Contents

Seguridad en .NET

02/07/2020 • 2 minutes to read • Edit Online

El Common Language Runtime y .NET proporcionan muchas clases y servicios útiles que permiten a los desarrolladores escribir código seguro fácilmente y permitir a los administradores del sistema personalizar los permisos concedidos al código para que pueda acceder a los recursos protegidos. Además, el tiempo de ejecución y .NET proporcionan clases y servicios útiles que facilitan el uso de la criptografía y la seguridad basada en roles.

En esta sección

- Conceptos clave de seguridad
  Proporciona información general de las características de seguridad de Common Language Runtime. Esta sección está dirigida a desarrolladores y administradores de sistemas.
- Seguridad basada en roles
  Describe cómo interactuar en su código con la seguridad basada en roles. Esta sección es de interés para desarrolladores.
- Modelo de criptografía
  Proporciona información general sobre los servicios criptográficos proporcionados por .NET. Esta sección es de interés para desarrolladores.
- Instrucciones de codificación segura
  Describe algunas de las prácticas recomendadas para crear aplicaciones .NET confiables. Esta sección es de interés para desarrolladores.

Secciones relacionadas

Guía de desarrollo
Proporciona una guía para todas las áreas y tareas tecnológicas principales para el desarrollo de aplicaciones, como la creación, configuración, depuración, seguridad e implementación de la aplicación, e información sobre programación dinámica, interoperabilidad, extensibilidad, administración de memoria y subprocesamiento.

Conceptos clave de seguridad

02/07/2020 • 10 minutes to read • Edit Online

Microsoft .NET Framework ofrece seguridad basada en roles para ayudar a solucionar problemas de seguridad relativos a código móvil y para permitir que los componentes determinen qué usuarios tienen autorización para operar.

Protección y seguridad de tipos

El código seguro de tipos sólo tiene acceso a las ubicaciones de memoria para las que tiene autorización. (En este debate, la seguridad de tipos se refiere específicamente a la seguridad de tipos de memoria y no debe confundirse con la seguridad de tipos en un respeto más amplio). Por ejemplo, el código seguro de tipos no puede leer valores de los campos privados de otro objeto. Sólo puede obtener acceso a tipos siguiendo métodos permitidos y perfectamente definidos.

Durante la compilación Just-in-time (JIT), un proceso opcional de comprobación examina los metadatos y el lenguaje intermedio de Microsoft (MSIL) de los métodos a compilar a código máquina nativo, para comprobar si tienen seguridad de tipos. Este proceso se omite si el código tiene permiso para evitar la comprobación. Para información sobre la comprobación, consulte Proceso de ejecución administrada.

Aunque la comprobación de la seguridad de tipos no es obligatoria para la ejecución de código administrado, la seguridad de tipos desempeña un rol crucial en el aislamiento del ensamblado y la exigencia de seguridad. Cuando el código tiene seguridad de tipos, Common Language Runtime puede aislar totalmente ensamblados entre sí. Este aislamiento ayuda a garantizar que los ensamblados no puedan ejercer influencias negativas entre sí y aumenta la confiabilidad de la aplicación. Los componentes seguros de tipos se pueden ejecutar de forma segura en el mismo proceso aunque dispongan de confianza en diferentes niveles. Cuando el código no tiene seguridad de tipos, pueden producirse efectos secundarios no deseados. Por ejemplo, el runtime no puede evitar la llamada de código administrado en código nativo (no administrado) ni la realización de operaciones malintencionadas. Cuando el código tiene seguridad de tipos, el mecanismo de exigencia de seguridad del motor en tiempo de ejecución garantiza que no tiene acceso a código nativo salvo que tenga permiso explícito para ello. El código sin seguridad de tipos debe haber obtenido permiso SecurityPermission con el miembro de enumeración SkipVerification transferido para la ejecución.

Para obtener más información, vea Conceptos básicos sobre la seguridad de acceso del código.

Principal

Una entidad de seguridad (principal) representa la identidad y el rol de un usuario y actúa en nombre del usuario. La seguridad basada en roles de .NET Framework admite tres tipos de entidades de seguridad:

- Las entidades de seguridad genéricas representan usuarios y roles que son independientes de los roles y los usuarios de Windows.
- Las entidades de seguridad de Windows representan a los usuarios de Windows y sus roles (o grupos de Windows). Una entidad de seguridad de Windows puede suplantar a otro usuario, lo que significa que la entidad de seguridad puede tener acceso a un recurso en nombre de un usuario presentando la identidad que pertenece a dicho usuario.
- Una aplicación puede definir entidades de seguridad personalizadas que se adapten a las necesidades de esa aplicación en particular. Asimismo, se puede ampliar la noción básica de la identidad y los roles de la entidad de seguridad.

Para más información, consulte Objetos Principal e Identity.

Authentication

La autenticación es el proceso de detectar y comprobar la identidad de una entidad de seguridad mediante el análisis de las credenciales del usuario y la validación de esas credenciales en alguna autoridad. Su código puede usar fácilmente la información obtenida durante la autenticación. También puede usar la seguridad basada en roles de .NET Framework para autenticar al usuario actual y para determinar si esa entidad de seguridad puede tener acceso a su código. Vea las sobrecargas del método WindowsPrincipal.IsInRole para obtener ejemplos de cómo autenticar la entidad de seguridad para roles específicos. Por ejemplo, puede usar la sobrecarga WindowsPrincipal.IsInRole(String) para determinar si el usuario actual es miembro del grupo Administradores.

Hoy en día se usan diversos mecanismos de autenticación, muchos de los cuales pueden utilizarse con la seguridad basada en roles de .NET Framework. Algunos de los mecanismos más usados son básico, implícito, Passport, sistema operativo (como NTLM o Kerberos) o los mecanismos definidos por la aplicación.

Ejemplo

El ejemplo siguiente requiere que la entidad de seguridad activa sea un administrador. El parámetro `name` es `null`, que permite que cualquier usuario que sea administrador pase la petición.

> **NOTE**
> En Windows Vista, el control de cuentas de usuario (UAC) determina los privilegios de un usuario. Si es miembro del grupo Administradores integrados, se le asignarán dos símbolos (tokens) de acceso en tiempo de ejecución: un símbolo (token) de acceso de usuario estándar y un símbolo (token) de acceso de administrador. De forma predeterminada, se le asignará el rol de usuario estándar. Para ejecutar código que requiere permisos de administrador, primero debe elevar el nivel de sus privilegios de usuario estándar a administrador. Para ello, inicie una aplicación haciendo clic con el botón derecho en el icono de la aplicación e indique que desea ejecutarla como administrador.

```
using namespace System;
using namespace System::Security;
using namespace System::Security::Permissions;
using namespace System::Security::Policy;
using namespace System::Security::Principal;

int main(array<System::String ^> ^args)
{
    System::String^ null;
    AppDomain::CurrentDomain->SetPrincipalPolicy(PrincipalPolicy::WindowsPrincipal);
    PrincipalPermission^ principalPerm = gcnew PrincipalPermission(null, "Administrators" );
      principalPerm->Demand();
      Console::WriteLine("Demand succeeded");
    return 0;
}
```

```
using System;
using System.Threading;
using System.Security.Permissions;
using System.Security.Principal;

class SecurityPrincipalDemo
{

    public static void Main()
    {
        AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
        PrincipalPermission principalPerm = new PrincipalPermission(null, "Administrators");
        principalPerm.Demand();
        Console.WriteLine("Demand succeeded.");
    }
}
```

```
Imports System.Threading
Imports System.Security.Permissions
Imports System.Security.Principal



Class SecurityPrincipalDemo


    Public Shared Sub Main()
        AppDomain.CurrentDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal)
        Dim principalPerm As New PrincipalPermission(Nothing, "Administrators")
        principalPerm.Demand()
        Console.WriteLine("Demand succeeded.")

    End Sub
End Class
```

En el ejemplo siguiente se muestra cómo determinar la identidad de la entidad de seguridad y los roles disponibles para dicha entidad de seguridad. Una posible aplicación de este ejemplo sería la confirmación de que el usuario actual está en un rol permitido para usar la aplicación.

```
public:
   static void DemonstrateWindowsBuiltInRoleEnum()
   {
      AppDomain^ myDomain = Thread::GetDomain();

      myDomain->SetPrincipalPolicy( PrincipalPolicy::WindowsPrincipal );
      WindowsPrincipal^ myPrincipal = dynamic_cast<WindowsPrincipal^>(Thread::CurrentPrincipal);

      Console::WriteLine( "{0} belongs to: ", myPrincipal->Identity->Name );

      Array^ wbirFields = Enum::GetValues( WindowsBuiltInRole::typeid );

      for each ( Object^ roleName in wbirFields )
      {
         try
         {
            Console::WriteLine( "{0}? {1}.", roleName,
               myPrincipal->IsInRole(  *dynamic_cast<WindowsBuiltInRole^>(roleName) ) );
         }
         catch ( Exception^ )
         {
            Console::WriteLine( "{0}: Could not obtain role for this RID.",
               roleName );
         }
      }
   }
```

```
using System;
using System.Threading;
using System.Security.Permissions;
using System.Security.Principal;

class SecurityPrincipalDemo
{
    public static void DemonstrateWindowsBuiltInRoleEnum()
    {
        AppDomain myDomain = Thread.GetDomain();

        myDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal);
        WindowsPrincipal myPrincipal = (WindowsPrincipal)Thread.CurrentPrincipal;
        Console.WriteLine("{0} belongs to: ", myPrincipal.Identity.Name.ToString());
        Array wbirFields = Enum.GetValues(typeof(WindowsBuiltInRole));
        foreach (object roleName in wbirFields)
        {
            try
            {
                // Cast the role name to a RID represented by the WindowsBuildInRole value.
                Console.WriteLine("{0}? {1}.", roleName,
                    myPrincipal.IsInRole((WindowsBuiltInRole)roleName));
                Console.WriteLine("The RID for this role is: " + ((int)roleName).ToString());
            }
            catch (Exception)
            {
                Console.WriteLine("{0}: Could not obtain role for this RID.",
                    roleName);
            }
        }
        // Get the role using the string value of the role.
        Console.WriteLine("{0}? {1}.", "Administrators",
            myPrincipal.IsInRole("BUILTIN\\" + "Administrators"));
        Console.WriteLine("{0}? {1}.", "Users",
            myPrincipal.IsInRole("BUILTIN\\" + "Users"));
        // Get the role using the WindowsBuiltInRole enumeration value.
        Console.WriteLine("{0}? {1}.", WindowsBuiltInRole.Administrator,
           myPrincipal.IsInRole(WindowsBuiltInRole.Administrator));
        // Get the role using the WellKnownSidType.
        SecurityIdentifier sid = new SecurityIdentifier(WellKnownSidType.BuiltinAdministratorsSid, null);
        Console.WriteLine("WellKnownSidType BuiltinAdministratorsSid  {0}? {1}.", sid.Value,
myPrincipal.IsInRole(sid));
    }

    public static void Main()
    {
        DemonstrateWindowsBuiltInRoleEnum();
    }
}
```

```
Imports System.Threading
Imports System.Security.Permissions
Imports System.Security.Principal

Class SecurityPrincipalDemo

    Public Shared Sub DemonstrateWindowsBuiltInRoleEnum()
        Dim myDomain As AppDomain = Thread.GetDomain()

        myDomain.SetPrincipalPolicy(PrincipalPolicy.WindowsPrincipal)
        Dim myPrincipal As WindowsPrincipal = CType(Thread.CurrentPrincipal, WindowsPrincipal)
        Console.WriteLine("{0} belongs to: ", myPrincipal.Identity.Name.ToString())
        Dim wbirFields As Array = [Enum].GetValues(GetType(WindowsBuiltInRole))
        Dim roleName As Object
        For Each roleName In wbirFields
            Try
                ' Cast the role name to a RID represented by the WindowsBuildInRole value.
                Console.WriteLine("{0}? {1}.", roleName, myPrincipal.IsInRole(CType(roleName,
WindowsBuiltInRole)))
                Console.WriteLine("The RID for this role is: " + Fix(roleName).ToString())

            Catch
                Console.WriteLine("{0}: Could not obtain role for this RID.", roleName)
            End Try
        Next roleName
        ' Get the role using the string value of the role.
        Console.WriteLine("{0}? {1}.", "Administrators", myPrincipal.IsInRole("BUILTIN\" + "Administrators"))
        Console.WriteLine("{0}? {1}.", "Users", myPrincipal.IsInRole("BUILTIN\" + "Users"))
        ' Get the role using the WindowsBuiltInRole enumeration value.
        Console.WriteLine("{0}? {1}.", WindowsBuiltInRole.Administrator,
myPrincipal.IsInRole(WindowsBuiltInRole.Administrator))
        ' Get the role using the WellKnownSidType.
        Dim sid As New SecurityIdentifier(WellKnownSidType.BuiltinAdministratorsSid, Nothing)
        Console.WriteLine("WellKnownSidType BuiltinAdministratorsSid  {0}? {1}.", sid.Value,
myPrincipal.IsInRole(sid))

    End Sub

    Public Shared Sub Main()
        DemonstrateWindowsBuiltInRoleEnum()

    End Sub
End Class
```

Authorization

La autorización es el proceso de determinar si una entidad de seguridad puede realizar una acción solicitada. La autorización se produce después de la autenticación y usa información sobre la identidad y los roles de la entidad de seguridad para determinar a qué recursos puede tener acceso la entidad de seguridad. Para implementar la autorización, se puede usar la seguridad basada en roles de .NET Framework.

Compatibilidad con Estándar federal de procesamiento de información de .NET Core (FIPS)

27/02/2020 • 2 minutes to read • Edit Online

La publicación Estándar federal de procesamiento de información (FIPS) 140-2 es un estándar del gobierno de EE. UU. que define los requisitos de seguridad mínimos para los módulos criptográficos en los productos de tecnología de la información, como se define en la sección 5131 de la información. Acto de reforma de administración de tecnología de 1996.

.NET Core:

- Pasa las llamadas primitivas criptográficas a través de a los módulos estándar que proporciona el sistema operativo subyacente.
- No **exige el** uso de algoritmos o tamaños de clave aprobados por FIPS en aplicaciones de .net Core.

El administrador del sistema es responsable de configurar la compatibilidad con FIPS para un sistema operativo.

Si el código se escribe para un entorno compatible con FIPS, el desarrollador es responsable de garantizar que no se usen algoritmos FIPS no compatibles.

Para obtener más información sobre la conformidad con FIPS, consulte los siguientes artículos:

- Compatibilidad con FIPS de Windows
- Configuración de Windows para la compatibilidad con FIPS
- Capítulo 8. Normas y regulaciones federales Red Hat Enterprise Linux 7

Seguridad basada en roles

02/07/2020 • 5 minutes to read • Edit Online

Suelen usarse roles en aplicaciones empresariales o financieras para aplicar la directiva. Por ejemplo, una aplicación puede imponer límites en el tamaño de la transacción que se va a procesar según si el usuario que realiza la solicitud es un miembro de rol especificado. Los empleados podrían tener autorización para procesar las transacciones que son inferiores a un umbral especificado, los supervisores podrían tener un límite mayor y los vicepresidentes podrían tener un límite aún mayor (o ningún límite). La seguridad basada en roles también se puede usar cuando una aplicación requiere varias aprobaciones para completar una acción. Este caso podría ser un sistema de compras en el que cualquier empleado puede generar una solicitud de compra, pero solo un agente de compras puede convertir la solicitud en un pedido de compras que se pueden enviar a un proveedor.

La seguridad basada en roles de .NET Framework admite la autorización haciendo que la información sobre el principal, que se construye a partir de una identidad asociada, esté disponible para el subproceso actual. La identidad (y la entidad de seguridad que ayuda a definir) puede estar basada en una cuenta de Windows o puede ser una identidad personalizada no relacionada con una cuenta de Windows. Las aplicaciones de .NET Framework pueden tomar decisiones sobre autorización basándose en la identidad de la entidad de seguridad, en su pertenencia a un rol o ambas. Un rol es un conjunto de entidades de seguridad que tienen los mismos privilegios de seguridad (como un tesorero o un administrador). Una entidad de seguridad puede ser miembro de uno o varios roles. Por lo tanto, las aplicaciones pueden usar la pertenencia a un rol para determinar si una entidad de seguridad está autorizada para realizar una acción solicitada.

Para facilitar el uso y mejorar la coherencia con la seguridad de acceso del código, la seguridad basada en roles de .NET Framework proporciona objetos System.Security.Permissions.PrincipalPermission que permiten a Common Language Runtime realizar la autorización de forma similar a las comprobaciones de seguridad de acceso del código. La clase PrincipalPermission representa la identidad o el rol con los que la entidad de seguridad debe coincidir y es compatible con las comprobaciones de seguridad declarativa e imperativa. También puede acceder directamente a la información de identidad de una entidad de seguridad y realizar las comprobaciones de identidad y de rol en el código cuando sea necesario.

.NET Framework proporciona compatibilidad de seguridad basada en roles que es suficientemente flexible y ampliable para satisfacer las necesidades de una amplia variedad de aplicaciones. Puede elegir interoperar con infraestructuras de autenticación existentes, como los servicios COM+ 1.0, o crear un sistema de autenticación personalizado. La seguridad basada en roles está especialmente indicada para aplicaciones web de ASP.NET, que se procesan principalmente en el servidor. Sin embargo, la seguridad basada en roles de .NET Framework puede usarse en el cliente o en el servidor.

Antes de leer esta sección, asegúrese de que comprende el material presentado en conceptos clave de seguridad.

Temas relacionados

TITLE	DESCRIPCIÓN
Objetos Principal e Identity	Explica cómo configurar y administrar identidades y entidades de seguridad de Windows y genéricas.
Conceptos clave de seguridad	Presenta los conceptos fundamentales que debe conocer antes de usar la seguridad de .NET Framework.

Referencia

System.Security.Permissions.PrincipalPermission

Objetos Principal e Identity

02/07/2020 • 7 minutes to read • Edit Online

El código administrado puede detectar la identidad o el rol de una entidad de seguridad a través de un IPrincipal objeto, que contiene una referencia a un IIdentity objeto. La comparación de objetos Identity y Principal puede resultar útil en conceptos familiares, como cuentas de usuario y de grupo. En la mayoría de los entornos de red, las cuentas de usuario representan a personas o programas, mientras que las cuentas de grupo representan a ciertas categorías de usuarios y los derechos que poseen. De forma similar, los objetos Identity de .NET Framework representan a usuarios, mientras que los roles representan pertenencias y contextos de seguridad. En .NET Framework, el objeto Principal encapsula un objeto Identity y un rol. Las aplicaciones .NET Framework conceden derechos a la entidad de seguridad basándose en su identidad o, lo que es más normal, en su pertenencia a un rol.

Objetos Identity

El objeto Identity encapsula información sobre el usuario o la entidad que se está validando. En su nivel más básico, los objetos Identity contienen un nombre y un tipo de autenticación. El nombre puede ser un nombre de usuario o el nombre de una cuenta de Windows, mientras que el tipo de autenticación puede ser un protocolo de inicio de admisión admitido, como Kerberos V5, o un valor personalizado. El .NET Framework define un GenericIdentity objeto que se puede utilizar para la mayoría de los escenarios de inicio de sesión personalizados y un objeto más especializado WindowsIdentity que se puede usar cuando se desea que la aplicación se base en la autenticación de Windows. Además, puede definir la clase de identidad propia que encapsula la información personalizada del usuario.

La IIdentity interfaz define las propiedades para tener acceso a un nombre y un tipo de autenticación, como Kerberos V5 o NTLM. Todas las clases **Identity** implementan la interfaz **IIdentity**. No hay una relación necesaria entre un objeto **Identity** y el token de proceso de Windows NT bajo el cual se está ejecutando actualmente un subproceso. No obstante, si el objeto **Identity** es un objeto **WindowsIdentity**, se supone que la identidad representa un token de seguridad de Windows NT.

Objetos Principal

El objeto Principal representa el contexto de seguridad bajo el cual se ejecuta código. Las aplicaciones que implementan seguridad basada en roles conceden derechos tomando como base el rol asociado a un objeto Principal. Al igual que los objetos de identidad, el .NET Framework proporciona un GenericPrincipal objeto y un WindowsPrincipal objeto. También puede definir sus propias clases de entidad de seguridad personalizadas.

La IPrincipal interfaz define una propiedad para tener acceso a un objeto de **identidad** asociado, así como un método para determinar si el usuario identificado por el objeto de **entidad** de seguridad es miembro de un rol determinado. Todas las clases **Principal** implementan la interfaz **IPrincipal**, así como las propiedades y métodos adicionales necesarios. Por ejemplo, Common Language Runtime proporciona la clase **WindowsPrincipal**, que implementa funcionalidad adicional para asignar a roles la pertenencia a un grupo de Windows NT o Windows 2000.

Un objeto de **entidad** de seguridad se enlaza a un CallContext objeto de contexto de llamada () dentro de un dominio de aplicación (AppDomain). Un contexto de llamada predeterminado se crea siempre con cada dominio **AppDomain** nuevo, por lo que siempre hay un contexto de llamada disponible para aceptar el objeto **Principal**. Cuando se crea un subproceso nuevo, también se crea un objeto **CallContext** para el subproceso. La referencia del objeto **Principal** se copia automáticamente desde el subproceso creador en el objeto **CallContext** del subproceso nuevo. Si el tiempo de ejecución no puede determinar qué objeto **Principal** pertenece al creador del

subproceso, sigue la directiva predeterminada para la creación de objetos **Principal** e **Identity**.

Una directiva específica de dominio de aplicación configurable define las reglas para decidir qué tipo de objeto **Principal** se ha de asociar a un dominio de aplicación nuevo. Cuando la directiva de seguridad lo permite, el tiempo de ejecución puede crear objetos **Principal** e **Identity** que reflejan el token de sistema operativo asociado al subproceso actual de ejecución. De forma predeterminada, el tiempo de ejecución utiliza objetos **Principal** e **Identity** que representan a usuarios no autenticados. El tiempo de ejecución no crea estos objetos **Principal** y **Identity** predeterminados hasta que el código trata de tener acceso a ellos.

