Cryptographic Signatures

Article • 08/10/2022 • 3 minutes to read

Cryptographic digital signatures use public key algorithms to provide data integrity. When you sign data with a digital signature, someone else can verify the signature, and can prove that the data originated from you and was not altered after you signed it. For more information about digital signatures, see Cryptographic Services.

This topic explains how to generate and verify digital signatures using classes in the System.Security.Cryptography namespace.

Generate a signature

Digital signatures are usually applied to hash values that represent larger data. The following example applies a digital signature to a hash value. First, a new instance of the RSA class is created to generate a public/private key pair. Next, the RSA is passed to a new instance of the RSAPKCS1SignatureFormatter class. This transfers the private key to the RSAPKCS1SignatureFormatter, which actually performs the digital signing. Before you can sign the hash code, you must specify a hash algorithm to use. This example uses the `SHA256` algorithm. Finally, the CreateSignature method is called to perform the signing.

C#

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

using SHA256 alg = SHA256.Create();

byte[] data = Encoding.ASCII.GetBytes("Hello, from the .NET Docs!");
byte[] hash = alg.ComputeHash(data);

RSAParameters sharedParameters;
byte[] signedHash;

// Generate signature
using (RSA rsa = RSA.Create())
{
    sharedParameters = rsa.ExportParameters(false);

    RSAPKCS1SignatureFormatter rsaFormatter = new(rsa);
    rsaFormatter.SetHashAlgorithm(nameof(SHA256));

    signedHash = rsaFormatter.CreateSignature(hash);
}

// The sharedParameters, hash, and signedHash are used to later verify the
signature.
```

Verify a signature

To verify that data was signed by a particular party, you must have the following information:

- The public key of the party that signed the data.
- The digital signature.
- The data that was signed.
- The hash algorithm used by the signer.

To verify a signature signed by the RSAPKCS1SignatureFormatter class, use the RSAPKCS1SignatureDeformatter class. The RSAPKCS1SignatureDeformatter class must be supplied the public key of the signer. For RSA, you will need at a minimal the values of the RSAParameters.Modulus and the RSAParameters.Exponent to specify the public key. One way to achieve this is to call RSA.ExportParameters during signature creation and then call RSA.ImportParameters during the verification process. The party that generated the public/private key pair should provide these values. First create an RSA object to hold the public key that will verify the signature, and then initialize an RSAParameters structure to the modulus and exponent values that specify the public key.

The following code shows the sharing of an RSAParameters structure. The `RSA` responsible for creating the signature exports its parameters. The parameters are then imported into the new `RSA` instance that is responsible for verifying the signature.

The RSA instance is, in turn, passed to the constructor of an RSAPKCS1SignatureDeformatter to transfer the key.

The following example illustrates this process. In this example, imagine that `sharedParameters`, `hash`, and `signedHash` are provided by a remote party. The remote party has signed `hash` using the `SHA256` algorithm to produce the digital signature `signedHash`. The RSAPKCS1SignatureDeformatter.VerifySignature method verifies that the digital signature is valid and was used to sign `hash`.

C#

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

using SHA256 alg = SHA256.Create();

byte[] data = Encoding.ASCII.GetBytes("Hello, from the .NET Docs!");
byte[] hash = alg.ComputeHash(data);

RSAParameters sharedParameters;
byte[] signedHash;

// Generate signature
using (RSA rsa = RSA.Create())
{
    sharedParameters = rsa.ExportParameters(false);

    RSAPKCS1SignatureFormatter rsaFormatter = new(rsa);
    rsaFormatter.SetHashAlgorithm(nameof(SHA256));

    signedHash = rsaFormatter.CreateSignature(hash);
}

// Verify signature
using (RSA rsa = RSA.Create())
{
    rsa.ImportParameters(sharedParameters);

    RSAPKCS1SignatureDeformatter rsaDeformatter = new(rsa);
    rsaDeformatter.SetHashAlgorithm(nameof(SHA256));

    if (rsaDeformatter.VerifySignature(hash, signedHash))
    {
        Console.WriteLine("The signature is valid.");
    }
    else
    {
        Console.WriteLine("The signature is not valid.");
    }
}
```

This code fragment displays "`The signature is valid`" if the signature is valid and "`The signature is not valid`" if it's not.

See also

- Cryptographic Services
- Cryptography Model
- Cross-Platform Cryptography
- ASP.NET Core Data Protection