding.UTF8.GetString(rsa.Decrypt(buffer, False))
End Function
```

C++.NET
```
[System::Web::Services::WebMethod]
String __gc* Encrypt(String __gc * stringToEncrypt)
{
        Byte buffer[] = Encoding::UTF8->GetBytes(stringToEncrypt);
        return Encoding::UTF8->GetString(rsa->Encrypt(buffer,false));
}

[System::Web::Services::WebMethod]
String __gc* Decrypt(String __gc * stringToDecrypt)
{
        Byte buffer[] = Encoding::UTF8->GetBytes(stringToDecrypt);
        return Encoding::UTF8->GetString(rsa->Decrypt(buffer,false));
}
```

J#
```
/** @attribute WebMethod() */
public String Encrypt(String stringToEncrypt)
{
    ubyte[] buffer = Encoding.get_UTF8().GetBytes(stringToEncrypt);
    return Encoding.get_UTF8().GetString( rsa.Encrypt(buffer,false) );
}

/** @attribute WebMethod() */
public String Decrypt(String stringToDecrypt)
```

```
{
    ubyte[] buffer = Encoding.get_UTF8().GetBytes(stringToDecrypt);
    return Encoding.get_UTF8().GetString( rsa.Decrypt(buffer,false) );
}
```

Bu iki metot RSA algoritmasını kullanarak şifreleme ve deşifreleme işlemini yapabilmektedir. Ancak bu kodu denerseniz Encrypt metodu ile şifrelediğiniz metni Decrypt ile açamadığınızı göreceksiniz. Bunun sebebi kullanılan anahtarları saklamamızdır. Dikkatinizi çekecek ikinci bir özellikte RSACryptoServiceProvider sınıfının Encrypt ve Decrypt metotlarının ikinci parametresidir. Burada örnek için false verdiğimiz değer, Microsoft Windows XP ve üzeri işletim sistemleri tarafından sağlanan bir özelliktir. OAEP (Optimal Asymmetric Encryption Padding) doldurmanın kullanılıp kullanılmayacağını belirtir. Şimdi System.Security.Cryptgraphy altındaki RSA ve RSACryptoServiceProvider sınıflarını inceleyelim.

RSA sınıfı RSA algoritmasını kullanmak isteyen kodların türetilmesi gereken ana sınıfı tanımlar. RSA sınıfı, AsymmetricAlgorithm sınıfından türeyen soyut bir sınıftır. Bu sınıfın erişilebilir özellikleri şunlardır:

- **KeyExchangeAlgorithm**, anahtar değişimi algoritmasının ismini belirtir. RSA için "RSA-PKCS1-KeyEx" tir.
- **KeySize**, şifreleme ve deşifreleme için kullanılacak anahtarların kaç bitten oluşacağını gösterir. Bu özelliğin değerini değiştirerek, kullanılacak bit sayısını ayarlayabilirsiniz. Varsayılan anahtar boyutu 1024 bittir.
- **LegalKeySize**, bu algoritma tarafından desteklenen geçerli anahtarlar bit olarak büyüklüğünü gösterir. RSA algoritması için anahtar büyüklüğü en az 384 bit en fazla 16384 bittir. 16384 bit 2KB büyüklüğünde bir anahtar anlamına gelmektir. Bu da yaklaşık 5000 basamaklı bir sayı anlamına gelmektedir.
- **SignatureAlgorithm**, imzalama için kullanılacak algoritmanın adını gösterir. RSA için http://www.w3.org/2000/09/xmldsig#rsa-sha1 'dir.

Bu sınıfın erişilebilir metotları ise şunlardır:

- **Clear**, RSA tarafından kullanılan tüm kaynakları sisteme geri verir.
- **Create**, RSA'nın özel uygulamalarının yazıldığı durumlarda nesnenin oluşturulduğunda yapılacak işleri içerir.
- **DecryptValue**, özel anahtar ile verilen bilgiyi deşifrelemek için kullanılır.
- **EncryptValue**, genel anahtar ile verilen bilgiyi şifrelemek için kullanılır.
- **Equals**, iki nesnenin birbirine eşit olup olmadığını test eder.
- **ExportParameters**, RSA'in tüm parametreleri özel ve genel anahtarlar dahil RSAParameter yapısından bir nesne içine kaydeder.
- **FromXmlString**, XML Deserileştirme gibidir. ToXMLString metodu ile XML'e aktarılmış nesneyi yeniden oluşturur, nesneyi XML'e yüklenmeden önceki durumuna getirir.
- **GetHashCode**, bellekteki o nesneye özgü bir hash kodu oluşturur.
- **GetType**, bu nesnenin tipini verir.
- **ImportParameters**, RSAParameters yapısından bir nesne içindeki, RSA'nın kullanacağı tüm parametreleri özel ve genel anahtarları dahil geri yükler.
- **ToString**, şu anki nesneyi ifade eden bir metin oluşturur.
- **ToXmlString**, XML Serileştirme gibidir. Nesnenin o anki durumunu yeniden oluşturulabilecek bir şekilde XML'e aktarır.