El código de confianza que crea un dominio de aplicación puede establecer la directiva de dominio de aplicación que controla la construcción de los objetos **Principal** e **Identity** predeterminados. Esta directiva específica de dominio de aplicación se aplica a todos los subprocesos de ejecución de dicho dominio. Un host de confianza sin administrar inherentemente tiene la capacidad de establecer esta Directiva, pero el código administrado que establece esta Directiva debe tener el System.Security.Permissions.SecurityPermission para controlar la Directiva de dominio.

Cuando se transmite un objeto **Principal** a través de dominios de aplicación, pero dentro del mismo proceso (y, por tanto, en el mismo equipo), la infraestructura de comunicación remota copia una referencia en el objeto **Principal** asociado al contexto de quien efectúa una llamada en el contexto de quien la recibe.

Consulte también

- Procedimiento para crear un objeto WindowsPrincipal
- Procedimiento para crear objetos GenericPrincipal y GenericIdentity
- Reemplazar un objeto Principal
- Suplantar y revertir
- Seguridad basada en roles
- Conceptos clave de seguridad

Procedimiento para crear un objeto WindowsPrincipal

02/07/2020 • 3 minutes to read • Edit Online

Hay dos maneras de crear un objeto WindowsPrincipal, dependiendo de si el código debe realizar la validación basada en rol una sola vez o varias veces.

Si el código debe realizar la validación basada en rol varias veces, el primero de los procedimientos indicados a continuación produce menos sobrecarga. Cuando solo hace falta llevar a cabo una vez validaciones basadas en rol, puede crear un objeto WindowsPrincipal usando el segundo de los procedimientos indicados a continuación.

Para crear un objeto WindowsPrincipal para validar varias veces

1. Llame al método SetPrincipalPolicy en el objeto AppDomain que devuelve la propiedad AppDomain.CurrentDomain estática, pasando al método un valor de enumeración PrincipalPolicy que indica cuál debe ser la nueva directiva. Los valores admitidos son NoPrincipal, UnauthenticatedPrincipal y WindowsPrincipal. El siguiente código muestra cómo llamar a este método.

```
AppDomain.CurrentDomain.SetPrincipalPolicy(
    PrincipalPolicy.WindowsPrincipal);
```

```
AppDomain.CurrentDomain.SetPrincipalPolicy( _
    PrincipalPolicy.WindowsPrincipal)
```

2. Una vez que se haya establecido la directiva, use la propiedad Thread.CurrentPrincipal estática para recuperar la entidad de seguridad que encapsula al usuario de Windows actual. Dado que el tipo de devolución de la propiedad es IPrincipal, debe convertir el resultado en un tipo WindowsPrincipal. El siguiente código inicializa un nuevo objeto WindowsPrincipal en el valor de la entidad de seguridad asociada al subproceso actual.

```
WindowsPrincipal myPrincipal =
    (WindowsPrincipal) Thread.CurrentPrincipal;
```

```
Dim myPrincipal As WindowsPrincipal = _
    CType(Thread.CurrentPrincipal, WindowsPrincipal)
```

3. Una vez que se ha creado el objeto de entidad de seguridad, dispone de varios métodos para validarlo.

Para crear un objeto WindowsPrincipal para validar una sola vez

1. Inicialice un nuevo objeto WindowsIdentity llamando al método WindowsIdentity.GetCurrent estático, que consulta la cuenta de Windows actual y coloca la información sobre esa cuenta en el objeto de identidad recién creado. El siguiente código crea un nuevo objeto WindowsIdentity y lo inicializa en el usuario autenticado actual.

```
WindowsIdentity myIdentity = WindowsIdentity.GetCurrent();
```

```
Dim myIdentity As WindowsIdentity = WindowsIdentity.GetCurrent()
```

2. Cree un nuevo objeto WindowsPrincipal y pásele el valor del objeto WindowsIdentity que se creó en el paso anterior.

```
WindowsPrincipal myPrincipal = new WindowsPrincipal(myIdentity);
```

```
Dim myPrincipal As New WindowsPrincipal(myIdentity)
```

3. Una vez que se ha creado el objeto de entidad de seguridad, dispone de varios métodos para validarlo.

Consulte también

- Objetos Principal e Identity

Procedimiento para crear objetos GenericPrincipal y GenericIdentity

02/07/2020 • 3 minutes to read • Edit Online

Puede utilizar la GenericIdentity clase junto con la GenericPrincipal clase para crear un esquema de autorización que exista de manera independiente de un dominio de Windows.

Para crear un objeto GenericPrincipal

1. Cree una nueva instancia de la clase de identidad e inicialícela con el nombre que desee conservar. El código siguiente crea un nuevo objeto **GenericIdentity** y lo inicializa con el nombre `MyUser`.

```
Dim myIdentity As New GenericIdentity("MyUser")
```

```
GenericIdentity myIdentity = new GenericIdentity("MyUser");
```

2. Cree una nueva instancia de la clase **GenericPrincipal** e inicialícela con el objeto **GenericIdentity** creado previamente y una matriz de cadenas que representan los roles que desee asociar a esta entidad de seguridad. En el ejemplo de código siguiente se especifica una matriz de cadenas que representa un rol de administrador y un rol de usuario. **GenericPrincipal** se inicializa con el **GenericIdentity** anterior y la matriz de cadenas.

```
Dim myStringArray As String() = {"Manager", "Teller"}
DIm myPrincipal As New GenericPrincipal(myIdentity, myStringArray)
```

```
String[] myStringArray = {"Manager", "Teller"};
GenericPrincipal myPrincipal = new GenericPrincipal(myIdentity, myStringArray);
```

3. Utilice el código siguiente para asociar la entidad de seguridad al subproceso actual. Esto es útil en situaciones en las que la entidad de seguridad debe validarse varias veces, debe ser validada por otro código que se ejecute en la aplicación o debe ser validada por un PrincipalPermission objeto. Todavía puede realizar la validación basada en roles en el objeto principal sin asociarlo al subproceso. Para más información, consulte Reemplazar un objeto Principal.

```
Thread.CurrentPrincipal = myPrincipal
```

```
Thread.CurrentPrincipal = myPrincipal;
```

Ejemplo

En el ejemplo de código siguiente se muestra cómo crear una instancia de **GenericPrincipal** y **GenericIdentity**. Este código muestra los valores de estos objetos en la consola.

```
Imports System.Security.Principal
Imports System.Threading

Public Class Class1

    Public Shared Sub Main()
        ' Create generic identity.
        Dim myIdentity As New GenericIdentity("MyIdentity")

        ' Create generic principal.
        Dim myStringArray As String() =  {"Manager", "Teller"}
        Dim myPrincipal As New GenericPrincipal(myIdentity, myStringArray)

        ' Attach the principal to the current thread.
        ' This is not required unless repeated validation must occur,
        ' other code in your application must validate, or the
        ' PrincipalPermission object is used.
        Thread.CurrentPrincipal = myPrincipal

        ' Print values to the console.
        Dim name As String = myPrincipal.Identity.Name
        Dim auth As Boolean = myPrincipal.Identity.IsAuthenticated
        Dim isInRole As Boolean = myPrincipal.IsInRole("Manager")

        Console.WriteLine("The name is: {0}", name)
        Console.WriteLine("The isAuthenticated is: {0}", auth)
        Console.WriteLine("Is this a Manager? {0}", isInRole)

    End Sub

End Class
```

```
using System;
using System.Security.Principal;
using System.Threading;

public class Class1
{
    public static int Main(string[] args)
    {
    // Create generic identity.
    GenericIdentity myIdentity = new GenericIdentity("MyIdentity");

    // Create generic principal.
    String[] myStringArray = {"Manager", "Teller"};
    GenericPrincipal myPrincipal =
        new GenericPrincipal(myIdentity, myStringArray);

    // Attach the principal to the current thread.
    // This is not required unless repeated validation must occur,
    // other code in your application must validate, or the
    // PrincipalPermission object is used.
    Thread.CurrentPrincipal = myPrincipal;

    // Print values to the console.
    String name =  myPrincipal.Identity.Name;
    bool auth =  myPrincipal.Identity.IsAuthenticated;
    bool isInRole =  myPrincipal.IsInRole("Manager");

    Console.WriteLine("The name is: {0}", name);
    Console.WriteLine("The isAuthenticated is: {0}", auth);
    Console.WriteLine("Is this a Manager? {0}", isInRole);

    return 0;
    }
}
```

Cuando se ejecuta, la aplicación muestra un resultado similar al siguiente.

```
The Name is: MyIdentity
The IsAuthenticated is: True
Is this a Manager? True
```

Consulte también

- GenericIdentity
- GenericPrincipal
- PrincipalPermission
- Reemplazar un objeto Principal
- Objetos Principal e Identity

Reemplazar un objeto Principal

02/07/2020 • 3 minutes to read • Edit Online

Las aplicaciones que ofrecen servicios de autenticación deben poder reemplazar el objeto **Principal** (IPrincipal) de un subproceso determinado. Además, el sistema de seguridad debe ayudar a proteger la capacidad de reemplazar objetos **Principal** , porque un objeto **Principal** incorrecto asociado de forma malintencionada pone en peligro la seguridad de la aplicación mediante la notificación de una identidad o un rol falsos. Por lo tanto, las aplicaciones que requieren la capacidad de reemplazar objetos **Principal** necesitan que se les conceda el objeto System.Security.Permissions.SecurityPermission para el control de la entidad de seguridad. (Tenga en cuenta que este permiso no se requiere para realizar comprobaciones de seguridad basada en roles o para crear objetos **Principal**).

El objeto **Principal** actual se puede reemplazar realizando las tareas siguientes:

1. Cree el objeto **Principal** de reemplazo y el objeto **Identity** asociado.
2. Adjunte el nuevo objeto **Principal** al contexto de llamada.

Ejemplo

En el ejemplo siguiente se muestra cómo se crea un objeto de entidad de seguridad genérico y cómo usarlo para establecer la entidad de seguridad de un subproceso.

```
using System;
using System.Threading;
using System.Security.Permissions;
using System.Security.Principal;

class SecurityPrincipalDemo
{
    public static void Main()
    {
        // Retrieve a GenericPrincipal that is based on the current user's
        // WindowsIdentity.
        GenericPrincipal genericPrincipal = GetGenericPrincipal();

        // Retrieve the generic identity of the GenericPrincipal object.
        GenericIdentity principalIdentity =
            (GenericIdentity)genericPrincipal.Identity;

        // Display the identity name and authentication type.
        if (principalIdentity.IsAuthenticated)
        {
            Console.WriteLine(principalIdentity.Name);
            Console.WriteLine("Type:" + principalIdentity.AuthenticationType);
        }

        // Verify that the generic principal has been assigned the
        // NetworkUser role.
        if (genericPrincipal.IsInRole("NetworkUser"))
        {
            Console.WriteLine("User belongs to the NetworkUser role.");
        }

        Thread.CurrentPrincipal = genericPrincipal;
    }

    // Create a generic principal based on values from the current
    // WindowsIdentity
```

```
    // WindowsIdentity.
    private static GenericPrincipal GetGenericPrincipal()
    {
        // Use values from the current WindowsIdentity to construct
        // a set of GenericPrincipal roles.
        WindowsIdentity windowsIdentity = WindowsIdentity.GetCurrent();
        string[] roles = new string[10];
        if (windowsIdentity.IsAuthenticated)
        {
            // Add custom NetworkUser role.
            roles[0] = "NetworkUser";
        }

        if (windowsIdentity.IsGuest)
        {
            // Add custom GuestUser role.
            roles[1] = "GuestUser";
        }

        if (windowsIdentity.IsSystem)
        {
            // Add custom SystemUser role.
            roles[2] = "SystemUser";
        }

        // Construct a GenericIdentity object based on the current Windows
        // identity name and authentication type.
        string authenticationType = windowsIdentity.AuthenticationType;
        string userName = windowsIdentity.Name;
        GenericIdentity genericIdentity =
            new GenericIdentity(userName, authenticationType);

        // Construct a GenericPrincipal object based on the generic identity
        // and custom roles for the user.
        GenericPrincipal genericPrincipal =
            new GenericPrincipal(genericIdentity, roles);

        return genericPrincipal;
    }
}
```

```
Imports System.Threading
Imports System.Security.Permissions
Imports System.Security.Principal



Class SecurityPrincipalDemo

    ' Create a generic principal based on values from the current
    ' WindowsIdentity.
    Private Shared Function GetGenericPrincipal() As GenericPrincipal
        ' Use values from the current WindowsIdentity to construct
        ' a set of GenericPrincipal roles.
        Dim windowsIdentity As WindowsIdentity = windowsIdentity.GetCurrent()
        Dim roles(9) As String
        If windowsIdentity.IsAuthenticated Then
            ' Add custom NetworkUser role.
            roles(0) = "NetworkUser"
        End If

        If windowsIdentity.IsGuest Then
            ' Add custom GuestUser role.
            roles(1) = "GuestUser"
        End If

        If windowsIdentity.IsSystem Then
            ' Add custom SystemUser role.
```

```
            ' Add custom SystemUser role.
            roles(2) = "SystemUser"
        End If

        ' Construct a GenericIdentity object based on the current Windows
        ' identity name and authentication type.
        Dim authenticationType As String = windowsIdentity.AuthenticationType
        Dim userName As String = windowsIdentity.Name
        Dim genericIdentity As New GenericIdentity(userName, authenticationType)

        ' Construct a GenericPrincipal object based on the generic identity
        ' and custom roles for the user.
        Dim genericPrincipal As New GenericPrincipal(genericIdentity, roles)

        Return genericPrincipal

    End Function 'GetGenericPrincipal

    Public Shared Sub Main()
        ' Retrieve a GenericPrincipal that is based on the current user's
        ' WindowsIdentity.
        Dim genericPrincipal As GenericPrincipal = GetGenericPrincipal()

        ' Retrieve the generic identity of the GenericPrincipal object.
        Dim principalIdentity As GenericIdentity = CType(genericPrincipal.Identity, GenericIdentity)

        ' Display the identity name and authentication type.
        If principalIdentity.IsAuthenticated Then
            Console.WriteLine(principalIdentity.Name)
            Console.WriteLine("Type:" + principalIdentity.AuthenticationType)
        End If

        ' Verify that the generic principal has been assigned the
        ' NetworkUser role.
        If genericPrincipal.IsInRole("NetworkUser") Then
            Console.WriteLine("User belongs to the NetworkUser role.")
        End If

        Thread.CurrentPrincipal = genericPrincipal

    End Sub



End Class
```

Consulte también

- System.Security.Permissions.SecurityPermission
- Objetos Principal e Identity

Suplantar y revertir

02/07/2020 • 4 minutes to read • Edit Online

En ocasiones es posible que deba obtener un token de cuenta de Windows para suplantar una cuenta de Windows. Por ejemplo, la aplicación basada en ASP.NET podría tener que actuar en nombre de varios usuarios en momentos distintos. La aplicación podría aceptar un token que represente un administrador de Internet Information Services (IIS), suplantar al usuario, realizar una operación y revertir a la identidad anterior. A continuación, podría aceptar un token de IIS que represente a un usuario con menos derechos, realizar alguna operación y revertir de nuevo.

En situaciones donde la aplicación deba suplantar una cuenta de Windows que IIS no haya asociado al subproceso actual, debe recuperar el token de esa cuenta y usarlo para activar la cuenta. Para ello, debe realizar las siguientes tareas:

1. Recupere un token de cuenta de un usuario concreto haciendo una llamada al método **LogonUser** no administrado. Este método no está en la biblioteca de clases base de .NET Framework, pero se encuentra en el archivo **advapi32.dll** no administrado. El acceso a métodos en código no administrado es una operación avanzada y queda fuera del ámbito de este contenido. Para más información, consulte Interoperating with Unmanaged Code (Interoperar con código no administrado) Para más información sobre el método **LogonUser** y el archivo **advapi32.dll**, consulte la documentación de Platform SDK.
2. Cree una nueva instancia de la clase **WindowsIdentity**, pasando el token. En el siguiente código se muestra esta llamada, donde `hToken` representa un token de Windows.

   ```
   WindowsIdentity impersonatedIdentity = new WindowsIdentity(hToken);
   ```

   ```
   Dim impersonatedIdentity As New WindowsIdentity(hToken)
   ```

3. Comience la suplantación con la creación de una nueva instancia de la clase WindowsImpersonationContext e inicialícela con el método WindowsIdentity.Impersonate de la clase inicializada, como se muestra en el código siguiente.

   ```
   WindowsImpersonationContext myImpersonation = impersonatedIdentity.Impersonate();
   ```

   ```
   WindowsImpersonationContext myImpersonation = impersonatedIdentity.Impersonate()
   ```

4. Cuando ya no necesite suplantar, llame al método WindowsImpersonationContext.Undo para revertir la suplantación, como se muestra en el código siguiente.

   ```
   myImpersonation.Undo();
   ```

   ```
   myImpersonation.Undo()
   ```

Si el código de confianza ya ha asociado un WindowsPrincipal objeto al subproceso, puede llamar al método de instancia **Impersonate**, que no toma un token de cuenta. Tenga en cuenta que esto solo es útil cuando el objeto **WindowsPrincipal** del subproceso representa a un usuario que no es en el que se está ejecutando el proceso. Por ejemplo, podría encontrarse con esta situación si usa ASP.NET con la autenticación de Windows activada y la

suplantación desactivada. En este caso, el proceso se ejecuta en una cuenta configurada en Internet Information Services (IIS) mientras la entidad de seguridad actual representa al usuario de Windows que está accediendo a la página.

Tenga en cuenta que ni **Impersonate** ni **Undo** cambian el objeto **principal** (IPrincipal) asociado al contexto de llamada actual. En su lugar, la suplantación y la reversión cambian el token asociado con el proceso de sistema operativo actual.

Consulte también

- WindowsIdentity
- WindowsImpersonationContext
- Objetos Principal e Identity
- Interoperar con código no administrado

Modelo de criptografía de .NET Framework

02/07/2020 • 6 minutes to read • Edit Online

.NET Framework proporciona implementaciones de numerosos algoritmos criptográficos estándar. Estos algoritmos son fáciles de usar y tienen las propiedades predeterminadas más seguras. Además, el modelo de criptografía de .NET Framework de herencia de objetos, diseño de secuencias y configuración es muy extensible.

Herencia de objetos

El sistema de seguridad de .NET Framework implementa un patrón extensible de herencia de clases derivadas. La jerarquía es la siguiente:

- Clase de tipo de algoritmo, como SymmetricAlgorithm, AsymmetricAlgorithm o HashAlgorithm. Este nivel es abstracto.
- Clase de algoritmo que hereda de una clase de tipo de algoritmo, como, por ejemplo, Aes, RC2 o ECDiffieHellman. Este nivel es abstracto.
- Implementación de una clase de algoritmo que hereda de una clase de algoritmo, como, por ejemplo, AesManaged, RC2CryptoServiceProvider o ECDiffieHellmanCng. Este nivel está completamente implementado.

Si se usa este modelo de clases derivadas, es fácil agregar un nuevo algoritmo o una nueva implementación de un algoritmo existente. Por ejemplo, para crear un nuevo algoritmo de clave pública, heredaría de la clase AsymmetricAlgorithm. Para crear una nueva implementación de un algoritmo específico, crearía una clase derivada no abstracta de ese algoritmo.

Cómo se implementan los algoritmos en .NET Framework

Como ejemplo de las distintas implementaciones disponibles de un algoritmo, considere los algoritmos simétricos. La base de todos los algoritmos simétricos es SymmetricAlgorithm, que heredan los siguientes algoritmos:

- Aes
- DES
- RC2
- Rijndael
- TripleDES

Dos clases heredan Aes: AesCryptoServiceProvider y AesManaged. La clase AesCryptoServiceProvider es un contenedor que envuelve la implementación de Aes de la API de criptografía de Windows (CAPI), mientras que la clase AesManaged está escrita completamente en código administrado. También hay un tercer tipo de implementación, Cryptography Next Generation (CNG), además de las implementaciones administradas y de CAPI. Un ejemplo de un algoritmo CNG es ECDiffieHellmanCng. Los algoritmos CNG se encuentran disponibles en Windows Vista y versiones posteriores.

Puede elegir qué implementación es mejor para usted. Las implementaciones administradas están disponibles en todas las plataformas que admiten .NET Framework. Las implementaciones de CAPI están disponibles en sistemas operativos anteriores y ya no se desarrollan. CNG es la implementación más reciente en la que se llevará a cabo el nuevo desarrollo. Sin embargo, las implementaciones administradas no disponen del certificado de Estándares de procesamiento de información federal (FIPS) y pueden ser más lentas que las clases contenedoras.

Diseño de secuencias

El Common Language Runtime usa un diseño orientado a secuencias para implementar algoritmos simétricos y algoritmos hash. El núcleo de este diseño es la clase CryptoStream, que se deriva de la clase Stream. Los objetos criptográficos basados en secuencias admiten una interfaz estándar única (`CryptoStream`) para controlar la parte de transferencia de datos del objeto. Como todos los objetos se crean en una interfaz estándar, puede encadenar varios objetos (como un objeto hash seguido de un objeto de cifrado) y puede realizar diversas operaciones en los datos sin necesidad de un almacenamiento intermedio para ellos. El modelo de transmisión por secuencias también permite crear objetos a partir de objetos más pequeños. Por ejemplo, un algoritmo combinado de cifrado y hash puede verse como un solo objeto de secuencia, aunque este objeto podría generarse a partir de un conjunto de objetos de secuencia.

Configuración criptográfica

La configuración criptográfica le permite resolver una implementación específica de un algoritmo a un nombre de algoritmo, lo que permite la extensibilidad de las clases de criptografía de .NET Framework. Puede agregar su propia implementación de hardware o software de un algoritmo y asignar la implementación al nombre del algoritmo que quiera. Si un algoritmo no se especifica en el archivo de configuración, se usa la configuración predeterminada. Para obtener más información sobre la configuración criptográfica, vea configurar las clases de criptografía.

Elegir un algoritmo

Puede seleccionar un algoritmo por diferentes motivos: por ejemplo, para proteger la integridad de los datos, para proteger la privacidad de los datos o para generar una clave. Los algoritmos simétricos y hash están diseñados para proteger los datos por motivos de integridad (impedir los cambios) o por motivos de privacidad (impedir la visualización). Los algoritmos hash se usan principalmente para proteger la integridad de los datos.

Esta es una lista de los algoritmos recomendados en función de la aplicación:

- Privacidad de los datos:
  - Aes
- Integridad de los datos:
  - HMACSHA256
  - HMACSHA512
- Firma digital:
  - ECDsa
  - RSA
- Intercambio de claves:
  - ECDiffieHellman
  - RSA
- Generación de números aleatorios:
  - RNGCryptoServiceProvider
- Generar una clave a partir de una contraseña:
  - Rfc2898DeriveBytes

Vea también

- Servicios criptográficos
- Protocolos de criptografía, algoritmos y código fuente aplicados en C, de Bruce Schneier

Servicios criptográficos

02/07/2020 • 37 minutes to read • Edit Online

Las redes públicas como Internet no proporcionan un medio de comunicación segura entre entidades. La comunicación en esas redes es susceptible de que terceras personas, sin autorización, tengan acceso a ella o la modifiquen. La criptografía ayuda a proteger los datos para que no puedan ser vistos, proporciona mecanismos para la detección de datos modificados y facilita un medio de comunicación seguro en canales que, de otra forma, no serían seguros. Por ejemplo, los datos pueden cifrarse con un algoritmo criptográfico y transmitirse en un estado cifrado a una tercera persona, que posteriormente los descifrará. Si un tercero intercepta los datos cifrados, le resultará difícil descifrarlos.

En .NET Framework, las clases del espacio de nombres System.Security.Cryptography se ocupan de administrar muchos de los detalles de criptografía. Algunas son contenedores de la interfaz de programación de aplicaciones criptográficas (CryptoAPI) no administrada de Microsoft, mientras que otras son meramente implementaciones administradas. No necesita ser un experto en criptografía para utilizar estas clases. Cuando crea una nueva instancia de una de las clases de algoritmos de cifrado, se generan automáticamente claves para facilitar el uso y las propiedades predeterminadas son lo más seguras posible.

En esta información general se proporciona una sinopsis de los métodos y las prácticas de cifrado compatibles con el .NET Framework, incluidos los manifiestos de ClickOnce, Suite B y la compatibilidad de Cryptography Next Generation (CNG) introducida en el .NET Framework 3,5.

Para obtener más información sobre la criptografía y sobre los servicios, componentes y herramientas de Microsoft que permiten agregar seguridad criptográfica a las aplicaciones, vea la sección Seguridad de Desarrollo para Win32 y COM de esta documentación.

Primitivos criptográficos

En una situación típica donde se usa la criptografía, dos partes (Alicia y Roberto) se comunican a través de un canal que no es seguro. Ambos desean garantizar que su comunicación no la comprenda nadie que pueda estar escuchando. Además, como Alicia y Roberto se encuentran en ubicaciones remotas, Alicia quiere asegurarse de que la información que recibe de Roberto no la modificó nadie durante la transmisión. Por último, debe asegurarse también de que la información proviene realmente de Roberto y no de alguien que lo suplanta.

La criptografía se utiliza para lograr los objetivos siguientes:

- Confidencialidad: para ayudar a proteger la identidad de un usuario o evitar que se lean sus datos.
- Integridad de los datos: para ayudar a evitar su alteración.
- Autenticación: para garantizar que los datos provienen de una parte concreta.
- Sin rechazo: para evitar que una determinada parte niegue que envió un mensaje.

Para alcanzar estos objetivos, se puede usar una combinación de algoritmos y prácticas conocidas como primitivas criptográficas para crear un esquema criptográfico. En la tabla siguiente se enumeran las primitivas criptográficas y su uso.

PRIMITIVA CRIPTOGRÁFICA	USO
Cifrado de clave secreta (criptografía simétrica)	Realiza la transformación de los datos para impedir que terceros los lean. Este tipo de cifrado utiliza una clave secreta compartida para cifrar y descifrar los datos.

PRIMITIVA CRIPTOGRÁFICA	USO
Cifrado de clave pública (criptografía asimétrica)	Realiza la transformación de los datos para impedir que terceros los lean. Este tipo de cifrado utiliza un par de claves pública y privada para cifrar y descifrar los datos.
Firmas criptográficas	Ayuda 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 dos bytes distinta no será el mismo.

Cifrado de clave secreta

Los algoritmos de cifrado de clave secreta utilizan una clave secreta única para cifrar y descifrar datos. Debe asegurarse de que agentes no autorizados no obtienen acceso a la clave, ya que cualquier persona que tenga la clave podría utilizarla para descifrar los datos o cifrar sus propios datos y alegar que provienen de usted.

El cifrado de clave secreta también se denomina cifrado simétrico puesto que se utiliza la misma clave para el cifrado y el descifrado. Los algoritmos de cifrado de clave secreta son muy rápidos (comparados con los de clave pública) y resultan adecuados para realizar transformaciones criptográficas en grandes flujos de datos. Los algoritmos de cifrado asimétricos, como RSA, están limitados matemáticamente respecto al volumen de datos que pueden cifrar. Los algoritmos de cifrado simétricos no suelen tener estos problemas.

El tipo de algoritmo de clave secreta denominado cifrado de bloques se utiliza para cifrar un bloque de datos cada vez. Los cifrados de bloques, como Estándar de cifrado de datos (DES), TripleDES y Estándar de cifrado avanzado (CA), transforman criptográficamente un bloque de entrada de *n* bytes en un bloque de salida de bytes cifrados. Si desea cifrar o descifrar una secuencia de bytes, debe hacerlo bloque a bloque. Dado que *n* es pequeño (8 bytes para DES y TripleDES y 16 bytes [valor predeterminado], 24 bytes o 32 bytes para AES), los valores mayores que *n* deben cifrarse bloque a bloque. Los valores que son menores que *n* tienen que ampliarse hasta *n* para que se puedan procesar.

Una forma sencilla de cifrado de bloques es el modo Electronic Codebook (ECB). El modo ECB no se considera un modo seguro porque no utiliza un vector de inicialización para iniciar el primer bloque de texto simple. Para una clave secreta *k*determinada, un cifrado de bloques simple que no utiliza un vector de inicialización codificará el mismo bloque de entrada de texto simple en el mismo bloque de salida de texto cifrado. Por tanto, si hay bloques duplicados dentro la secuencia de texto simple de entrada, habrá bloques duplicados en la secuencia de texto cifrado de salida. Estos bloques de salida duplicados podrían alertar a los usuarios sin autorización sobre la posibilidad de que se haya utilizado un cifrado débil en los algoritmos y los posibles modos de ataque. El modo de cifrado ECB es por tanto bastante vulnerable al análisis, y en última instancia, a la detección de claves.

Las clases de cifrado de bloques que se proporcionan en la biblioteca de clases base utilizan un modo de encadenamiento predeterminado denominado encadenamiento de bloques de cifrado (CBC), aunque este valor puede cambiarse, si así se desea.

El cifrado CBC subsana los problemas asociados con el cifrado ECB utilizando un vector de inicialización (IV) para cifrar el primer bloque de texto simple. Cada uno de los bloques de texto simple siguientes se somete a una operación OR exclusiva bit a bit (`XOR`) con el bloque de texto cifrado anterior antes de cifrarse. Cada bloque de

texto cifrado depende por tanto de todos los bloques anteriores. Cuando se utiliza este sistema, los encabezados de los mensajes comunes que un usuario no autorizado podría conocer no pueden utilizarse para aplicar técnicas de ingeniería inversa en una clave.

Un modo de comprometer los datos cifrados con un cifrado CBC consiste en realizar una búsqueda exhaustiva de todas las claves posibles. En función del tamaño de la clave utilizada para realizar el cifrado, este tipo de búsqueda llevará mucho tiempo aunque se utilicen los equipos más rápidos, y, por lo tanto, no es factible. Los tamaños de clave mayores son más difíciles de descifrar. Aunque el cifrado no garantiza totalmente que un adversario no recupere los datos cifrados, aumenta considerablemente la dificultad. Si se tardan tres meses en realizar una búsqueda exhaustiva para recuperar datos que solo son relevantes durante varios días, este método de búsqueda resulta poco práctico.

La desventaja del cifrado de clave secreta es que presupone que dos partes acuerdan una clave y un vector de inicialización, y que se comunican sus valores. El vector de inicialización no se considera secreto y se transmite como texto simple con el mensaje. Sin embargo, la clave debe mantenerse en secreto a salvo de usuarios no autorizados. Debido a estos problemas, el cifrado de clave secreta a menudo se utiliza junto con el cifrado de clave pública para comunicar en privado los valores de la clave y el vector de inicialización.

Supongamos que Alicia y Roberto son dos personas que desean comunicarse a través de un canal que no es seguro. Ellos podrían utilizar el cifrado de clave secreta del modo siguiente: Alicia y Roberto están de acuerdo en utilizar un algoritmo determinado (AES, por ejemplo) con una clave y un vector de inicialización determinados. Alice redacta un mensaje y crea una secuencia de red (quizás una canalización con nombre o correo electrónico de red) en la que se va a enviar el mensaje. A continuación, cifra el texto utilizando la clave y el vector de inicialización y envía el mensaje cifrado y el vector de inicialización a Roberto a través de intranet. Roberto recibe el texto cifrado y lo descifra utilizando el vector de inicialización y la clave acordada anteriormente. Si se intercepta la transmisión, el interceptor no puede recuperar el mensaje original porque no conoce la clave. En este caso, solo debe mantenerse en secreto la clave. En un ejemplo real, Alicia o Roberto genera una clave secreta y utiliza el cifrado (asimétrico) de clave pública para transferir la clave (simétrica) secreta a la otra parte. Para obtener más información sobre el cifrado de clave pública, vea la sección siguiente.

El .NET Framework proporciona las siguientes clases que implementan algoritmos de cifrado de clave secreta:

- AesManaged(introducido en el .NET Framework 3,5).
- DESCryptoServiceProvider.
- HMACSHA1 (Técnicamente, se trata de un algoritmo de clave secreta, ya que representa un código de autenticación de mensajes que se calcula utilizando una función hash criptográfica junto con una clave secreta. Vea Valores hashmás adelante en este mismo tema.)
- RC2CryptoServiceProvider.
- RijndaelManaged.
- TripleDESCryptoServiceProvider.

Cifrado de clave pública

El cifrado de clave pública utiliza una clave privada que debe mantenerse en secreto y a salvo de los usuarios no autorizados, así como una clave pública que puede comunicarse. La clave pública y la clave privada están vinculadas matemáticamente; los datos cifrados con la clave pública solo pueden descifrarse con la clave privada y los datos firmados con la clave privada solo pueden comprobarse con la clave pública. La clave pública está a disposición de cualquier persona y se utiliza para cifrar los datos que se envían a quien guarda la clave privada. Los algoritmos criptográficos de clave pública también se conocen como algoritmos asimétricos porque se necesita una clave para cifrar los datos y otra para descifrarlos. Una regla criptográfica básica prohíbe la reutilización de claves; además, las dos claves deben ser únicas en cada sesión de comunicación. Sin embargo, en la práctica, las claves asimétricas suelen durar bastante tiempo.

Dos personas (Alicia y Roberto) podrían utilizar el cifrado de clave pública del modo siguiente: en primer lugar, Alicia podría generar un par de claves pública y privada. Si Roberto desea enviar a Alicia un mensaje cifrado, le pide la clave pública. Alicia envía a Roberto su clave pública a través de una red que no es segura y Roberto utiliza esta clave para cifrar un mensaje. Roberto envía el mensaje cifrado a Alicia y ésta lo descifra utilizando su clave privada. Si Roberto recibiera la clave de Alicia a través de un canal que no es seguro, como una red pública, Roberto estaría expuesto a un ataque del tipo "Man in the middle". Por consiguiente, Roberto debe comprobar con Alicia que tiene una copia correcta de su clave pública.

Durante la transmisión de la clave pública de Alicia, un agente no autorizado podría interceptarla. Además, el mismo agente podría interceptar el mensaje cifrado de Roberto. No obstante, el agente no puede descifrar el mensaje con la clave pública. El mensaje solo puede descifrarse con la clave privada de Alicia, que no se ha transmitido. Alicia no utiliza su clave privada para cifrar un mensaje de respuesta a Roberto, porque cualquiera que tenga la clave pública podría descifrarlo. Si Alicia desea enviar un mensaje a Roberto, le pide su clave pública y cifra su mensaje con ella. Seguidamente, Roberto descifra el mensaje utilizando su clave privada asociada.

En este escenario, Alicia y Roberto utilizan un cifrado de clave pública (asimétrica) para transferir una clave secreta (simétrica) y utilizan el cifrado de clave secreta para el resto de la sesión.

En la lista siguiente se incluyen comparaciones entre algoritmos criptográficos de clave pública y de clave secreta:

- Los algoritmos criptográficos de clave pública utilizan un tamaño de búfer fijo mientras que los de clave secreta usan un búfer de longitud variable.
- Los algoritmos de clave pública no se pueden usar para encadenar datos juntos en secuencias como sucede con los de clave secreta, ya que solo pueden cifrarse pequeñas cantidades de datos. Por lo tanto, las operaciones asimétricas no utilizan el mismo modelo de streaming que las simétricas.
- El cifrado de claves públicas tiene un espacio de claves (el intervalo de valores posibles de la clave) mucho mayor que el cifrado de claves secretas. Por tanto, el cifrado de claves públicas es menos vulnerable a los ataques exhaustivos que prueban cada clave posible.
- Al no tener que estar protegidas, las claves públicas resultan fáciles de distribuir, siempre que exista algún mecanismo para comprobar la identidad del remitente.
- Algunos algoritmos de clave pública (como RSA y DSA, aunque no Diffie-Hellman) se pueden utilizar para crear firmas digitales con el fin de comprobar la identidad del remitente de los datos.
- Los algoritmos de clave pública son muy lentos comparados con los de clave secreta y no están diseñados para cifrar grandes cantidades de datos. Son útiles solo para transferir cantidades muy pequeñas de información. Normalmente, el cifrado de clave pública se utiliza para cifrar la clave y el vector de inicialización que serán utilizados por un algoritmo de clave secreta. Después de transferir la clave y el vector de inicialización, el cifrado de clave secreta se utiliza para el resto de la sesión.

El .NET Framework proporciona las siguientes clases que implementan algoritmos de cifrado de clave pública:

- DSACryptoServiceProvider
- RSACryptoServiceProvider
- ECDiffieHellman (clase base)
- ECDiffieHellmanCng
- ECDiffieHellmanCngPublicKey (clase base)
- ECDiffieHellmanKeyDerivationFunction (clase base)
- ECDsaCng

RSA permite tanto el cifrado como el uso de firmas, pero DSA solo se puede utilizar para firmar y Diffie-Hellman

solo se puede utilizar para la generación de claves. En general, los algoritmos de clave pública tienen una aplicación más limitada que los algoritmos de clave privada.

firmas digitales

Los algoritmos de clave pública también se pueden usar para formar firmas digitales. Las firmas digitales autentican la identidad de un remitente (si se fía de la clave pública de éste) y ayudan a proteger la integridad de los datos. Con una clave pública generada por Alicia, el remitente de los datos de Alicia puede comprobar que ésta los envió si compara la firma digital de los datos de Alicia y la clave pública de ésta.

Para utilizar criptografía de clave pública con el objeto de firmar digitalmente un mensaje, Alicia aplica primero un algoritmo hash al mensaje para crear una síntesis del mismo. La síntesis del mensaje es una representación compacta y única de los datos. Seguidamente, Alicia cifra la síntesis del mensaje con su clave privada para crear su firma personal. Después de recibir el mensaje y la firma, Roberto descifra esta última utilizando la clave pública de Alicia para recuperar la síntesis del mensaje y envía éste de forma aleatoria mediante el mismo algoritmo hash que utilizó Alicia. Si la síntesis del mensaje que calcula Roberto coincide exactamente con la que ha recibido de Alicia, Roberto puede estar seguro de que el mensaje provino del poseedor de la clave privada y de que no se han modificado los datos. Si Roberto confía en que Alicia es la propietaria de la clave privada, sabe que el mensaje proviene de Alicia.

> **NOTE**
> Cualquiera puede comprobar una firma, ya que la clave pública del remitente es de dominio público y normalmente se incluye en el formato de firma digital. Este método no mantiene la confidencialidad del mensaje; para que sea secreto, también se debe cifrar.

El .NET Framework proporciona las siguientes clases que implementan algoritmos de firma digital:

- DSACryptoServiceProvider
- RSACryptoServiceProvider
- ECDsa (clase base)
- ECDsaCng

Valores hash

Los algoritmos hash asignan valores binarios de longitud arbitraria a valores binarios más pequeños de longitud fija, que se denominan valores hash. Un valor hash es una representación numérica de un segmento de datos. Si aplica un algoritmo hash a un párrafo de texto simple y cambia solo una letra del párrafo, el valor hash subsiguiente producirá un valor distinto. Si el valor hash es criptográficamente seguro, su valor cambiará significativamente. Por ejemplo, si se cambia únicamente un solo bit de un mensaje, una función hash segura puede generar un resultado que difiera en un 50 por ciento. Muchos valores de entrada pueden aplicar un algoritmo hash al mismo valor de salida. Sin embargo, técnicamente es poco factible encontrar dos entradas distintas cuyo valor hash sea el mismo.

Dos partes (Alicia y Roberto) podrían utilizar una función hash para asegurar la integridad de los mensajes. Podrían seleccionar un algoritmo hash para firmar sus mensajes. Alicia escribiría un mensaje y, a continuación, crearía un valor hash de ese mensaje utilizando el algoritmo seleccionado. Después seguiría uno de los métodos siguientes:

- Alicia enviaría a Roberto el mensaje de texto simple y el mensaje al que aplicó el algoritmo hash (firma digital). Roberto recibiría el mensaje y le aplicaría el algoritmo hash. A continuación, compararía su valor hash con el valor hash que recibió de Alicia. Si los dos valores hash son idénticos, el mensaje no se ha

modificado. Si los valores no son idénticos, el mensaje se modificó después de que Alicia lo escribiera.

Desgraciadamente, este método no determina la autenticidad del remitente. Cualquiera puede suplantar a Alicia y enviar un mensaje a Roberto. Pueden utilizar el mismo algoritmo hash para firmar su mensaje, y todo lo que Roberto podría determinar es que el mensaje coincide con su firma. Esta es una forma de ataque de tipo "Man in the middle". Para obtener más información, vea ejemplo de comunicación segura de Cryptography Next Generation (CNG).

- Alicia envía el mensaje de texto simple a Roberto a través de un canal público que no es seguro. Envía a Roberto el mensaje al que aplicó el algoritmo hash a través de un canal privado seguro. Roberto recibe el mensaje de texto simple, le aplica un algoritmo hash y compara el valor hash con el valor hash que se intercambió de forma privada. Si los valores hash coinciden, Roberto sabe dos cosas:
  - que el mensaje no se alteró.
  - que el remitente del mensaje (Alicia) es auténtico.

  Para que este sistema funcione, Alicia debe ocultar el valor hash original a todos excepto a Roberto.
- Alicia envía el mensaje de texto simple a Roberto a través de un canal público que no es seguro y publica el mensaje al que aplicó el algoritmo hash en su sitio web para que esté visible públicamente.

  Este método evita que se manipule el mensaje, ya que impide que nadie modifique el valor hash. Aunque cualquier persona puede leer el mensaje y su valor hash, este valor hash únicamente lo puede modificar Alicia. Un atacante que desee suplantar a Alicia necesitaría tener acceso al sitio web de Alicia.

Ninguno de los métodos anteriores evitará que alguien lea los mensajes de Alicia, ya que se transmiten en texto simple. Para conseguir una seguridad total, normalmente se necesitarán firmas digitales (firma del mensaje) y mecanismos de cifrado.

El .NET Framework proporciona las siguientes clases que implementan algoritmos hash:

- HMACSHA1.
- MACTripleDES.
- MD5CryptoServiceProvider.
- RIPEMD160.
- SHA1Managed.
- SHA256Managed.
- SHA384Managed.
- SHA512Managed.
- Variaciones HMAC de todos los algoritmos SHA (algoritmo hash seguro), MD5 (Message Digest 5) y RIPEMD-160.
- Implementaciones de CryptoServiceProvider (contenedores del código administrado) de todos los algoritmos SHA.
- Implementaciones de Criptografía de próxima generación (CNG) de todos los algoritmos SHA y MD5.

NOTE

En 1996 se detectaron defectos de diseño en MD5 y en su lugar se recomendó SHA-1. En 2004 se detectaron nuevos defectos y el algoritmo MD5 ya no se consideró seguro. También se ha descubierto que el algoritmo SHA-1 no es seguro y ahora se recomienda utilizar el algoritmo SHA 2 en su lugar.

generación de números aleatorios

La generación de números aleatorios es propia de muchas operaciones criptográficas. Por ejemplo, las claves criptográficas deben ser lo más aleatorias posible para que no se puedan reproducir. Los generadores de números aleatorios criptográficos deben generar resultados que, mediante cálculos, no se pueden predecir con una probabilidad mayor del cincuenta por ciento. Por tanto, cualquier método para predecir el siguiente bit del resultado no debe ser más eficaz que el cálculo aleatorio. Las clases de la .NET Framework utilizan generadores de números aleatorios para generar claves criptográficas.

La clase RNGCryptoServiceProvider es una implementación de un algoritmo generador de números aleatorios.