Şimdi, RSA sınıfından bahsederken ismi geçen RSAParameters yapısını inceleyelim. Bu yapı da yine System.Security.Cryptography altında yer almaktadır. Bu yapının en önemli özelliği serileştirilebiliyor olmasıdır. Bu özelliği sayesinde Binary(ikili) ve ya XML olarak bu yapıyı serileştirip, aktarabiliriz. Bu yapı içerisinde algoritmayı anlatırken kullandığımız P ve Q gibi semboller ile ifade ettiğimiz algoritmanın temel parametreleri yer almaktadır.

Örneklerimizde de yer alan RSACryptoServiceProvider sınıfı ise doğrudan RSA şifrelemesi için kullanılacak sınıfı ifade eder. Bu sınıf biraz evvel bahsettiğimiz RSA sınıfından türeyen mühürlü bir sınıftır. Mühürlü sınıf, hiç bir sınıfın kendisinden türetilemeyeceğini ifade eder. Bu sınıfın erişilebilir özellikleri şunlardır:

- **KeyExchangeAlgorithm**, anahtar değişimi algoritmasının ismini belirtir. RSA için "RSA-PKCS1-KeyEx" tir.
- **KeySize**, şifreleme ve deşifreleme için kullanılacak anahtarların kaç bitten oluşacağını gösterir. Bu özelliğin değerini değiştirerek, kullanılacak bit sayısını ayarlayabilirsiniz. Varsayılan anahtar boyutu 1024 bittir.
- **LegalKeySize**, bu algoritma tarafından desteklenen geçerli anahtarlar bit olarak büyüklüğünü gösterir. RSA algoritması için anahtar büyüklüğü en az 384 bit en fazla 16384 bittir. 16384 bit 2KB büyüklüğünde bir anahtar anlamına gelmektir. Bu da yaklaşık 5000 basamaklı bir sayı anlamına gelmektedir.
- **PersistKeyInCsp**, anahtarın CSP (Cryptographic Service Provider) içerisinde kalıcı olarak tutulup tutulmayacağını belirten özelliktir. Bu özelliğin değerini değiştirerek dışarıdan bu özelliği aktif/pasif kılabilirsiniz.
- **SignatureAlgorithm**, imzalama için kullanılacak algoritmanın adını gösterir. RSA için http://www.w3.org/2000/09/xmldsig#rsa-sha1 'dir.
- **UseMachineKeyStore**, PersistKeyInCsp ile bilgisayar üzerinde kalıcı olarak tutulması istenen anahtarları o an ki kullanıcı profilinde mi yoksa tüm kullanıcılar için ortak olan bir yerde mi tutulacağını belirtir.

Bu sınıfın erişilebilir metotları ise şunlardır:

- **Clear**, RSA tarafından kullanılan tüm kaynakları sisteme geri verir.
- **Decrypt**, RSA algoritması ile verilen bilgiyi deşifre eder.
- **DecryptValue**, özel anahtar ile verilen bilgiyi deşifrelemek için kullanılır. Ancak Microsoft .NET Framework' ün 1.1 versiyonunda bu metot desteklenmemektedir.
- **Encrypt**, RSA algoritması ile verilen bilgiyi şifreler.
- **EncryptValue**, genel anahtar ile verilen bilgiyi şifrelemek için kullanılır. Ancak Microsoft .NET Framework' ün 1.1 versiyonunda bu metot desteklenmemektedir.
- **Equals**, iki nesnenin birbirine eşit olup olmadığını test eder.
- **ExportParameters**, RSA'in tüm parametreleri özel ve genel anahtarlar dahil RSAParameter yapısından bir nesne içine kaydeder.
- **FromXmlString**, XML Deserileştirme gibidir. ToXMLString metodu ile XML'e aktarılmış nesneyi yeniden oluşturur, nesneyi XML'e yüklenmeden önceki durumuna getirir.
- **GetHashCode**, bellekteki o nesneye özgü bir hash kodu oluşturur.
- **GetType**, bu nesnenin tipini verir.
- **ImportParameters**, RSAParameters yapısından bir nesne içindeki, RSA'nın kullanacağı tüm parametreleri özel ve genel anahtarları dahil geri yükler.
- **SignData**, verilen bilginin hash değerini hesaplar ve bu değer ile veriyi imzalar.

- **SignHash**, belirtilen hash değeri için imzayı hesaplar ve özel anahtar ile bu bilgiyi şifreler.
- **ToString**, şu an ki nesneyi ifade eden bir metin oluşturur.
- **ToXmlString**, XML Serileştirme gibidir. Nesnenin o anki durumunu yeniden oluşturulabilecek bir şekilde XML'e aktarır.
- **VerifyData**, belirtilen bilgi için imzayı tekrar hesaplayıp var olan imza ile karşılaştırır.
- **VerifyHash**, belirtilen hash değeri için imzayı tekrar hesaplayıp var olan imza ile karşılaştırır.

RSA şifreleme tekniğini algoritmik olarak incelediğimiz bu makalemizde, RSA sınıfının kullanımından bahsettik. Bir sonraki makalemizde RSA anahtarlarının dağıtımı ve RSA kullanarak dijital imza oluşturmayı inceleyeceğiz. Bir sonraki makalemizde görüşünceye kadar güvende kalın...

Yunus Emre ALPÖZEN

25.03.2005