Manifiestos de ClickOnce

En el .NET Framework 3,5, las siguientes clases de criptografía permiten obtener y comprobar información sobre las firmas de manifiesto para las aplicaciones que se implementan mediante la tecnología ClickOnce:

- La clase ManifestSignatureInformation obtiene información sobre una firma de manifiesto cuando se utiliza la sobrecarga de su método VerifySignature .
- Puede utilizar la enumeración ManifestKinds para especificar los manifiestos que se van a comprobar. El resultado de la comprobación es uno de los valores de la enumeración SignatureVerificationResult .
- La clase ManifestSignatureInformationCollection proporciona una colección de objetos ManifestSignatureInformation de solo lectura de las firmas comprobadas.

Además, las clases siguientes proporcionan información de firma específica:

- StrongNameSignatureInformation contiene información sobre la firma de nombre seguro de un manifiesto.
- AuthenticodeSignatureInformation representa la información sobre la firma Authenticode de un manifiesto.
- TimestampInformation contiene información sobre la marca de tiempo de una firma Authenticode.
- TrustStatus proporciona un mecanismo sencillo para comprobar si se confía en una firma Authenticode.

Compatibilidad con Suite B

El .NET Framework 3,5 admite el conjunto Suite B de algoritmos criptográficos publicados por National Security Agency (NSA). Para obtener más información sobre Suite B, vea la hoja informativa sobre la criptografía de Suite B de la NSA.

Se incluyen los siguientes algoritmos:

- Algoritmo del Estándar de cifrado avanzado (AES) con tamaños clave de cifrado de 128, 192 y 256 bits.
- Algoritmos hash seguros SHA-1, SHA-256, SHA-384 y SHA-512 para crear valores hash. (Los tres últimos normalmente están agrupados juntos y se les conoce como SHA 2).
- Algoritmo de firma digital de curva elíptica (ECDSA), que utiliza curvas de módulos primos de 256, 384 y

521 bits. En la documentación de NSA se definen explícitamente estas curvas y se denominan P-256, P-384 y P-521. Este algoritmo lo proporciona la clase ECDsaCng . Permite firmar con una clave privada y comprobar la firma con una clave pública.

- Algoritmo de Diffie-Hellman de curva elíptica (ECDH), que utiliza curvas de módulos primos de 256, 384 y 521 bits para el acuerdo confidencial e intercambio de claves. Este algoritmo lo proporciona la clase ECDiffieHellmanCng .

Los contenedores de código administrado para las implementaciones certificadas del Estándar federal de procesamiento de información (FIPS) de AES, SHA-256, SHA-384 y SHA-512 están disponibles en las nuevas clases AesCryptoServiceProvider, SHA256CryptoServiceProvider, SHA384CryptoServiceProvidery SHA512CryptoServiceProvider .

Clases de criptografía de próxima generación (CNG)

Las clases de Criptografía de próxima generación (CNG) proporcionan un contenedor administrado en torno a funciones CNG nativas. (CNG es el sustituto de CryptoAPI). Estas clases tienen "CNG" como parte de sus nombres. La clase contenedora de claves CngKey es fundamental en estas clases contenedoras CNG, pues abstrae el almacenamiento y el uso de claves CNG. Esta clase permite almacenar de forma segura un par de claves o una clave pública y hacer referencia a ella utilizando un nombre de cadena simple. La clase de firma ECDsaCng y la clase de cifrado ECDiffieHellmanCng basadas en curvas elípticas pueden utilizar los objetos CngKey .

La clase CngKey se utiliza en otras numerosas operaciones, entre las que se incluyen la apertura, creación, eliminación y exportación de claves. También proporciona acceso al identificador de clave subyacente que se va a utilizar en las llamadas directas a las funciones nativas.

El .NET Framework 3,5 también incluye una serie de clases CNG compatibles, como las siguientes:

- CngProvider mantiene un proveedor de almacenamiento de claves.
- CngAlgorithm mantiene un algoritmo CNG.
- CngProperty mantiene las propiedades clave que se utilizan con frecuencia.

Temas relacionados

TITLE	DESCRIPCIÓN
Modelo de criptografía	Describe la forma de implementar la criptografía en la biblioteca de clases base.
Tutorial: Crear una aplicación criptográfica	Explica el cifrado básico y las tareas de descifrado.
Configurar clases de criptografía	Describe la forma de asignar nombres de algoritmo a clases criptográficas e identificadores de objetos a un algoritmo criptográfico.

Generar claves para cifrado y descifrado

02/07/2020 • 7 minutes to read • Edit Online

La creación y administración de claves es una parte importante del proceso criptográfico. Los algoritmos simétricos requieren la creación de una clave y un IV (Initialization Vector, vector de inicialización). La clave debe mantenerse en secreto y a salvo de quienes no deban descifrar los datos. No es necesario que el vector de inicialización sea secreto, pero debe cambiarse para cada sesión. Los algoritmos asimétricos requieren la creación de una clave pública y una clave privada. La clave pública puede revelarse a cualquiera, mientras que la privada debe conocerla sólo la parte que descifrará los datos cifrados con la clave pública. En esta sección se describe cómo generar y administrar claves para algoritmos simétricos y asimétricos.

Claves simétricas

Las clases de cifrado simétrico que proporciona .NET Framework requieren una clave y un nuevo vector de inicialización (IV) para cifrar y descifrar datos. Siempre que se crea una nueva instancia de una de las clases criptográficas simétricas administradas mediante el constructor sin parámetros, se crean automáticamente una nueva clave y un IV. Cualquier persona a la que permita descifrar sus datos debe poseer la misma clave y el mismo IV, y utilizar el mismo algoritmo. Normalmente, debe crearse una nueva clave y vector de inicialización para cada sesión, y ni la clave ni el vector deberían almacenarse para utilizarlos en una sesión posterior.

Para comunicar una clave simétrica y un IV a una parte remota, la clave simétrica normalmente se cifra usando cifrado asimétrico. El envío de la clave a través de una red insegura sin cifrarla resulta peligroso, ya que quien intercepte la clave y el IV podrá descifrar los datos. Para obtener más información sobre el intercambio de datos mediante el cifrado, vea Crear un esquema criptográfico.

En el ejemplo siguiente se muestra la creación de una nueva instancia de la clase TripleDESCryptoServiceProvider que implementa el algoritmo TripleDES.

```
Dim tdes As TripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider()
```

```
TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider();
```

Cuando se ejecuta el código anterior, se genera una nueva clave y un IV, y se colocan en las propiedades **Key** e **IV** , respectivamente.

En ocasiones tendrá que generar varias claves. En este caso, puede crear una nueva instancia de una clase que implemente un algoritmo simétrico y después crear una nueva clave e IV mediante una llamada a los métodos **GenerateKey** y **GenerateIV** . En el ejemplo de código siguiente se muestra cómo crear nuevas claves y IV después de realizar una nueva instancia de la clase criptográfica simétrica.

```
Dim tdes As TripleDESCryptoServiceProvider = new TripleDESCryptoServiceProvider()
tdes.GenerateIV()
tdes.GenerateKey()
```

```
TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider();
tdes.GenerateIV();
tdes.GenerateKey();
```

Cuando se ejecuta el código anterior, se generan una clave y un IV al crear la nueva instancia de **TripleDESCryptoServiceProvider** . Se crean otra clave e IV cuando se llama a los métodos **GenerateKey** y **GenerateIV** .

Claves asimétricas

.NET Framework proporciona las clases RSACryptoServiceProvider y DSACryptoServiceProvider para el cifrado asimétrico. Estas clases crean un par de claves pública y privada cuando se utiliza el constructor sin parámetros para crear una nueva instancia. Las claves asimétricas pueden almacenarse para utilizarse en sesiones múltiples o bien generarse para una sesión únicamente. Aunque la clave pública puede ponerse a disposición general, la clave privada debe guardarse bien.

Un par de claves pública y privada se genera siempre que se crea una nueva instancia de una clase de algoritmo asimétrico. Una vez creada una nueva instancia de la clase, la información de la clave puede extraerse mediante uno de estos dos métodos:

- El método ToXmlString , que devuelve una representación XML de la información de la clave.
- Método ExportParameters que devuelve una estructura RSAParameters que contiene la información de la clave.

Ambos métodos aceptan un valor booleano que indica si hay que devolver sólo la información de la clave pública o si se devuelve también la correspondiente a la clave privada. El valor de una clase **RSACryptoServiceProvider** puede inicializarse con una estructura **RSAParameters** mediante el método ImportParameters .

Las claves privadas asimétricas nunca deben almacenarse literalmente o en texto sin formato en el equipo local. Si debe almacenar una clave privada, utilice un contenedor de claves. Para más información sobre cómo almacenar una clave privada en un contenedor de claves, vea How to: Store Asymmetric Keys in a Key Container.

En el siguiente ejemplo de código se crea una nueva instancia de la clase **RSACryptoServiceProvider** , se crea un par de claves pública y privada, y se guarda la información de la clave pública en una estructura **RSAParameters** .

```
'Generate a public/private key pair.
Dim rsa as RSACryptoServiceProvider = new RSACryptoServiceProvider()
'Save the public key information to an RSAParameters structure.
Dim rsaKeyInfo As RSAParameters = rsa.ExportParameters(false)
```

```
//Generate a public/private key pair.
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
//Save the public key information to an RSAParameters structure.
RSAParameters rsaKeyInfo = rsa.ExportParameters(false);
```

Consulte también

- Cifrar datos
- Descifrar datos
- Servicios criptográficos
- Procedimiento para almacenar claves asimétricas en un contenedor de claves

Almacenar claves asimétricas en un contenedor de claves

02/07/2020 • 7 minutes to read • Edit Online

Las claves privadas asimétricas nunca deben almacenarse literalmente o en texto sin formato en el equipo local. Si necesita almacenar una clave privada, utilice un contenedor de claves. Para obtener más información acerca de los contenedores de claves, consulte Descripción de los contenedores de claves RSA de nivel de equipo y de nivel de usuario.

Crear una clave asimétrica y guardarla en un contenedor de claves

1. Cree una nueva instancia de una CspParameters clase y pase el nombre que desea que llame al contenedor de claves al CspParameters.KeyContainerName campo.
2. Cree una nueva instancia de una clase que derive de la AsymmetricAlgorithm clase (normalmente RSACryptoServiceProvider o DSACryptoServiceProvider) y pase el objeto creado previamente `CspParameters` a su constructor.

> **NOTE**
> La creación y recuperación de una clave asimétrica es una operación. Si una clave todavía no está en el contenedor, se crea antes de que se devuelva.
>
> - RSA.ToXmlString
> - DSA.ToXmlString

Eliminar la clave del contenedor de claves

1. Cree una nueva instancia de una `CspParameters` clase y pase el nombre que desea que llame al contenedor de claves al CspParameters.KeyContainerName campo.
2. Cree una nueva instancia de una clase que derive de la AsymmetricAlgorithm clase (normalmente `RSACryptoServiceProvider` o `DSACryptoServiceProvider`) y pase el objeto creado previamente `CspParameters` a su constructor.
3. Establezca la RSACryptoServiceProvider.PersistKeyInCsp propiedad o DSACryptoServiceProvider.PersistKeyInCsp de la clase que se deriva de `AsymmetricAlgorithm` en `false` (`False` en Visual Basic).
4. Llame al `Clear` método de la clase que deriva de `AsymmetricAlgorithm` . Este método libera todos los recursos de la clase y borra el contenedor de claves.

Ejemplo

En el ejemplo siguiente se muestra cómo crear una clave asimétrica, guardarla en un contenedor de claves, recuperarla posteriormente y eliminarla del contenedor.

Observe que el código de los métodos `GenKey_SaveInContainer` y `GetKeyFromContainer` es similar. Cuando se especifica un nombre de contenedor de claves para un CspParameters objeto y se pasa a un AsymmetricAlgorithm objeto con la PersistKeyInCsp propiedad o la PersistKeyInCsp propiedad establecida en `true` , el comportamiento

es el siguiente:

- Si no existe un contenedor de claves con el nombre especificado, se crea uno y la clave se conserva.
- Si no existe un contenedor de claves con el nombre especificado, la clave del contenedor se carga automáticamente en el objeto AsymmetricAlgorithm actual.

Por lo tanto, el código del `GenKey_SaveInContainer` método conserva la clave porque se ejecuta primero, mientras que el código del `GetKeyFromContainer` método carga la clave porque se ejecuta en segundo lugar.

```
Imports System
Imports System.Security.Cryptography

Public Class StoreKey

    Public Shared Sub Main()
        Try
            ' Create a key and save it in a container.
            GenKey_SaveInContainer("MyKeyContainer")

            ' Retrieve the key from the container.
            GetKeyFromContainer("MyKeyContainer")

            ' Delete the key from the container.
            DeleteKeyFromContainer("MyKeyContainer")

            ' Create a key and save it in a container.
            GenKey_SaveInContainer("MyKeyContainer")

            ' Delete the key from the container.
            DeleteKeyFromContainer("MyKeyContainer")
        Catch e As CryptographicException
            Console.WriteLine(e.Message)
        End Try
    End Sub

    Private Shared Sub GenKey_SaveInContainer(ByVal ContainerName As String)
        ' Create the CspParameters object and set the key container
        ' name used to store the RSA key pair.
        Dim parameters As New CspParameters With {
            .KeyContainerName = ContainerName
        }

        ' Create a new instance of RSACryptoServiceProvider that accesses
        ' the key container MyKeyContainerName.
        Using rsa As New RSACryptoServiceProvider(parameters)
            ' Display the key information to the console.
            Console.WriteLine($"Key added to container:  {rsa.ToXmlString(True)}")
        End Using
    End Sub

    Private Shared Sub GetKeyFromContainer(ByVal ContainerName As String)
        ' Create the CspParameters object and set the key container
        '  name used to store the RSA key pair.
        Dim parameters As New CspParameters With {
            .KeyContainerName = ContainerName
        }

        ' Create a new instance of RSACryptoServiceProvider that accesses
        ' the key container MyKeyContainerName.
        Using rsa As New RSACryptoServiceProvider(parameters)
            ' Display the key information to the console.
            Console.WriteLine($"Key retrieved from container : {rsa.ToXmlString(True)}")
        End Using
    End Sub

    Private Shared Sub DeleteKeyFromContainer(ByVal ContainerName As String)
        ' Create the CspParameters object and set the key container
```

```
        ' Create the CspParameters object and set the key container
        '  name used to store the RSA key pair.
        Dim parameters As New CspParameters With {
            .KeyContainerName = ContainerName
        }

        ' Create a new instance of RSACryptoServiceProvider that accesses
        ' the key container.
        ' Delete the key entry in the container.
        Dim rsa As New RSACryptoServiceProvider(parameters) With {
            .PersistKeyInCsp = False
        }

        ' Call Clear to release resources and delete the key from the container.
        rsa.Clear()

        Console.WriteLine("Key deleted.")
    End Sub
End Class
```

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

public class StoreKey
{
    public static void Main()
    {
        try
        {
            // Create a key and save it in a container.
            GenKey_SaveInContainer("MyKeyContainer");

            // Retrieve the key from the container.
            GetKeyFromContainer("MyKeyContainer");

            // Delete the key from the container.
            DeleteKeyFromContainer("MyKeyContainer");

            // Create a key and save it in a container.
            GenKey_SaveInContainer("MyKeyContainer");

            // Delete the key from the container.
            DeleteKeyFromContainer("MyKeyContainer");
        }
        catch (CryptographicException e)
        {
            Console.WriteLine(e.Message);
        }
    }

    private static void GenKey_SaveInContainer(string containerName)
    {
        // Create the CspParameters object and set the key container
        // name used to store the RSA key pair.
        var parameters = new CspParameters
        {
            KeyContainerName = containerName
        };

        // Create a new instance of RSACryptoServiceProvider that accesses
        // the key container MyKeyContainerName.
        using var rsa = new RSACryptoServiceProvider(parameters);

        // Display the key information to the console.
        Console.WriteLine($"Key added to container: \n  {rsa.ToXmlString(true)}");
    }

    private static void GetKeyFromContainer(string containerName)
```

```
    private static void GetKeyFromContainer(string containerName)
    {
        // Create the CspParameters object and set the key container
        // name used to store the RSA key pair.
        var parameters = new CspParameters
        {
            KeyContainerName = containerName
        };

        // Create a new instance of RSACryptoServiceProvider that accesses
        // the key container MyKeyContainerName.
        using var rsa = new RSACryptoServiceProvider(parameters);

        // Display the key information to the console.
        Console.WriteLine($"Key retrieved from container : \n {rsa.ToXmlString(true)}");
    }

    private static void DeleteKeyFromContainer(string containerName)
    {
        // Create the CspParameters object and set the key container
        // name used to store the RSA key pair.
        var parameters = new CspParameters
        {
            KeyContainerName = containerName
        };

        // Create a new instance of RSACryptoServiceProvider that accesses
        // the key container.
        using var rsa = new RSACryptoServiceProvider(parameters)
        {
            // Delete the key entry in the container.
            PersistKeyInCsp = false
        };

        // Call Clear to release resources and delete the key from the container.
        rsa.Clear();

        Console.WriteLine("Key deleted.");
    }
}
```

La salida es como sigue:

```
Key added to container:
<RSAKeyValue> Key Information A</RSAKeyValue>
Key retrieved from container :
<RSAKeyValue> Key Information A</RSAKeyValue>
Key deleted.
Key added to container:
<RSAKeyValue> Key Information B</RSAKeyValue>
Key deleted.
```

Consulte también

- Generar claves para cifrado y descifrado
- Cifrar datos
- Descifrar datos
- Servicios criptográficos

Cifrar datos

02/07/2020 • 9 minutes to read • Edit Online

El cifrado simétrico y el cifrado asimétrico se efectúan mediante procesos distintos. El cifrado simétrico se realiza en secuencias y, por tanto, resulta útil para cifrar grandes cantidades de datos. El cifrado asimétrico se realiza en un pequeño número de bytes y, por tanto, solo resulta útil para pequeñas cantidades de datos.

Cifrado simétrico

Las clases de criptografía simétrica administrada se usan con una clase de secuencia especial llamada CryptoStream que cifra los datos leídos en la secuencia. La clase **CryptoStream** se inicializa con una clase de secuencia administrada, una clase que implementa la interfaz ICryptoTransform (creada a partir de una clase que implementa un algoritmo criptográfico) y una enumeración CryptoStreamMode que describe el tipo de acceso permitido a **CryptoStream**. La clase **CryptoStream** puede inicializarse usando cualquier clase que derive de la Stream clase, incluidas FileStream , MemoryStream y NetworkStream . Mediante estas clases, puede realizar el cifrado simétrico en diversos objetos de secuencia.

En el ejemplo siguiente se muestra cómo crear una nueva instancia de la clase RijndaelManaged , que implementa el algoritmo de cifrado Rijndael, y usarla para realizar el cifrado en una clase **CryptoStream** . En este ejemplo, **CryptoStream** se inicializa con un objeto de secuencia llamado `myStream` que puede ser cualquier tipo de secuencia administrada. Se pasan la clave y el IV que se usan para el cifrado al método **CreateEncryptor** de la clase **RijndaelManaged** . En este caso, se usan la clave predeterminada y el IV generados desde `rmCrypto` . Por último, se pasa **CryptoStreamMode.Write y** , que especifica el acceso de escritura a la secuencia.

```
Dim rmCrypto As New RijndaelManaged()
Dim cryptStream As New CryptoStream(myStream, rmCrypto.CreateEncryptor(rmCrypto.Key, rmCrypto.IV),
CryptoStreamMode.Write)
```

```
RijndaelManaged rmCrypto = new RijndaelManaged();
CryptoStream cryptStream = new CryptoStream(myStream, rmCrypto.CreateEncryptor(), CryptoStreamMode.Write);
```

Después de que se ejecute este código, los datos escritos en el objeto **CryptoStream** se cifran usando el algoritmo Rijndael.

En el ejemplo siguiente se muestra todo el proceso de crear una secuencia, cifrarla, escribir en ella y cerrarla. En este ejemplo se crea una secuencia de red que se cifra usando las clases **CryptoStream** y **RijndaelManaged** . Se escribe un mensaje en la secuencia cifrada con la clase StreamWriter .

> **NOTE**
>
> También puede usar este ejemplo para escribir en un archivo. Para ello, elimine la referencia TcpClient y reemplace NetworkStream con FileStream.

```
Imports System
Imports System.IO
Imports System.Security.Cryptography
Imports System.Net.Sockets

Module Module1
Sub Main()
   Try
      'Create a TCP connection to a listening TCP process.
      'Use "localhost" to specify the current computer or
      'replace "localhost" with the IP address of the
      'listening process.
      Dim tcp As New TcpClient("localhost", 11000)

      'Create a network stream from the TCP connection.
      Dim netStream As NetworkStream = tcp.GetStream()

      'Create a new instance of the RijndaelManaged class
      'and encrypt the stream.
      Dim rmCrypto As New RijndaelManaged()

            Dim key As Byte() = {&H1, &H2, &H3, &H4, &H5, &H6, &H7, &H8, &H9, &H10, &H11, &H12, &H13, &H14,
&H15, &H16}
            Dim iv As Byte() = {&H1, &H2, &H3, &H4, &H5, &H6, &H7, &H8, &H9, &H10, &H11, &H12, &H13, &H14,
&H15, &H16}

      'Create a CryptoStream, pass it the NetworkStream, and encrypt
      'it with the Rijndael class.
      Dim cryptStream As New CryptoStream(netStream, rmCrypto.CreateEncryptor(key, iv),
CryptoStreamMode.Write)

      'Create a StreamWriter for easy writing to the
      'network stream.
      Dim sWriter As New StreamWriter(cryptStream)

      'Write to the stream.
      sWriter.WriteLine("Hello World!")

      'Inform the user that the message was written
      'to the stream.
      Console.WriteLine("The message was sent.")

      'Close all the connections.
      sWriter.Close()
      cryptStream.Close()
      netStream.Close()
      tcp.Close()
   Catch
      'Inform the user that an exception was raised.
      Console.WriteLine("The connection failed.")
   End Try
End Sub
End Module
```

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Net.Sockets;

public class main
{
   public static void Main(string[] args)
   {
      try
      {
         //Create a TCP connection to a listening TCP process.
         //Use "localhost" to specify the current computer or
         //replace "localhost" with the IP address of the
         //listening process.
         TcpClient tcp = new TcpClient("localhost",11000);

         //Create a network stream from the TCP connection.
         NetworkStream netStream = tcp.GetStream();

         //Create a new instance of the RijndaelManaged class
         // and encrypt the stream.
         RijndaelManaged rmCrypto = new RijndaelManaged();

         byte[] key = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14,
0x15, 0x16};
         byte[] iv = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14,
0x15, 0x16};

         //Create a CryptoStream, pass it the NetworkStream, and encrypt
         //it with the Rijndael class.
         CryptoStream cryptStream = new CryptoStream(netStream,
         rmCrypto.CreateEncryptor(key, iv),
         CryptoStreamMode.Write);

         //Create a StreamWriter for easy writing to the
         //network stream.
         StreamWriter sWriter = new StreamWriter(cryptStream);

         //Write to the stream.
         sWriter.WriteLine("Hello World!");

         //Inform the user that the message was written
         //to the stream.
         Console.WriteLine("The message was sent.");

         //Close all the connections.
         sWriter.Close();
         cryptStream.Close();
         netStream.Close();
         tcp.Close();
      }
      catch
      {
         //Inform the user that an exception was raised.
         Console.WriteLine("The connection failed.");
      }
   }
}
```

Para que el ejemplo anterior se ejecute correctamente, debe haber un proceso escuchando en la dirección IP y un número de puerto especificado en la clase TcpClient . Si existe un proceso de escucha, el código se conectará con el proceso de escucha, cifrará la secuencia usando el algoritmo simétrico Rijndael y escribirá "Hello World!" en la secuencia. Si el código es correcto, mostrará el texto siguiente en la consola:

```
The message was sent.
```

Sin embargo, si no se encuentra ningún proceso de escucha o si se produce una excepción, el código muestra el texto siguiente en la consola:

```
The connection failed.
```

Cifrado asimétrico

Los algoritmos asimétricos suelen usarse para cifrar pequeñas cantidades de datos, como el cifrado de una clave simétrica y un IV. Normalmente, cuando se realiza un cifrado asimétrico, se usa la clave pública generada por terceros. .NET Framework proporciona la clase RSACryptoServiceProvider necesaria para este propósito.

En el ejemplo siguiente se usa la información de clave pública para cifrar una clave simétrica y un IV. Se inicializan dos matrices de bytes que representan la clave pública de un tercero. Se inicializa un objeto RSAParameters en estos valores. A continuación, el objeto **RSAParameters** (junto con la clave pública que representa) se importa en **RSACryptoServiceProvider** a través del método RSACryptoServiceProvider.ImportParameters . Por último, se cifran la clave privada y el IV creados por una clase RijndaelManaged . Este ejemplo requiere sistemas que tengan instalado el cifrado de 128 bits.

```
Imports System
Imports System.Security.Cryptography

Module Module1

    Sub Main()
        'Initialize the byte arrays to the public key information.
      Dim publicKey As Byte() =  {214, 46, 220, 83, 160, 73, 40, 39, 201, 155, 19,202, 3, 11, 191, 178, 56,
74, 90, 36, 248, 103, 18, 144, 170, 163, 145, 87, 54, 61, 34, 220, 222, 207, 137, 149, 173, 14, 92, 120, 206,
222, 158, 28, 40, 24, 30, 16, 175, 108, 128, 35, 230, 118, 40, 121, 113, 125, 216, 130, 11, 24, 90, 48, 194,
240, 105, 44, 76, 34, 57, 249, 228, 125, 80, 38, 9, 136, 29, 117, 207, 139, 168, 181, 85, 137, 126, 10, 126,
242, 120, 247, 121, 8, 100, 12, 201, 171, 38, 226, 193, 180, 190, 117, 177, 87, 143, 242, 213, 11, 44, 180,
113, 93, 106, 99, 179, 68, 175, 211, 164, 116, 64, 148, 226, 254, 172, 147}

        Dim exponent As Byte() = {1, 0, 1}

        'Create values to store encrypted symmetric keys.
        Dim encryptedSymmetricKey() As Byte
        Dim encryptedSymmetricIV() As Byte

        'Create a new instance of the RSACryptoServiceProvider class.
        Dim rsa As New RSACryptoServiceProvider()

        'Create a new instance of the RSAParameters structure.
        Dim rsaKeyInfo As New RSAParameters()

        'Set rsaKeyInfo to the public key values.
        rsaKeyInfo.Modulus = publicKey
        rsaKeyInfo.Exponent = exponent

        'Import key parameters into rsa.
        rsa.ImportParameters(rsaKeyInfo)

        'Create a new instance of the RijndaelManaged class.
        Dim RM As New RijndaelManaged()

        'Encrypt the symmetric key and IV.
        encryptedSymmetricKey = rsa.Encrypt(RM.Key, False)
        encryptedSymmetricIV = rsa.Encrypt(RM.IV, False)
    End Sub

End Module
```

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

class Class1
{
   static void Main()
   {
      //Initialize the byte arrays to the public key information.
      byte[] publicKey = {214,46,220,83,160,73,40,39,201,155,19,202,3,11,191,178,56,
            74,90,36,248,103,18,144,170,163,145,87,54,61,34,220,222,
            207,137,149,173,14,92,120,206,222,158,28,40,24,30,16,175,
            108,128,35,230,118,40,121,113,125,216,130,11,24,90,48,194,
            240,105,44,76,34,57,249,228,125,80,38,9,136,29,117,207,139,
            168,181,85,137,126,10,126,242,120,247,121,8,100,12,201,171,
            38,226,193,180,190,117,177,87,143,242,213,11,44,180,113,93,
            106,99,179,68,175,211,164,116,64,148,226,254,172,147};

      byte[] exponent = {1,0,1};

      //Create values to store encrypted symmetric keys.
      byte[] encryptedSymmetricKey;
      byte[] encryptedSymmetricIV;

      //Create a new instance of the RSACryptoServiceProvider class.
      RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

      //Create a new instance of the RSAParameters structure.
      RSAParameters rsaKeyInfo = new RSAParameters();

      //Set rsaKeyInfo to the public key values.
      rsaKeyInfo.Modulus = publicKey;
      rsaKeyInfo.Exponent = exponent;

      //Import key parameters into RSA.
      rsa.ImportParameters(rsaKeyInfo);

      //Create a new instance of the RijndaelManaged class.
      RijndaelManaged rm = new RijndaelManaged();

      //Encrypt the symmetric key and IV.
      encryptedSymmetricKey = rsa.Encrypt(rm.Key, false);
      encryptedSymmetricIV = rsa.Encrypt(rm.IV, false);
   }
}
```

Consulte también

- Generar claves para cifrado y descifrado
- Descifrar datos
- Servicios criptográficos

Descifrar datos

02/07/2020 • 8 minutes to read • Edit Online

El descifrado es la operación inversa del cifrado. Para el cifrado de clave secreta, debe conocer la clave y el IV que se usaron para cifrar los datos. Para el cifrado de clave pública, debe conocer la clave pública (si los datos se cifraron mediante la clave privada) o la clave privada (si los datos se cifran mediante la clave pública).

Descifrado simétrico

El descifrado de los datos cifrados con algoritmos simétricos es similar al proceso usado para cifrar datos con algoritmos simétricos. La clase CryptoStream se usa con las clases de criptografía simétrica que proporciona .NET Framework para descifrar datos leídos de cualquier objeto de secuencia administrado.

En el ejemplo siguiente se muestra cómo crear una nueva instancia de la clase RijndaelManaged y usarla para realizar el descifrado en un objeto CryptoStream . En primer lugar, en este ejemplo se crea una nueva instancia de la clase **RijndaelManaged** . A continuación, se crea un objeto **CryptoStream** y se inicializa con el valor de una secuencia administrada llamada `myStream` . Después, se pasa al método **CreateDecryptor** de la clase **RijndaelManaged** la misma clave e IV que se usaron para el cifrado y, a continuación, se pasa al constructor **CryptoStream** . Por último, se pasa la enumeración **CryptoStreamMode.Read** al constructor **CryptoStream** para especificar el acceso de lectura a la secuencia.

```
Dim rmCrypto As New RijndaelManaged()
Dim cryptStream As New CryptoStream(myStream, rmCrypto.CreateDecryptor(rmCrypto.Key, rmCrypto.IV),
CryptoStreamMode.Read)
```

```
RijndaelManaged rmCrypto = new RijndaelManaged();
CryptoStream cryptStream = new CryptoStream(myStream, rmCrypto.CreateDecryptor(Key, IV),
CryptoStreamMode.Read);
```

En el ejemplo siguiente se muestra todo el proceso de crear una secuencia, descifrarla, leer en ella y cerrarla. Se crea un objeto TcpListener que inicializa una secuencia de red cuando se realiza una conexión al objeto de escucha. A continuación, la secuencia de red se descifra usando las clases **CryptoStream** y **RijndaelManaged** . En este ejemplo se supone que los valores de la clave y del IV se transfirieron correctamente o se acordaron previamente. No se muestra el código necesario para cifrar y transferir estos valores.

```
Imports System.IO
Imports System.Net
Imports System.Net.Sockets
Imports System.Security.Cryptography
Imports System.Threading

Module Module1
    Sub Main()
            'The key and IV must be the same values that were used
            'to encrypt the stream.
            Dim key As Byte() = {&H1, &H2, &H3, &H4, &H5, &H6, &H7, &H8, &H9, &H10, &H11, &H12, &H13, &H14,
&H15, &H16}
            Dim iv As Byte() = {&H1, &H2, &H3, &H4, &H5, &H6, &H7, &H8, &H9, &H10, &H11, &H12, &H13, &H14,
&H15, &H16}
        Try
            'Initialize a TCPListener on port 11000
            'using the current IP address.
            Dim tcpListen As New TcpListener(IPAddress.Any, 11000)

            'Start the listener.
            tcpListen.Start()

            'Check for a connection every five seconds.
            While Not tcpListen.Pending()
                Console.WriteLine("Still listening. Will try in 5 seconds.")

                Thread.Sleep(5000)
            End While

            'Accept the client if one is found.
            Dim tcp As TcpClient = tcpListen.AcceptTcpClient()

            'Create a network stream from the connection.
            Dim netStream As NetworkStream = tcp.GetStream()

            'Create a new instance of the RijndaelManaged class
            'and decrypt the stream.
            Dim rmCrypto As New RijndaelManaged()

            'Create an instance of the CryptoStream class, pass it the NetworkStream, and decrypt
            'it with the Rijndael class using the key and IV.
            Dim cryptStream As New CryptoStream(netStream, rmCrypto.CreateDecryptor(key, iv),
CryptoStreamMode.Read)

            'Read the stream.
            Dim sReader As New StreamReader(cryptStream)

            'Display the message.
            Console.WriteLine("The decrypted original message: {0}", sReader.ReadToEnd())

            'Close the streams.
            sReader.Close()
            netStream.Close()
            tcp.Close()
            'Catch any exceptions.
        Catch
            Console.WriteLine("The Listener Failed.")
        End Try
    End Sub
End Module
```

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Security.Cryptography;
```

```
using System.Threading;

class Class1
{
   static void Main(string[] args)
   {
      //The key and IV must be the same values that were used
      //to encrypt the stream.
      byte[] key = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
0x16};
      byte[] iv = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
0x16};
      try
      {
         //Initialize a TCPListener on port 11000
         //using the current IP address.
         TcpListener tcpListen = new TcpListener(IPAddress.Any, 11000);

         //Start the listener.
         tcpListen.Start();

         //Check for a connection every five seconds.
         while(!tcpListen.Pending())
         {
            Console.WriteLine("Still listening. Will try in 5 seconds.");
            Thread.Sleep(5000);
         }

         //Accept the client if one is found.
         TcpClient tcp = tcpListen.AcceptTcpClient();

         //Create a network stream from the connection.
         NetworkStream netStream = tcp.GetStream();

         //Create a new instance of the RijndaelManaged class
         // and decrypt the stream.
         RijndaelManaged rmCrypto = new RijndaelManaged();

         //Create a CryptoStream, pass it the NetworkStream, and decrypt
         //it with the Rijndael class using the key and IV.
         CryptoStream cryptStream = new CryptoStream(netStream,
            rmCrypto.CreateDecryptor(key, iv),
            CryptoStreamMode.Read);

         //Read the stream.
         StreamReader sReader = new StreamReader(cryptStream);

         //Display the message.
         Console.WriteLine("The decrypted original message: {0}", sReader.ReadToEnd());

         //Close the streams.
         sReader.Close();
         netStream.Close();
         tcp.Close();
      }
      //Catch any exceptions.
      catch
      {
         Console.WriteLine("The Listener Failed.");
      }
   }
}
```

Para que el ejemplo anterior funcione, debe realizarse una conexión cifrada con el agente de escucha. La conexión debe usar la misma clave, IV y algoritmo usados en el agente de escucha. Si se realiza esa conexión, el mensaje se descifra y se muestra en la consola.

Descifrado asimétrico

Normalmente, una parte (parte A) genera tanto una clave pública como privada y la almacena en memoria o en un contenedor de claves criptográficas. A continuación, la parte A envía la clave pública a otra parte (parte B). Mediante el uso de la clave pública, la parte B cifra los datos y envía los datos de vuelta a la entidad A. Después de recibir los datos, la parte A los descifra mediante la clave privada correspondiente. El descifrado será correcto solo si la parte A usa la clave privada que corresponde a la clave pública que la parte B usó para cifrar los datos.

Para obtener información sobre cómo almacenar una clave asimétrica en un contenedor de claves criptográficas seguro y cómo recuperar posteriormente la clave asimétrica, consulte How to: Store Asymmetric Keys in a Key Container.

En el ejemplo siguiente se muestra el descifrado de dos matrices de bytes que representan una clave simétrica y un IV. Para obtener información sobre cómo extraer la clave pública asimétrica del objeto RSACryptoServiceProvider en un formato que se pueda enviar fácilmente a un tercero, consulte Encrypting Data.

```
'Create a new instance of the RSACryptoServiceProvider class.
Dim rsa As New RSACryptoServiceProvider()

' Export the public key information and send it to a third party.
' Wait for the third party to encrypt some data and send it back.

'Decrypt the symmetric key and IV.
symmetricKey = rsa.Decrypt(encryptedSymmetricKey, False)
symmetricIV = rsa.Decrypt(encryptedSymmetricIV, False)
```

```
//Create a new instance of the RSACryptoServiceProvider class.
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

// Export the public key information and send it to a third party.
// Wait for the third party to encrypt some data and send it back.

//Decrypt the symmetric key and IV.
symmetricKey = rsa.Decrypt(encryptedSymmetricKey, false);
symmetricIV = rsa.Decrypt(encryptedSymmetricIV , false);
```

Consulte también

- Generar claves para cifrado y descifrado
- Cifrar datos
- Servicios criptográficos

Firmas criptográficas

02/07/2020 • 6 minutes to read • Edit Online

Las firmas digitales criptográficas usan algoritmos de clave pública para mantener la integridad de los datos. Si firma datos con una firma digital, otra persona puede comprobar la firma y confirmar que los datos provienen de usted y que no se han modificado después de ser firmados. Para más información sobre firmas digitales, vea Cryptographic Services.

En este tema se explica cómo generar y comprobar firmas digitales mediante clases en el espacio de nombres System.Security.Cryptography .

Generación de firmas

Las firmas digitales suelen aplicarse a valores hash que representan datos de mayor volumen. En el siguiente ejemplo se aplica una firma digital a un valor hash. Primero se crea una instancia de la clase RSACryptoServiceProvider para generar un par de claves pública y privada. Después se pasa RSACryptoServiceProvider a una nueva instancia de la clase RSAPKCS1SignatureFormatter . Con esto se transfiere la clave privada a RSAPKCS1SignatureFormatter, que es el que realmente realiza la firma digital. Antes de poder firmar el código hash, hay que especificar el algoritmo hash que se usará. En este ejemplo se usa el algoritmo SHA1. Por último, se llama al método CreateSignature para realizar el proceso de firma.

Debido a problemas de colisión con SHA1, Microsoft recomienda SHA256 o superior.

```
Imports System.Security.Cryptography

Module Module1
    Sub Main()
        'The hash value to sign.
        Dim hashValue As Byte() = {59, 4, 248, 102, 77, 97, 142, 201, 210, 12, 224, 93, 25, 41, 100, 197, 213,
134, 130, 135}

        'The value to hold the signed value.
        Dim signedHashValue() As Byte

        'Generate a public/private key pair.
        Dim rsa As New RSACryptoServiceProvider()

        'Create an RSAPKCS1SignatureFormatter object and pass it
        'the RSACryptoServiceProvider to transfer the private key.
        Dim rsaFormatter As New RSAPKCS1SignatureFormatter(rsa)

        'Set the hash algorithm to SHA1.
        rsaFormatter.SetHashAlgorithm("SHA1")

        'Create a signature for hashValue and assign it to
        'signedHashValue.
        signedHashValue = rsaFormatter.CreateSignature(hashValue)
    End Sub
End Module
```

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

class Class1
{
   static void Main()
   {
      //The hash value to sign.
      byte[] hashValue = {59,4,248,102,77,97,142,201,210,12,224,93,25,41,100,197,213,134,130,135};

      //The value to hold the signed value.
      byte[] signedHashValue;

      //Generate a public/private key pair.
      RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

      //Create an RSAPKCS1SignatureFormatter object and pass it the
      //RSACryptoServiceProvider to transfer the private key.
      RSAPKCS1SignatureFormatter rsaFormatter = new RSAPKCS1SignatureFormatter(rsa);

      //Set the hash algorithm to SHA1.
      rsaFormatter.SetHashAlgorithm("SHA1");

      //Create a signature for hashValue and assign it to
      //signedHashValue.
      signedHashValue = rsaFormatter.CreateSignature(hashValue);
   }
}
```

Firma de archivos XML

.NET Framework proporciona el espacio de nombres System.Security.Cryptography.Xml , que permite firmar XML. Es importante firmar XML cuando se desea comprobar su procedencia. Por ejemplo, si usa un servicio de cotización de acciones que utiliza XML firmado, puede comprobar el origen del XML.

Las clases de este espacio de nombres siguen la recomendación sobre procesamiento y sintaxis de firma de XML del World Wide Web Consortium.

Comprobación de firmas

Para comprobar que alguien en concreto firmó los datos, es necesaria la información siguiente:

- La clave pública del firmante de los datos.
- La firma digital.
- Los datos que se firmaron.
- El algoritmo de hash usado por el firmante.

Para comprobar una firma realizada por la clase RSAPKCS1SignatureFormatter , use la clase RSAPKCS1SignatureDeformatter . A la clase RSAPKCS1SignatureDeformatter debe proporcionársele la clave pública del firmante. Necesitará los valores del módulo y el exponente para especificar la clave pública. (La entidad que generó el par de claves pública y privada debe proporcionar estos valores). En primer lugar RSACryptoServiceProvider , cree un objeto que contenga la clave pública que comprobará la firma y, a continuación, inicialice una RSAParameters estructura a los valores de módulo y exponente que especifican la clave pública.

En el código siguiente se muestra la creación de una estructura RSAParameters . La propiedad `Modulus` se establece en el valor de una matriz de bytes denominada `modulusData` y la propiedad `Exponent` se establece en el valor de una matriz de bytes denominada `exponentData` .

```
Dim rsaKeyInfo As RSAParameters
rsaKeyInfo.Modulus = modulusData
rsaKeyInfo.Exponent = exponentData
```

```
RSAParameters rsaKeyInfo;
rsaKeyInfo.Modulus = modulusData;
rsaKeyInfo.Exponent = exponentData;
```

Después de crear el objeto RSAParameters , puede inicializar una nueva instancia de la clase RSACryptoServiceProvider para los valores especificados en RSAParameters. RSACryptoServiceProvider a su vez, se pasa al constructor de un RSAPKCS1SignatureDeformatter para transferir la clave.

El ejemplo siguiente ilustra este proceso. En este ejemplo, `hashValue` y `signedHashValue` son matrices de bytes que proporciona una parte remota. La parte remota firmó el `hashValue` mediante el algoritmo SHA1, produciendo así la firma digital `signedHashValue`. El RSAPKCS1SignatureDeformatter.VerifySignature método comprueba que la firma digital es válida y que se usó para firmar el `hashValue` .

```
Dim rsa As New RSACryptoServiceProvider()
rsa.ImportParameters(rsaKeyInfo)
Dim rsaDeformatter As New RSAPKCS1SignatureDeformatter(rsa)
rsaDeformatter.SetHashAlgorithm("SHA1")
If rsaDeformatter.VerifySignature(hashValue, signedHashValue) Then
   Console.WriteLine("The signature is valid.")
Else
   Console.WriteLine("The signature is not valid.")
End If
```

```
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
rsa.ImportParameters(rsaKeyInfo);
RSAPKCS1SignatureDeformatter rsaDeformatter = new RSAPKCS1SignatureDeformatter(rsa);
rsaDeformatter.SetHashAlgorithm("SHA1");
if(rsaDeformatter.VerifySignature(hashValue, signedHashValue))
{
   Console.WriteLine("The signature is valid.");
}
else
{
   Console.WriteLine("The signature is not valid.");
}
```

Este fragmento de código mostrará " `The signature is valid` " si la firma es válida y " `The signature is not valid` " si no lo es.

Consulte también

- Servicios criptográficos

Asegurar la integridad de los datos mediante códigos hash

02/07/2020 • 6 minutes to read • Edit Online

Un valor hash es un valor numérico de longitud fija que solo identifica datos. Los valores hash representan grandes cantidades de datos como valores numéricos mucho menores, por lo que se usan con firmas digitales. Puede firmar un valor hash de forma más eficaz que un valor mayor. Los valores hash también son útiles para comprobar la integridad de datos enviados a través de canales no seguros. El valor hash de los datos recibidos puede compararse con el valor hash de los datos porque se envió para determinar si se alteraron los datos.

En este tema, se describe cómo generar y comprobar códigos hash mediante las clases en el espacio de nombres System.Security.Cryptography.

Generar un valor hash

Las clases hash administradas pueden aplicar un valor hash a una matriz de bytes o a un objeto de secuencia administrado. En el ejemplo siguiente, se utiliza el algoritmo hash SHA1 para crear un valor hash para una cadena. El ejemplo utiliza la clase UnicodeEncoding para convertir la cadena en una matriz de bytes que aplica un valor hash mediante el uso de la clase SHA1Managed. Posteriormente el valor hash se muestra en la consola.

Debido a problemas de colisión con SHA1, Microsoft recomienda SHA256 o superior.

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

class Class1
{
    static void Main(string[] args)
    {
        byte[] hashValue;

        string messageString = "This is the original message!";

        //Create a new instance of the UnicodeEncoding class to
        //convert the string into an array of Unicode bytes.
        UnicodeEncoding ue = new UnicodeEncoding();

        //Convert the string into an array of bytes.
        byte[] messageBytes = ue.GetBytes(messageString);

        //Create a new instance of the SHA1Managed class to create
        //the hash value.
        SHA1Managed shHash = new SHA1Managed();

        //Create the hash value from the array of bytes.
        hashValue = shHash.ComputeHash(messageBytes);

        //Display the hash value to the console.
        foreach (byte b in hashValue)
        {
            Console.Write("{0} ", b);
        }
    }
}
```

```
Imports System.Security.Cryptography
Imports System.Text

Module Program
    Sub Main()
        Dim hashValue() As Byte

        Dim messageString As String = "This is the original message!"

        'Create a new instance of the UnicodeEncoding class to
        'convert the string into an array of Unicode bytes.
        Dim ue As New UnicodeEncoding()

        'Convert the string into an array of bytes.
        Dim messageBytes As Byte() = ue.GetBytes(messageString)

        'Create a new instance of the SHA1Managed class to create
        'the hash value.
        Dim shHash As New SHA1Managed()

        'Create the hash value from the array of bytes.
        hashValue = shHash.ComputeHash(messageBytes)

        'Display the hash value to the console.
        Dim b As Byte
        For Each b In hashValue
            Console.Write("{0} ", b)
        Next b
    End Sub
End Module
```

Este código mostrará la cadena siguiente en la consola:

```
59 4 248 102 77 97 142 201 210 12 224 93 25 41 100 197 213 134 130 135
```

Comprobar un valor hash

Los datos pueden compararse con un valor hash para determinar su integridad. Normalmente, el valor hash de los datos se calcula en un momento determinado y se protege de algún modo. Posteriormente, se puede calcular de nuevo el valor hash de los datos y compararse con el valor protegido. Si coinciden los dos valores hash, los datos no se han alterado. En caso contrario, los datos se han dañado. Para que este sistema funcione, el valor hash protegido debe cifrarse o mantenerse fuera del alcance de quienes no sean de confianza.

En el ejemplo siguiente, se compara el valor hash anterior de una cadena con un valor hash nuevo. En este ejemplo, se recorre cada byte de los valores hash y se realiza una comparación.

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

class Class1
{
    static void Main()
    {
        //This hash value is produced from "This is the original message!"
        //using SHA1Managed.
        byte[] sentHashValue = { 59, 4, 248, 102, 77, 97, 142, 201, 210, 12, 224, 93, 25, 41, 100, 197, 213,
134, 130, 135 };

        //This is the string that corresponds to the previous hash value.
        string messageString = "This is the original message!";

        byte[] compareHashValue;

        //Create a new instance of the UnicodeEncoding class to
        //convert the string into an array of Unicode bytes.
        UnicodeEncoding ue = new UnicodeEncoding();

        //Convert the string into an array of bytes.
        byte[] messageBytes = ue.GetBytes(messageString);

        //Create a new instance of the SHA1Managed class to create
        //the hash value.
        SHA1Managed shHash = new SHA1Managed();

        //Create the hash value from the array of bytes.
        compareHashValue = shHash.ComputeHash(messageBytes);

        bool same = true;

        //Compare the values of the two byte arrays.
        for (int x = 0; x < sentHashValue.Length; x++)
        {
            if (sentHashValue[x] != compareHashValue[x])
            {
                same = false;
            }
        }
        //Display whether or not the hash values are the same.
        if (same)
        {
            Console.WriteLine("The hash codes match.");
        }
        else
        {
            Console.WriteLine("The hash codes do not match.");
        }
    }
}
```

```
Imports System.Security.Cryptography
Imports System.Text

Module Module1
    Sub Main()
        'This hash value is produced from "This is the original message!"
        'using SHA1Managed.
        Dim sentHashValue As Byte() = {59, 4, 248, 102, 77, 97, 142, 201, 210, 12, 224, 93, 25, 41, 100, 197,
213, 134, 130, 135}

        'This is the string that corresponds to the previous hash value.
        Dim messageString As String = "This is the original message!"

        Dim compareHashValue() As Byte

        'Create a new instance of the UnicodeEncoding class to
        'convert the string into an array of Unicode bytes.
        Dim ue As New UnicodeEncoding()

        'Convert the string into an array of bytes.
        Dim messageBytes As Byte() = ue.GetBytes(messageString)

        'Create a new instance of the SHA1Managed class to create
        'the hash value.
        Dim shHash As New SHA1Managed()

        'Create the hash value from the array of bytes.
        compareHashValue = shHash.ComputeHash(messageBytes)

        Dim same As Boolean = True

        'Compare the values of the two byte arrays.
        Dim x As Integer
        For x = 0 To sentHashValue.Length - 1
            If sentHashValue(x) <> compareHashValue(x) Then
                same = False
            End If
        Next x
        'Display whether or not the hash values are the same.
        If same Then
            Console.WriteLine("The hash codes match.")
        Else
            Console.WriteLine("The hash codes do not match.")
        End If
    End Sub
End Module
```

Si los dos valores hash coinciden, este código mostrará lo siguiente en la consola:

```
The hash codes match.
```

Si no coinciden, el código muestra lo siguiente:

```
The hash codes do not match.
```

Consulte también

- Servicios criptográficos

Crear un esquema criptográfico

02/07/2020 • 2 minutes to read • Edit Online

Los componentes criptográficos de .NET Framework se pueden combinar para crear distintos esquemas para cifrar y descifrar datos.

Un esquema criptográfico simple para cifrar y descifrar datos podría especificar los siguientes pasos:

1. Cada parte genera un par de claves pública y privada.
2. Las partes intercambian sus claves públicas.
3. Cada parte genera una clave secreta para el cifrado TripleDES, por ejemplo, y cifra la clave recién creada con la clave pública de la otra parte.
4. Una parte envía los datos a la otra y combina la clave secreta de la otra con la suya propia en un determinado orden para crear una nueva clave secreta.
5. Después, las partes inician una conversación mediante el cifrado simétrico.

Crear un esquema criptográfico no es una tarea trivial.

Consulte también

- Servicios criptográficos

Procedimiento para cifrar elementos XML con claves simétricas

02/07/2020 • 11 minutes to read • Edit Online

Puede usar las clases en el espacio de nombres System.Security.Cryptography.Xml para cifrar un elemento dentro de un documento XML. El cifrado XML le permite almacenar o transportar información XML confidencial, sin preocuparse de que los datos se puedan leer con facilidad. Este procedimiento cifra un elemento XML mediante el algoritmo Estándar de cifrado avanzado (AES), también conocido como Rijndael.

Para obtener información sobre cómo descifrar un elemento XML cifrado mediante este procedimiento, vea Cómo: descifrar elementos XML con claves simétricas.

Al usar un algoritmo simétrico como AES para cifrar datos XML, debe usar la misma clave para cifrar y descifrar los datos XML. En el ejemplo de este procedimiento se supone que el código XML cifrado se descifrará con la misma clave y que las partes de cifrado y descifrado están de acuerdo en el algoritmo y la clave que se van a usar. En este ejemplo no se almacena ni se cifra la clave AES dentro del código XML cifrado.

Este ejemplo es idóneo en las situaciones en las que una aplicación necesita cifrar datos en función de una clave de sesión almacenada en memoria o en función de una clave criptográficamente segura derivada de una contraseña. En las situaciones en las que dos o más aplicaciones deben compartir datos XML cifrados, considere el uso de un esquema de cifrado basado en un algoritmo asimétrico o un certificado X.509.

Para cifrar un elemento XML con una clave simétrica

1. Genere una clave simétrica mediante la clase RijndaelManaged. Esta clave se usará para cifrar el elemento XML.

```
RijndaelManaged key = null;

try
{
    // Create a new Rijndael key.
    key = new RijndaelManaged();
```

```
Dim key As RijndaelManaged = Nothing

Try
    ' Create a new Rijndael key.
    key = New RijndaelManaged()
```

2. Cree un objeto XmlDocument cargando un archivo XML del disco. El objeto XmlDocument contiene el elemento XML que se va a cifrar.

```
// Load an XML document.
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.PreserveWhitespace = true;
xmlDoc.Load("test.xml");
```

```
' Load an XML document.
Dim xmlDoc As New XmlDocument()
xmlDoc.PreserveWhitespace = True
xmlDoc.Load("test.xml")
```

3. Busque el elemento especificado en el objeto XmlDocument y cree un objeto XmlElement nuevo para representar el elemento que desea cifrar. En este ejemplo, el elemento `"creditcard"` está cifrado.

```
XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementName)[0] as XmlElement;
```

```
Dim elementToEncrypt As XmlElement = Doc.GetElementsByTagName(ElementName)(0)
```

4. Cree una nueva instancia de la clase EncryptedXml y úsela para cifrar el XmlElement con la clave simétrica. El método EncryptData devuelve el elemento cifrado como una matriz de bytes cifrados.

```
EncryptedXml eXml = new EncryptedXml();

byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, Key, false);
```

```
Dim eXml As New EncryptedXml()

Dim encryptedElement As Byte() = eXml.EncryptData(elementToEncrypt, Key, False)
```

5. Construya un objeto EncryptedData y rellénelo con el identificador de dirección URL del elemento XML cifrado. Este identificador de dirección URL permite que una entidad descifradora sepa que el código XML contiene un elemento cifrado. Puede usar el campo XmlEncElementUrl para especificar el identificador de dirección URL.

```
EncryptedData edElement = new EncryptedData();
edElement.Type = EncryptedXml.XmlEncElementUrl;
```

```
Dim edElement As New EncryptedData()
edElement.Type = EncryptedXml.XmlEncElementUrl
```

6. Cree un objeto EncryptionMethod inicializado en el identificador de dirección URL del algoritmo criptográfico usado para generar la clave. Pase el objeto EncryptionMethod a la propiedad EncryptionMethod.

```
string encryptionMethod = null;

if (Key is TripleDES)
{
    encryptionMethod = EncryptedXml.XmlEncTripleDESUrl;
}
else if (Key is DES)
{
    encryptionMethod = EncryptedXml.XmlEncDESUrl;
}
if (Key is Rijndael)
{
    switch (Key.KeySize)
    {
        case 128:
            encryptionMethod = EncryptedXml.XmlEncAES128Url;
            break;
        case 192:
            encryptionMethod = EncryptedXml.XmlEncAES192Url;
            break;
        case 256:
            encryptionMethod = EncryptedXml.XmlEncAES256Url;
            break;
    }
}
else
{
    // Throw an exception if the transform is not in the previous categories
    throw new CryptographicException("The specified algorithm is not supported for XML Encryption.");
}

edElement.EncryptionMethod = new EncryptionMethod(encryptionMethod);
```

```
Dim encryptionMethod As String = Nothing

If TypeOf Key Is TripleDES Then
    encryptionMethod = EncryptedXml.XmlEncTripleDESUrl
ElseIf TypeOf Key Is DES Then
    encryptionMethod = EncryptedXml.XmlEncDESUrl
End If
If TypeOf Key Is Rijndael Then
    Select Case Key.KeySize
        Case 128
            encryptionMethod = EncryptedXml.XmlEncAES128Url
        Case 192
            encryptionMethod = EncryptedXml.XmlEncAES192Url
        Case 256
            encryptionMethod = EncryptedXml.XmlEncAES256Url
    End Select
Else
    ' Throw an exception if the transform is not in the previous categories
    Throw New CryptographicException("The specified algorithm is not supported for XML Encryption.")
End If

edElement.EncryptionMethod = New EncryptionMethod(encryptionMethod)
```

7. Agregue los datos de elementos cifrados al objeto EncryptedData.

```
edElement.CipherData.CipherValue = encryptedElement;
```

```
edElement.CipherData.CipherValue = encryptedElement
```

8. Reemplace el elemento del objeto XmlDocument original por el elemento EncryptedData.

```
EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);
```

```
EncryptedXml.ReplaceElement(elementToEncrypt, edElement, False)
```

Ejemplo

```
<root>
    <creditcard>
        <number>19834209</number>
        <expiry>02/02/2002</expiry>
    </creditcard>
</root>
```

```
using System;
using System.Xml;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;

namespace CSCrypto
{
    class Program
    {
        static void Main(string[] args)
        {
            RijndaelManaged key = null;

            try
            {
                // Create a new Rijndael key.
                key = new RijndaelManaged();
                // Load an XML document.
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.PreserveWhitespace = true;
                xmlDoc.Load("test.xml");

                // Encrypt the "creditcard" element.
                Encrypt(xmlDoc, "creditcard", key);

                Console.WriteLine("The element was encrypted");

                Console.WriteLine(xmlDoc.InnerXml);

                Decrypt(xmlDoc, key);

                Console.WriteLine("The element was decrypted");

                Console.WriteLine(xmlDoc.InnerXml);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            finally
            {
                // Clear the key.
                if (key != null)
                {
                    key.Clear();
                }
            }
```

```
        }
    }

    public static void Encrypt(XmlDocument Doc, string ElementName, SymmetricAlgorithm Key)
    {
        // Check the arguments.
        if (Doc == null)
            throw new ArgumentNullException("Doc");
        if (ElementName == null)
            throw new ArgumentNullException("ElementToEncrypt");
        if (Key == null)
            throw new ArgumentNullException("Alg");

        //////////////////////////////////////////////////
        // Find the specified element in the XmlDocument
        // object and create a new XmlElemnt object.
        //////////////////////////////////////////////////
        XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementName)[0] as XmlElement;
        // Throw an XmlException if the element was not found.
        if (elementToEncrypt == null)
        {
            throw new XmlException("The specified element was not found");
        }

        //////////////////////////////////////////////////
        // Create a new instance of the EncryptedXml class
        // and use it to encrypt the XmlElement with the
        // symmetric key.
        //////////////////////////////////////////////////

        EncryptedXml eXml = new EncryptedXml();

        byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, Key, false);
        //////////////////////////////////////////////////
        // Construct an EncryptedData object and populate
        // it with the desired encryption information.
        //////////////////////////////////////////////////

        EncryptedData edElement = new EncryptedData();
        edElement.Type = EncryptedXml.XmlEncElementUrl;

        // Create an EncryptionMethod element so that the
        // receiver knows which algorithm to use for decryption.
        // Determine what kind of algorithm is being used and
        // supply the appropriate URL to the EncryptionMethod element.

        string encryptionMethod = null;

        if (Key is TripleDES)
        {
            encryptionMethod = EncryptedXml.XmlEncTripleDESUrl;
        }
        else if (Key is DES)
        {
            encryptionMethod = EncryptedXml.XmlEncDESUrl;
        }
        if (Key is Rijndael)
        {
            switch (Key.KeySize)
            {
                case 128:
                    encryptionMethod = EncryptedXml.XmlEncAES128Url;
                    break;
                case 192:
                    encryptionMethod = EncryptedXml.XmlEncAES192Url;
                    break;
                case 256:
                    encryptionMethod = EncryptedXml.XmlEncAES256Url;
                    break;
            }
```

```
                }
            }
            else
            {
                // Throw an exception if the transform is not in the previous categories
                throw new CryptographicException("The specified algorithm is not supported for XML
Encryption.");
            }

            edElement.EncryptionMethod = new EncryptionMethod(encryptionMethod);

            // Add the encrypted element data to the
            // EncryptedData object.
            edElement.CipherData.CipherValue = encryptedElement;

            ////////////////////////////////////////////////
            // Replace the element from the original XmlDocument
            // object with the EncryptedData element.
            ////////////////////////////////////////////////
            EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);
        }

        public static void Decrypt(XmlDocument Doc, SymmetricAlgorithm Alg)
        {
            // Check the arguments.
            if (Doc == null)
                throw new ArgumentNullException("Doc");
            if (Alg == null)
                throw new ArgumentNullException("Alg");

            // Find the EncryptedData element in the XmlDocument.
            XmlElement encryptedElement = Doc.GetElementsByTagName("EncryptedData")[0] as XmlElement;

            // If the EncryptedData element was not found, throw an exception.
            if (encryptedElement == null)
            {
                throw new XmlException("The EncryptedData element was not found.");
            }

            // Create an EncryptedData object and populate it.
            EncryptedData edElement = new EncryptedData();
            edElement.LoadXml(encryptedElement);

            // Create a new EncryptedXml object.
            EncryptedXml exml = new EncryptedXml();

            // Decrypt the element using the symmetric key.
            byte[] rgbOutput = exml.DecryptData(edElement, Alg);

            // Replace the encryptedData element with the plaintext XML element.
            exml.ReplaceData(encryptedElement, rgbOutput);
        }
    }
}
```

```
Imports System.Xml
Imports System.Security.Cryptography
Imports System.Security.Cryptography.Xml




Module Program

    Sub Main(ByVal args() As String)
        Dim key As RijndaelManaged = Nothing

        Try
```

```
        Try
            ' Create a new Rijndael key.
            key = New RijndaelManaged()
            ' Load an XML document.
            Dim xmlDoc As New XmlDocument()
            xmlDoc.PreserveWhitespace = True
            xmlDoc.Load("test.xml")
            ' Encrypt the "creditcard" element.
            Encrypt(xmlDoc, "creditcard", key)

            Console.WriteLine("The element was encrypted")

            Console.WriteLine(xmlDoc.InnerXml)

            Decrypt(xmlDoc, key)

            Console.WriteLine("The element was decrypted")

            Console.WriteLine(xmlDoc.InnerXml)


        Catch e As Exception
            Console.WriteLine(e.Message)
        Finally
            ' Clear the key.
            If Not (key Is Nothing) Then
                key.Clear()
            End If
        End Try

    End Sub


    Sub Encrypt(ByVal Doc As XmlDocument, ByVal ElementName As String, ByVal Key As SymmetricAlgorithm)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If
        If ElementName Is Nothing Then
            Throw New ArgumentNullException("ElementToEncrypt")
        End If
        If Key Is Nothing Then
            Throw New ArgumentNullException("Alg")
        End If
        '''''''''''''''''''''''''''''''''''''''''''''''
        ' Find the specified element in the XmlDocument
        ' object and create a new XmlElemnt object.
        '''''''''''''''''''''''''''''''''''''''''''''''
        Dim elementToEncrypt As XmlElement = Doc.GetElementsByTagName(ElementName)(0)

        ' Throw an XmlException if the element was not found.
        If elementToEncrypt Is Nothing Then
            Throw New XmlException("The specified element was not found")
        End If

        '''''''''''''''''''''''''''''''''''''''''''''''
        ' Create a new instance of the EncryptedXml class
        ' and use it to encrypt the XmlElement with the
        ' symmetric key.
        '''''''''''''''''''''''''''''''''''''''''''''''
        Dim eXml As New EncryptedXml()

        Dim encryptedElement As Byte() = eXml.EncryptData(elementToEncrypt, Key, False)
        '''''''''''''''''''''''''''''''''''''''''''''''
        ' Construct an EncryptedData object and populate
        ' it with the desired encryption information.
        '''''''''''''''''''''''''''''''''''''''''''''''
        Dim edElement As New EncryptedData()
        edElement.Type = EncryptedXml.XmlEncElementUrl
```

```
        ' Create an EncryptionMethod element so that the
        ' receiver knows which algorithm to use for decryption.
        ' Determine what kind of algorithm is being used and
        ' supply the appropriate URL to the EncryptionMethod element.
        Dim encryptionMethod As String = Nothing

        If TypeOf Key Is TripleDES Then
            encryptionMethod = EncryptedXml.XmlEncTripleDESUrl
        ElseIf TypeOf Key Is DES Then
            encryptionMethod = EncryptedXml.XmlEncDESUrl
        End If
        If TypeOf Key Is Rijndael Then
            Select Case Key.KeySize
                Case 128
                    encryptionMethod = EncryptedXml.XmlEncAES128Url
                Case 192
                    encryptionMethod = EncryptedXml.XmlEncAES192Url
                Case 256
                    encryptionMethod = EncryptedXml.XmlEncAES256Url
            End Select
        Else
            ' Throw an exception if the transform is not in the previous categories
            Throw New CryptographicException("The specified algorithm is not supported for XML Encryption.")
        End If

        edElement.EncryptionMethod = New EncryptionMethod(encryptionMethod)
        ' Add the encrypted element data to the
        ' EncryptedData object.
        edElement.CipherData.CipherValue = encryptedElement
        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' Replace the element from the original XmlDocument
        ' object with the EncryptedData element.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        EncryptedXml.ReplaceElement(elementToEncrypt, edElement, False)

    End Sub


    Sub Decrypt(ByVal Doc As XmlDocument, ByVal Alg As SymmetricAlgorithm)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If
        If Alg Is Nothing Then
            Throw New ArgumentNullException("Alg")
        End If
        ' Find the EncryptedData element in the XmlDocument.
        Dim encryptedElement As XmlElement = Doc.GetElementsByTagName("EncryptedData")(0)

        ' If the EncryptedData element was not found, throw an exception.
        If encryptedElement Is Nothing Then
            Throw New XmlException("The EncryptedData element was not found.")
        End If


        ' Create an EncryptedData object and populate it.
        Dim edElement As New EncryptedData()
        edElement.LoadXml(encryptedElement)
        ' Create a new EncryptedXml object.
        Dim exml As New EncryptedXml()


        ' Decrypt the element using the symmetric key.
        Dim rgbOutput As Byte() = exml.DecryptData(edElement, Alg)
        ' Replace the encryptedData element with the plaintext XML element.
        exml.ReplaceData(encryptedElement, rgbOutput)
    End Sub
End Module
```

Compilar el código

- Para compilar este ejemplo, debe incluir una referencia a `System.Security.dll`.
- Incluya los siguientes espacios de nombres: System.Xml, System.Security.Cryptography y System.Security.Cryptography.Xml.

Seguridad de .NET Framework

No almacene nunca una clave criptográfica en texto sin formato ni transfiera una clave entre equipos en texto sin formato. En su lugar, use un contenedor de claves seguro para almacenar las claves criptográficas.

Cuando termine de usar una clave criptográfica, bórrela de la memoria estableciendo cada byte en cero o llamando al método Clear de la clase criptográfica administrada.

Consulte también

- System.Security.Cryptography.Xml
- Procedimiento para descifrar elementos XML con claves simétricas

Procedimiento para descifrar elementos XML con claves simétricas

02/07/2020 • 10 minutes to read • Edit Online

Puede usar las clases en el espacio de nombres System.Security.Cryptography.Xml para cifrar un elemento dentro de un documento XML. El cifrado XML le permite almacenar o transportar información XML confidencial, sin preocuparse de que los datos se puedan leer con facilidad. Este ejemplo de código descifra un elemento XML mediante el algoritmo AES (Estándar de cifrado avanzado), también conocido como Rijndael.

Para obtener información sobre cómo cifrar un elemento XML mediante este procedimiento, vea Cómo: cifrar elementos XML con claves simétricas.

Al usar un algoritmo simétrico como AES para cifrar datos XML, debe usar la misma clave para cifrar y descifrar los datos XML. En el ejemplo de este procedimiento se supone que el código XML cifrado se cifró con la misma clave y que las partes de cifrado y descifrado están de acuerdo en el algoritmo y la clave que se van a usar. En este ejemplo no se almacena ni se cifra la clave AES dentro del código XML cifrado.

Este ejemplo es idóneo en las situaciones en las que una aplicación necesita cifrar datos en función de una clave de sesión almacenada en memoria o en función de una clave criptográficamente segura derivada de una contraseña. En las situaciones en las que dos o más aplicaciones deben compartir datos XML cifrados, considere el uso de un esquema de cifrado basado en un algoritmo asimétrico o un certificado X.509.

Para descifrar un elemento XML con una clave simétrica

1. Cifre un elemento XML con la clave generada anteriormente mediante las técnicas descritas en Cómo: cifrar elementos XML con claves simétricas.
2. Busque el `EncryptedData` elemento <> (definido por el estándar de cifrado XML) en un XmlDocument objeto que contenga el XML cifrado y cree un nuevo XmlElement objeto para representar ese elemento.

```
XmlElement encryptedElement = Doc.GetElementsByTagName("EncryptedData")[0] as XmlElement;
```

```
Dim encryptedElement As XmlElement = Doc.GetElementsByTagName("EncryptedData")(0)
```

3. Cree un objeto EncryptedData cargando los datos XML sin formato desde el objeto XmlElement que se creó previamente.

```
EncryptedData edElement = new EncryptedData();
edElement.LoadXml(encryptedElement);
```

```
Dim edElement As New EncryptedData()
edElement.LoadXml(encryptedElement)
```

4. Cree un nuevo objeto EncryptedXml y úselo para descifrar los datos XML usando la misma clave que se usó para el cifrado.

```
EncryptedXml exml = new EncryptedXml();

// Decrypt the element using the symmetric key.
byte[] rgbOutput = exml.DecryptData(edElement, Alg);
```

```
Dim exml As New EncryptedXml()


' Decrypt the element using the symmetric key.
Dim rgbOutput As Byte() = exml.DecryptData(edElement, Alg)
```

5. Reemplace el elemento cifrado por el elemento de texto simple recién descifrado dentro del documento XML.

```
exml.ReplaceData(encryptedElement, rgbOutput);
```

```
exml.ReplaceData(encryptedElement, rgbOutput)
```

Ejemplo

En este ejemplo se supone que un archivo llamado `"test.xml"` se encuentra en el mismo directorio que el programa compilado. También se supone que `"test.xml"` contiene un elemento `"creditcard"`. Puede colocar el siguiente código XML en un archivo llamado `test.xml` y usarlo con este ejemplo.

```
<root>
    <creditcard>
        <number>19834209</number>
        <expiry>02/02/2002</expiry>
    </creditcard>
</root>
```

```
using System;
using System.Xml;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;

namespace CSCrypto
{
    class Program
    {
        static void Main(string[] args)
        {
            RijndaelManaged key = null;

            try
            {
                // Create a new Rijndael key.
                key = new RijndaelManaged();
                // Load an XML document.
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.PreserveWhitespace = true;
                xmlDoc.Load("test.xml");

                // Encrypt the "creditcard" element.
                Encrypt(xmlDoc, "creditcard", key);

                Console.WriteLine("The element was encrypted");
```

```
            Console.WriteLine(xmlDoc.InnerXml);

            Decrypt(xmlDoc, key);

            Console.WriteLine("The element was decrypted");

            Console.WriteLine(xmlDoc.InnerXml);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        finally
        {
            // Clear the key.
            if (key != null)
            {
                key.Clear();
            }
        }
    }

    public static void Encrypt(XmlDocument Doc, string ElementName, SymmetricAlgorithm Key)
    {
        // Check the arguments.
        if (Doc == null)
            throw new ArgumentNullException("Doc");
        if (ElementName == null)
            throw new ArgumentNullException("ElementToEncrypt");
        if (Key == null)
            throw new ArgumentNullException("Alg");

        ////////////////////////////////////////////////
        // Find the specified element in the XmlDocument
        // object and create a new XmlElemnt object.
        ////////////////////////////////////////////////
        XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementName)[0] as XmlElement;
        // Throw an XmlException if the element was not found.
        if (elementToEncrypt == null)
        {
            throw new XmlException("The specified element was not found");
        }

        //////////////////////////////////////////////////
        // Create a new instance of the EncryptedXml class
        // and use it to encrypt the XmlElement with the
        // symmetric key.
        //////////////////////////////////////////////////

        EncryptedXml eXml = new EncryptedXml();

        byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, Key, false);
        ////////////////////////////////////////////////
        // Construct an EncryptedData object and populate
        // it with the desired encryption information.
        ////////////////////////////////////////////////

        EncryptedData edElement = new EncryptedData();
        edElement.Type = EncryptedXml.XmlEncElementUrl;

        // Create an EncryptionMethod element so that the
        // receiver knows which algorithm to use for decryption.
        // Determine what kind of algorithm is being used and
        // supply the appropriate URL to the EncryptionMethod element.

        string encryptionMethod = null;

        if (Key is TripleDES)
```

```
            {
                encryptionMethod = EncryptedXml.XmlEncTripleDESUrl;
            }
            else if (Key is DES)
            {
                encryptionMethod = EncryptedXml.XmlEncDESUrl;
            }
            if (Key is Rijndael)
            {
                switch (Key.KeySize)
                {
                    case 128:
                        encryptionMethod = EncryptedXml.XmlEncAES128Url;
                        break;
                    case 192:
                        encryptionMethod = EncryptedXml.XmlEncAES192Url;
                        break;
                    case 256:
                        encryptionMethod = EncryptedXml.XmlEncAES256Url;
                        break;
                }
            }
            else
            {
                // Throw an exception if the transform is not in the previous categories
                throw new CryptographicException("The specified algorithm is not supported for XML
Encryption.");
            }

            edElement.EncryptionMethod = new EncryptionMethod(encryptionMethod);

            // Add the encrypted element data to the
            // EncryptedData object.
            edElement.CipherData.CipherValue = encryptedElement;

            //////////////////////////////////////////////////
            // Replace the element from the original XmlDocument
            // object with the EncryptedData element.
            //////////////////////////////////////////////////
            EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);
        }

        public static void Decrypt(XmlDocument Doc, SymmetricAlgorithm Alg)
        {
            // Check the arguments.
            if (Doc == null)
                throw new ArgumentNullException("Doc");
            if (Alg == null)
                throw new ArgumentNullException("Alg");

            // Find the EncryptedData element in the XmlDocument.
            XmlElement encryptedElement = Doc.GetElementsByTagName("EncryptedData")[0] as XmlElement;

            // If the EncryptedData element was not found, throw an exception.
            if (encryptedElement == null)
            {
                throw new XmlException("The EncryptedData element was not found.");
            }

            // Create an EncryptedData object and populate it.
            EncryptedData edElement = new EncryptedData();
            edElement.LoadXml(encryptedElement);

            // Create a new EncryptedXml object.
            EncryptedXml exml = new EncryptedXml();

            // Decrypt the element using the symmetric key.
            byte[] rgbOutput = exml.DecryptData(edElement, Alg);
```

```
            // Replace the encryptedData element with the plaintext XML element.
            exml.ReplaceData(encryptedElement, rgbOutput);
        }
    }
}
```

```
Imports System.Xml
Imports System.Security.Cryptography
Imports System.Security.Cryptography.Xml



Module Program

    Sub Main(ByVal args() As String)
        Dim key As RijndaelManaged = Nothing

        Try
            ' Create a new Rijndael key.
            key = New RijndaelManaged()
            ' Load an XML document.
            Dim xmlDoc As New XmlDocument()
            xmlDoc.PreserveWhitespace = True
            xmlDoc.Load("test.xml")
            ' Encrypt the "creditcard" element.
            Encrypt(xmlDoc, "creditcard", key)

            Console.WriteLine("The element was encrypted")

            Console.WriteLine(xmlDoc.InnerXml)

            Decrypt(xmlDoc, key)

            Console.WriteLine("The element was decrypted")

            Console.WriteLine(xmlDoc.InnerXml)


        Catch e As Exception
            Console.WriteLine(e.Message)
        Finally
            ' Clear the key.
            If Not (key Is Nothing) Then
                key.Clear()
            End If
        End Try

    End Sub


    Sub Encrypt(ByVal Doc As XmlDocument, ByVal ElementName As String, ByVal Key As SymmetricAlgorithm)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If
        If ElementName Is Nothing Then
            Throw New ArgumentNullException("ElementToEncrypt")
        End If
        If Key Is Nothing Then
            Throw New ArgumentNullException("Alg")
        End If
        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' Find the specified element in the XmlDocument
        ' object and create a new XmlElemnt object.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        Dim elementToEncrypt As XmlElement = Doc.GetElementsByTagName(ElementName)(0)
```

```
        Dim elementToEncrypt As XmlElement = Doc.GetElementsByTagName(ElementName)(0)

        ' Throw an XmlException if the element was not found.
        If elementToEncrypt Is Nothing Then
            Throw New XmlException("The specified element was not found")
        End If

        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' Create a new instance of the EncryptedXml class
        ' and use it to encrypt the XmlElement with the
        ' symmetric key.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        Dim eXml As New EncryptedXml()

        Dim encryptedElement As Byte() = eXml.EncryptData(elementToEncrypt, Key, False)
        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' Construct an EncryptedData object and populate
        ' it with the desired encryption information.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        Dim edElement As New EncryptedData()
        edElement.Type = EncryptedXml.XmlEncElementUrl
        ' Create an EncryptionMethod element so that the
        ' receiver knows which algorithm to use for decryption.
        ' Determine what kind of algorithm is being used and
        ' supply the appropriate URL to the EncryptionMethod element.
        Dim encryptionMethod As String = Nothing

        If TypeOf Key Is TripleDES Then
            encryptionMethod = EncryptedXml.XmlEncTripleDESUrl
        ElseIf TypeOf Key Is DES Then
            encryptionMethod = EncryptedXml.XmlEncDESUrl
        End If
        If TypeOf Key Is Rijndael Then
            Select Case Key.KeySize
                Case 128
                    encryptionMethod = EncryptedXml.XmlEncAES128Url
                Case 192
                    encryptionMethod = EncryptedXml.XmlEncAES192Url
                Case 256
                    encryptionMethod = EncryptedXml.XmlEncAES256Url
            End Select
        Else
            ' Throw an exception if the transform is not in the previous categories
            Throw New CryptographicException("The specified algorithm is not supported for XML Encryption.")
        End If

        edElement.EncryptionMethod = New EncryptionMethod(encryptionMethod)
        ' Add the encrypted element data to the
        ' EncryptedData object.
        edElement.CipherData.CipherValue = encryptedElement
        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' Replace the element from the original XmlDocument
        ' object with the EncryptedData element.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        EncryptedXml.ReplaceElement(elementToEncrypt, edElement, False)

    End Sub


    Sub Decrypt(ByVal Doc As XmlDocument, ByVal Alg As SymmetricAlgorithm)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If
        If Alg Is Nothing Then
            Throw New ArgumentNullException("Alg")
        End If
        ' Find the EncryptedData element in the XmlDocument.
        Dim encryptedElement As XmlElement = Doc.GetElementsByTagName("EncryptedData")(0)
```

```
        ' If the EncryptedData element was not found, throw an exception.
        If encryptedElement Is Nothing Then
            Throw New XmlException("The EncryptedData element was not found.")
        End If


        ' Create an EncryptedData object and populate it.
        Dim edElement As New EncryptedData()
        edElement.LoadXml(encryptedElement)
        ' Create a new EncryptedXml object.
        Dim exml As New EncryptedXml()


        ' Decrypt the element using the symmetric key.
        Dim rgbOutput As Byte() = exml.DecryptData(edElement, Alg)
        ' Replace the encryptedData element with the plaintext XML element.
        exml.ReplaceData(encryptedElement, rgbOutput)
    End Sub
End Module
```

Compilar el código

- Para compilar este ejemplo, debe incluir una referencia a `System.Security.dll`.
- Incluya los siguientes espacios de nombres: System.Xml, System.Security.Cryptography y System.Security.Cryptography.Xml.

Seguridad de .NET Framework

No almacene nunca una clave criptográfica en texto sin formato ni transfiera una clave entre equipos en texto sin formato.

Cuando haya terminado de usar una clave criptográfica simétrica, bórrela de la memoria estableciendo cada byte en cero o llamando al método Clear de la clase criptográfica administrada.

Consulte también

- System.Security.Cryptography.Xml
- Procedimiento para cifrar elementos XML con claves simétricas

Procedimiento para cifrar elementos XML con claves asimétricas

02/07/2020 • 18 minutes to read • Edit Online

Puede usar las clases en el espacio de nombres System.Security.Cryptography.Xml para cifrar un elemento dentro de un documento XML. El cifrado XML es un método estándar para intercambiar o almacenar datos XML cifrados sin preocuparse de que los datos puedan leerse con facilidad. Para obtener más información sobre el estándar de cifrado XML, consulte la especificación de World Wide Web Consortium (W3C) para el cifrado XML ubicado en https://www.w3.org/TR/xmldsig-core/ .

Puede usar el cifrado de XML para reemplazar cualquier elemento o documento XML con un elemento < `EncryptedData` > que contenga los datos XML cifrados. El `EncryptedData` elemento <> también puede contener subelementos que incluyan información sobre las claves y los procesos usados durante el cifrado. El cifrado XML permite que un documento contenga varios elementos cifrados y permite cifrar varias veces un elemento. En el ejemplo de código de este procedimiento se muestra cómo crear un `EncryptedData` elemento <> junto con otros subelementos que se pueden usar posteriormente durante el descifrado.

En este ejemplo se cifra un elemento XML mediante dos claves. Genera un par de claves públicas/privadas RSA y lo guarda en un contenedor de claves seguras. Después, el ejemplo crea una clave de sesión independiente mediante el algoritmo AES (Estándar de cifrado avanzado), también llamado algoritmo de Rijndael. El ejemplo usa la clave de sesión AES para cifrar el documento XML y usa la clave pública RSA para cifrar la clave de sesión AES. Por último, en el ejemplo se guarda la clave de sesión AES cifrada y los datos XML cifrados en el documento XML dentro de un nuevo < `EncryptedData` > elemento.

Para descifrar el elemento XML, recupere la clave privada RSA del contenedor de claves, úsela para descifrar la clave de sesión y, posteriormente, descifrar el documento. Para obtener más información sobre cómo descifrar un elemento XML cifrado mediante este procedimiento, vea Cómo: descifrar elementos XML con claves asimétricas.

Este ejemplo resulta adecuado en aquellas situaciones en las que varias aplicaciones tienen que compartir datos cifrados o en las que una aplicación tiene que guardar datos cifrados entre los intervalos en los que se ejecuta.

Para cifrar un elemento XML con una clave asimétrica

1. Cree un objeto CspParameters y especifique el nombre del contenedor de claves.

```
CspParameters cspParams = new CspParameters();
cspParams.KeyContainerName = "XML_ENC_RSA_KEY";
```

```
Dim cspParams As New CspParameters()
cspParams.KeyContainerName = "XML_ENC_RSA_KEY"
```

2. Genere una clave simétrica mediante la clase RSACryptoServiceProvider. La clave se guarda automáticamente en el contenedor de claves al pasar el objeto CspParameters al constructor de la clase RSACryptoServiceProvider. Esta clave se usará para cifrar la clave de sesión AES y se puede recuperar más adelante para descifrarla.

```
RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);
```

```
Dim rsaKey As New RSACryptoServiceProvider(cspParams)
```

3. Cree un objeto XmlDocument cargando un archivo XML del disco. El objeto XmlDocument contiene el elemento XML que se va a cifrar.

```
// Create an XmlDocument object.
XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.
try
{
    xmlDoc.PreserveWhitespace = true;
    xmlDoc.Load("test.xml");
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
```

```
' Create an XmlDocument object.
Dim xmlDoc As New XmlDocument()

' Load an XML file into the XmlDocument object.
Try
    xmlDoc.PreserveWhitespace = True
    xmlDoc.Load("test.xml")
Catch e As Exception
    Console.WriteLine(e.Message)
End Try
```

4. Busque el elemento especificado en el objeto XmlDocument y cree un objeto XmlElement nuevo para representar el elemento que desea cifrar. En este ejemplo, el elemento `"creditcard"` está cifrado.

```
XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementToEncrypt)[0] as XmlElement;

// Throw an XmlException if the element was not found.
if (elementToEncrypt == null)
{
    throw new XmlException("The specified element was not found");
}
```

```
Dim elementToEncrypt As XmlElement = Doc.GetElementsByTagName(EncryptionElement)(0) '

' Throw an XmlException if the element was not found.
If elementToEncrypt Is Nothing Then
    Throw New XmlException("The specified element was not found")
End If
```

5. Cree una nueva sesión de clave mediante la clase RijndaelManaged. Esta clave cifrará el elemento XML, se cifrará ella misma y se colocará en el documento XML.

```
// Create a 256 bit Rijndael key.
sessionKey = new RijndaelManaged();
sessionKey.KeySize = 256;
```

```
' Create a 256 bit Rijndael key.
sessionKey = New RijndaelManaged()
sessionKey.KeySize = 256
```

6. Cree una nueva instancia de la clase EncryptedXml y úsela para cifrar el elemento especificado mediante la clave de sesión. El método EncryptData devuelve el elemento cifrado como una matriz de bytes cifrados.

```
EncryptedXml eXml = new EncryptedXml();

byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, sessionKey, false);
```

```
Dim eXml As New EncryptedXml()

Dim encryptedElement As Byte() = eXml.EncryptData(elementToEncrypt, sessionKey, False)
```

7. Construya un objeto EncryptedData y rellénelo con el identificador de dirección URL del elemento XML cifrado. Este identificador de dirección URL permite que una entidad descifradora sepa que el código XML contiene un elemento cifrado. Puede usar el campo XmlEncElementUrl para especificar el identificador de dirección URL. El elemento XML de texto simple se reemplazará por un `EncryptedData` elemento <> encapsulado por este EncryptedData objeto.

```
EncryptedData edElement = new EncryptedData();
edElement.Type = EncryptedXml.XmlEncElementUrl;
edElement.Id = EncryptionElementID;
```

```
Dim edElement As New EncryptedData()
edElement.Type = EncryptedXml.XmlEncElementUrl
edElement.Id = EncryptionElementID
```

8. Cree un objeto EncryptionMethod inicializado en el identificador de dirección URL del algoritmo criptográfico usado para generar la clave de sesión. Pase el objeto EncryptionMethod a la propiedad EncryptionMethod.

```
edElement.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncAES256Url);
```

```
edElement.EncryptionMethod = New EncryptionMethod(EncryptedXml.XmlEncAES256Url)
```

9. Cree un objeto EncryptedKey para incluir la clave de sesión cifrada. Cifre la clave de sesión, agréguela al objeto EncryptedKey y escriba un nombre de clave de sesión y una dirección URL de identificador de clave.

```
EncryptedKey ek = new EncryptedKey();

byte[] encryptedKey = EncryptedXml.EncryptKey(sessionKey.Key, Alg, false);

ek.CipherData = new CipherData(encryptedKey);

ek.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncRSA15Url);
```

```
Dim ek As New EncryptedKey()

Dim encryptedKey As Byte() = EncryptedXml.EncryptKey(sessionKey.Key, Alg, False)

ek.CipherData = New CipherData(encryptedKey)

ek.EncryptionMethod = New EncryptionMethod(EncryptedXml.XmlEncRSA15Url)
```

10. Cree un nuevo objeto DataReference que asigne los datos cifrados a una clave de sesión determinada. Este paso opcional le permite especificar fácilmente que una clave única cifró varias partes de un documento XML.

```
DataReference dRef = new DataReference();

// Specify the EncryptedData URI.
dRef.Uri = "#" + EncryptionElementID;

// Add the DataReference to the EncryptedKey.
ek.AddReference(dRef);
```

```
Dim dRef As New DataReference()

' Specify the EncryptedData URI.
dRef.Uri = "#" + EncryptionElementID

' Add the DataReference to the EncryptedKey.
ek.AddReference(dRef)
```

11. Agregue la clave cifrada al objeto EncryptedData.

```
edElement.KeyInfo.AddClause(new KeyInfoEncryptedKey(ek));
```

```
edElement.KeyInfo.AddClause(New KeyInfoEncryptedKey(ek))
```

12. Cree un nuevo objeto KeyInfo para especificar el nombre de la clave RSA. Agréguelo al objeto EncryptedData. Así, la entidad descifradora podrá identificar la clave asimétrica que debe usar al descifrar la clave de sesión.

```
// Create a new KeyInfoName element.
KeyInfoName kin = new KeyInfoName();

// Specify a name for the key.
kin.Value = KeyName;

// Add the KeyInfoName element to the
// EncryptedKey object.
ek.KeyInfo.AddClause(kin);
```

```
' Create a new KeyInfoName element.
Dim kin As New KeyInfoName()

' Specify a name for the key.
kin.Value = KeyName

' Add the KeyInfoName element to the
' EncryptedKey object.
ek.KeyInfo.AddClause(kin)
```

13. Agregue los datos de elementos cifrados al objeto EncryptedData.

```
edElement.CipherData.CipherValue = encryptedElement;
```

```
edElement.CipherData.CipherValue = encryptedElement
```

14. Reemplace el elemento del objeto XmlDocument original por el elemento EncryptedData.

```
EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);
```

```
EncryptedXml.ReplaceElement(elementToEncrypt, edElement, False)
```

15. Guarde el objeto XmlDocument.

```
xmlDoc.Save("test.xml");
```

```
xmlDoc.Save("test.xml")
```

Ejemplo

En este ejemplo se supone que un archivo llamado `"test.xml"` se encuentra en el mismo directorio que el programa compilado. También se supone que `"test.xml"` contiene un elemento `"creditcard"`. Puede colocar el siguiente código XML en un archivo llamado `test.xml` y usarlo con este ejemplo.

```
<root>
    <creditcard>
        <number>19834209</number>
        <expiry>02/02/2002</expiry>
    </creditcard>
</root>
```

```
using System;
using System.Xml;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;

class Program
{
    static void Main(string[] args)
    {
        // Create an XmlDocument object.
```

```
        XmlDocument xmlDoc = new XmlDocument();

        // Load an XML file into the XmlDocument object.
        try
        {
            xmlDoc.PreserveWhitespace = true;
            xmlDoc.Load("test.xml");
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }

        // Create a new CspParameters object to specify
        // a key container.
        CspParameters cspParams = new CspParameters();
        cspParams.KeyContainerName = "XML_ENC_RSA_KEY";

        // Create a new RSA key and save it in the container.  This key will encrypt
        // a symmetric key, which will then be encryped in the XML document.
        RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

        try
        {
            // Encrypt the "creditcard" element.
            Encrypt(xmlDoc, "creditcard", "EncryptedElement1", rsaKey, "rsaKey");

            // Save the XML document.
            xmlDoc.Save("test.xml");

            // Display the encrypted XML to the console.
            Console.WriteLine("Encrypted XML:");
            Console.WriteLine();
            Console.WriteLine(xmlDoc.OuterXml);
            Decrypt(xmlDoc, rsaKey, "rsaKey");
            xmlDoc.Save("test.xml");
            // Display the encrypted XML to the console.
            Console.WriteLine();
            Console.WriteLine("Decrypted XML:");
            Console.WriteLine();
            Console.WriteLine(xmlDoc.OuterXml);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        finally
        {
            // Clear the RSA key.
            rsaKey.Clear();
        }

        Console.ReadLine();
    }

    public static void Encrypt(XmlDocument Doc, string ElementToEncrypt, string EncryptionElementID, RSA Alg,
string KeyName)
    {
        // Check the arguments.
        if (Doc == null)
            throw new ArgumentNullException("Doc");
        if (ElementToEncrypt == null)
            throw new ArgumentNullException("ElementToEncrypt");
        if (EncryptionElementID == null)
            throw new ArgumentNullException("EncryptionElementID");
        if (Alg == null)
            throw new ArgumentNullException("Alg");
        if (KeyName == null)
            throw new ArgumentNullException("KeyName");
```

```
        throw new ArgumentNullException( "KeyName" );

    ////////////////////////////////////////////////
    // Find the specified element in the XmlDocument
    // object and create a new XmlElemnt object.
    ////////////////////////////////////////////////
    XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementToEncrypt)[0] as XmlElement;

    // Throw an XmlException if the element was not found.
    if (elementToEncrypt == null)
    {
        throw new XmlException("The specified element was not found");
    }
    RijndaelManaged sessionKey = null;

    try
    {
        //////////////////////////////////////////////////
        // Create a new instance of the EncryptedXml class
        // and use it to encrypt the XmlElement with the
        // a new random symmetric key.
        //////////////////////////////////////////////////

        // Create a 256 bit Rijndael key.
        sessionKey = new RijndaelManaged();
        sessionKey.KeySize = 256;

        EncryptedXml eXml = new EncryptedXml();

        byte[] encryptedElement = eXml.EncryptData(elementToEncrypt, sessionKey, false);
        ////////////////////////////////////////////////
        // Construct an EncryptedData object and populate
        // it with the desired encryption information.
        ////////////////////////////////////////////////

        EncryptedData edElement = new EncryptedData();
        edElement.Type = EncryptedXml.XmlEncElementUrl;
        edElement.Id = EncryptionElementID;
        // Create an EncryptionMethod element so that the
        // receiver knows which algorithm to use for decryption.

        edElement.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncAES256Url);
        // Encrypt the session key and add it to an EncryptedKey element.
        EncryptedKey ek = new EncryptedKey();

        byte[] encryptedKey = EncryptedXml.EncryptKey(sessionKey.Key, Alg, false);

        ek.CipherData = new CipherData(encryptedKey);

        ek.EncryptionMethod = new EncryptionMethod(EncryptedXml.XmlEncRSA15Url);

        // Create a new DataReference element
        // for the KeyInfo element.  This optional
        // element specifies which EncryptedData
        // uses this key.  An XML document can have
        // multiple EncryptedData elements that use
        // different keys.
        DataReference dRef = new DataReference();

        // Specify the EncryptedData URI.
        dRef.Uri = "#" + EncryptionElementID;

        // Add the DataReference to the EncryptedKey.
        ek.AddReference(dRef);
        // Add the encrypted key to the
        // EncryptedData object.

        edElement.KeyInfo.AddClause(new KeyInfoEncryptedKey(ek));
        // Set the KeyInfo element to specify the
        // name of the RSA key
```

```
            // name of the RSA key.


            // Create a new KeyInfoName element.
            KeyInfoName kin = new KeyInfoName();

            // Specify a name for the key.
            kin.Value = KeyName;

            // Add the KeyInfoName element to the
            // EncryptedKey object.
            ek.KeyInfo.AddClause(kin);
            // Add the encrypted element data to the
            // EncryptedData object.
            edElement.CipherData.CipherValue = encryptedElement;
            //////////////////////////////////////////////////
            // Replace the element from the original XmlDocument
            // object with the EncryptedData element.
            //////////////////////////////////////////////////
            EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);
        }
        catch (Exception e)
        {
            // re-throw the exception.
            throw e;
        }
        finally
        {
            if (sessionKey != null)
            {
                sessionKey.Clear();
            }
        }
    }

    public static void Decrypt(XmlDocument Doc, RSA Alg, string KeyName)
    {
        // Check the arguments.
        if (Doc == null)
            throw new ArgumentNullException("Doc");
        if (Alg == null)
            throw new ArgumentNullException("Alg");
        if (KeyName == null)
            throw new ArgumentNullException("KeyName");

        // Create a new EncryptedXml object.
        EncryptedXml exml = new EncryptedXml(Doc);

        // Add a key-name mapping.
        // This method can only decrypt documents
        // that present the specified key name.
        exml.AddKeyNameMapping(KeyName, Alg);

        // Decrypt the element.
        exml.DecryptDocument();
    }
}
```

```
Imports System.Xml
Imports System.Security.Cryptography
Imports System.Security.Cryptography.Xml



Class Program

    Shared Sub Main(ByVal args() As String)
```

```
    Shared Sub Main(ByVal args() As String)
        ' Create an XmlDocument object.
        Dim xmlDoc As New XmlDocument()

        ' Load an XML file into the XmlDocument object.
        Try
            xmlDoc.PreserveWhitespace = True
            xmlDoc.Load("test.xml")
        Catch e As Exception
            Console.WriteLine(e.Message)
        End Try
        ' Create a new CspParameters object to specify
        ' a key container.
        Dim cspParams As New CspParameters()
        cspParams.KeyContainerName = "XML_ENC_RSA_KEY"
        ' Create a new RSA key and save it in the container.  This key will encrypt
        ' a symmetric key, which will then be encryped in the XML document.
        Dim rsaKey As New RSACryptoServiceProvider(cspParams)
        Try
            ' Encrypt the "creditcard" element.
            Encrypt(xmlDoc, "creditcard", "EncryptedElement1", rsaKey, "rsaKey")


            ' Save the XML document.
            xmlDoc.Save("test.xml")
            ' Display the encrypted XML to the console.
            Console.WriteLine("Encrypted XML:")
            Console.WriteLine()
            Console.WriteLine(xmlDoc.OuterXml)
            Decrypt(xmlDoc, rsaKey, "rsaKey")
            xmlDoc.Save("test.xml")
            ' Display the encrypted XML to the console.
            Console.WriteLine()
            Console.WriteLine("Decrypted XML:")
            Console.WriteLine()
            Console.WriteLine(xmlDoc.OuterXml)

        Catch e As Exception
            Console.WriteLine(e.Message)
        Finally
            ' Clear the RSA key.
            rsaKey.Clear()
        End Try


        Console.ReadLine()

    End Sub


    Public Shared Sub Encrypt(ByVal Doc As XmlDocument, ByVal EncryptionElement As String, ByVal
EncryptionElementID As String, ByVal Alg As RSA, ByVal KeyName As String)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If
        If EncryptionElement Is Nothing Then
            Throw New ArgumentNullException("EncryptionElement")
        End If
        If EncryptionElementID Is Nothing Then
            Throw New ArgumentNullException("EncryptionElementID")
        End If
        If Alg Is Nothing Then
            Throw New ArgumentNullException("Alg")
        End If
        If KeyName Is Nothing Then
            Throw New ArgumentNullException("KeyName")
        End If
        '////////////////////////////////////////////
        ' Find the specified element in the XmlDocument
```

```
' Find the specified element in the XmlDocument
' object and create a new XmlElemnt object.
'//////////////////////////////////////////////
Dim elementToEncrypt As XmlElement = Doc.GetElementsByTagName(EncryptionElement)(0) '

' Throw an XmlException if the element was not found.
If elementToEncrypt Is Nothing Then
    Throw New XmlException("The specified element was not found")
End If
Dim sessionKey As RijndaelManaged = Nothing

Try
    '//////////////////////////////////////////////
    ' Create a new instance of the EncryptedXml class
    ' and use it to encrypt the XmlElement with the
    ' a new random symmetric key.
    '//////////////////////////////////////////////
    ' Create a 256 bit Rijndael key.
    sessionKey = New RijndaelManaged()
    sessionKey.KeySize = 256
    Dim eXml As New EncryptedXml()

    Dim encryptedElement As Byte() = eXml.EncryptData(elementToEncrypt, sessionKey, False)
    '//////////////////////////////////////////////
    ' Construct an EncryptedData object and populate
    ' it with the desired encryption information.
    '//////////////////////////////////////////////
    Dim edElement As New EncryptedData()
    edElement.Type = EncryptedXml.XmlEncElementUrl
    edElement.Id = EncryptionElementID
    ' Create an EncryptionMethod element so that the
    ' receiver knows which algorithm to use for decryption.
    edElement.EncryptionMethod = New EncryptionMethod(EncryptedXml.XmlEncAES256Url)
    ' Encrypt the session key and add it to an EncryptedKey element.
    Dim ek As New EncryptedKey()

    Dim encryptedKey As Byte() = EncryptedXml.EncryptKey(sessionKey.Key, Alg, False)

    ek.CipherData = New CipherData(encryptedKey)

    ek.EncryptionMethod = New EncryptionMethod(EncryptedXml.XmlEncRSA15Url)
    ' Create a new DataReference element
    ' for the KeyInfo element.  This optional
    ' element specifies which EncryptedData
    ' uses this key.  An XML document can have
    ' multiple EncryptedData elements that use
    ' different keys.
    Dim dRef As New DataReference()

    ' Specify the EncryptedData URI.
    dRef.Uri = "#" + EncryptionElementID

    ' Add the DataReference to the EncryptedKey.
    ek.AddReference(dRef)
    ' Add the encrypted key to the
    ' EncryptedData object.
    edElement.KeyInfo.AddClause(New KeyInfoEncryptedKey(ek))
    ' Set the KeyInfo element to specify the
    ' name of the RSA key.
    ' Create a new KeyInfoName element.
    Dim kin As New KeyInfoName()

    ' Specify a name for the key.
    kin.Value = KeyName

    ' Add the KeyInfoName element to the
    ' EncryptedKey object.
    ek.KeyInfo.AddClause(kin)
    ' Add the encrypted element data to the
```

```
            ' EncryptedData object.
            edElement.CipherData.CipherValue = encryptedElement
            '////////////////////////////////////////////////
            ' Replace the element from the original XmlDocument
            ' object with the EncryptedData element.
            '////////////////////////////////////////////////
            EncryptedXml.ReplaceElement(elementToEncrypt, edElement, False)
        Catch e As Exception
            ' re-throw the exception.
            Throw e
        Finally
            If Not (sessionKey Is Nothing) Then
                sessionKey.Clear()
            End If
        End Try

    End Sub



    Public Shared Sub Decrypt(ByVal Doc As XmlDocument, ByVal Alg As RSA, ByVal KeyName As String)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If
        If Alg Is Nothing Then
            Throw New ArgumentNullException("Alg")
        End If
        If KeyName Is Nothing Then
            Throw New ArgumentNullException("KeyName")
        End If
        ' Create a new EncryptedXml object.
        Dim exml As New EncryptedXml(Doc)

        ' Add a key-name mapping.
        ' This method can only decrypt documents
        ' that present the specified key name.
        exml.AddKeyNameMapping(KeyName, Alg)

        ' Decrypt the element.
        exml.DecryptDocument()

    End Sub
End Class
```

Compilar el código

- Para compilar este ejemplo, debe incluir una referencia a `System.Security.dll`.
- Incluya los siguientes espacios de nombres: System.Xml, System.Security.Cryptography y System.Security.Cryptography.Xml.

Seguridad de .NET Framework

No almacene nunca una clave criptográfica simétrica en texto sin formato ni transfiera una clave simétrica entre equipos en texto sin formato. Tampoco debe almacenar ni transferir nunca la clave privada de un par de claves asimétricas en texto sin formato. Para obtener más información acerca de las claves criptográficas simétricas y asimétricas, vea generar claves para cifrado y descifrado.

No inserte nunca una clave directamente en el código fuente. Las claves incrustadas se pueden leer fácilmente desde un ensamblado mediante Ildasm. exe (desensamblador de IL) o abriendo el ensamblado en un editor de texto como el Bloc de notas.

Cuando termine de usar una clave criptográfica, bórrela de la memoria estableciendo cada byte en cero o llamando al método Clear de la clase criptográfica administrada. A veces, las claves criptográficas se pueden leer desde la memoria con un depurador o desde un disco duro si la ubicación de memoria se pagina en el disco.

Consulte también

- System.Security.Cryptography.Xml
- Procedimiento para descifrar elementos XML con claves asimétricas

Procedimiento para descifrar elementos XML con claves asimétricas

02/07/2020 • 8 minutes to read • Edit Online

Puede usar las clases en el espacio de nombres System.Security.Cryptography.Xml para cifrar y descifrar un elemento dentro de un documento XML. El cifrado XML es un método estándar para intercambiar o almacenar datos XML cifrados sin preocuparse de que los datos puedan leerse con facilidad. Para obtener más información sobre el estándar de cifrado XML, vea la recomendación del World Wide Web Consortium (W3C) sintaxis y procesamiento de firmas XML.

El ejemplo de este procedimiento descifra un elemento XML cifrado mediante los métodos descritos en Cómo: cifrar elementos XML con claves asimétricas. Busca un elemento de `EncryptedData` > de <, descifra el elemento y, a continuación, reemplaza el elemento por el elemento XML de texto sin formato original.

En este ejemplo se descifra un elemento XML mediante dos claves. Recupera una clave privada RSA generada previamente a partir de un contenedor de claves y, a continuación, usa la clave RSA para descifrar una clave de sesión almacenada en el `EncryptedKey` elemento <> del `EncryptedData` elemento> <. Luego, el ejemplo usa la clave de sesión para descifrar el elemento XML.

Este ejemplo resulta adecuado en aquellas situaciones en las que varias aplicaciones tienen que compartir datos cifrados o en las que una aplicación tiene que guardar datos cifrados entre los intervalos en los que se ejecuta.

Para descifrar un elemento XML con una clave asimétrica

1. Cree un objeto CspParameters y especifique el nombre del contenedor de claves.

```
CspParameters cspParams = new CspParameters();
cspParams.KeyContainerName = "XML_ENC_RSA_KEY";
```

```
Dim cspParams As New CspParameters()
cspParams.KeyContainerName = "XML_ENC_RSA_KEY"
```

2. Recupere una clave asimétrica previamente generada desde el contenedor mediante el objeto RSACryptoServiceProvider. La clave se recupera automáticamente del contenedor de claves al pasar el objeto CspParameters al constructor RSACryptoServiceProvider.

```
RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);
```

```
Dim rsaKey As New RSACryptoServiceProvider(cspParams)
```

3. Cree un objeto EncryptedXml nuevo para descifrar el documento.

```
// Create a new EncryptedXml object.
EncryptedXml exml = new EncryptedXml(Doc);
```

```
' Create a new EncryptedXml object.
Dim exml As New EncryptedXml(Doc)
```

4. Agregue una asignación de clave/nombre para asociar la clave RSA al elemento del documento que se debe descifrar. Debe usar el mismo nombre para la clave que usó al cifrar el documento. Tenga en cuenta que este nombre es independiente del que haya usado para identificar la clave en el contenedor de claves especificado en el paso 1.

```
exml.AddKeyNameMapping(KeyName, Alg);
```

```
exml.AddKeyNameMapping(KeyName, Alg)
```

5. Llame al DecryptDocument método para descifrar el elemento de> de < `EncryptedData` . Este método usa la clave RSA para descifrar la clave de sesión y la usa automáticamente para descifrar el elemento XML. También reemplaza automáticamente el elemento < `EncryptedData` > por el texto sin formato original.

```
exml.DecryptDocument();
```

```
exml.DecryptDocument()
```

6. Guarde el documento XML.

```
xmlDoc.Save("test.xml");
```

```
xmlDoc.Save("test.xml")
```

Ejemplo

En este ejemplo se supone que un archivo llamado `test.xml` se encuentra en el mismo directorio que el programa compilado. También se supone que `test.xml` contiene un elemento XML cifrado mediante las técnicas descritas en Cómo: cifrar elementos XML con claves asimétricas.

```
using System;
using System.Xml;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;

class Program
{
    static void Main(string[] args)
    {

        // Create an XmlDocument object.
        XmlDocument xmlDoc = new XmlDocument();

        // Load an XML file into the XmlDocument object.
        try
        {
            xmlDoc.PreserveWhitespace = true;
            xmlDoc.Load("test.xml");
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        CspParameters cspParams = new CspParameters();
```

```
        cspParams.KeyContainerName = "XML_ENC_RSA_KEY";

        // Get the RSA key from the key container.  This key will decrypt
        // a symmetric key that was imbedded in the XML document.
        RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

        try
        {

            // Decrypt the elements.
            Decrypt(xmlDoc, rsaKey, "rsaKey");

            // Save the XML document.
            xmlDoc.Save("test.xml");

            // Display the encrypted XML to the console.
            Console.WriteLine();
            Console.WriteLine("Decrypted XML:");
            Console.WriteLine();
            Console.WriteLine(xmlDoc.OuterXml);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        finally
        {
            // Clear the RSA key.
            rsaKey.Clear();
        }

        Console.ReadLine();
    }

    public static void Decrypt(XmlDocument Doc, RSA Alg, string KeyName)
    {
        // Check the arguments.
        if (Doc == null)
            throw new ArgumentNullException("Doc");
        if (Alg == null)
            throw new ArgumentNullException("Alg");
        if (KeyName == null)
            throw new ArgumentNullException("KeyName");
        // Create a new EncryptedXml object.
        EncryptedXml exml = new EncryptedXml(Doc);

        // Add a key-name mapping.
        // This method can only decrypt documents
        // that present the specified key name.
        exml.AddKeyNameMapping(KeyName, Alg);

        // Decrypt the element.
        exml.DecryptDocument();
    }
}
```

```
Imports System.Xml
Imports System.Security.Cryptography
Imports System.Security.Cryptography.Xml



Module Program

    Sub Main(ByVal args() As String)

        ' Create an XmlDocument object.
```

```
        Dim xmlDoc As New XmlDocument()

        ' Load an XML file into the XmlDocument object.
        Try
            xmlDoc.PreserveWhitespace = True
            xmlDoc.Load("test.xml")
        Catch e As Exception
            Console.WriteLine(e.Message)
        End Try
        Dim cspParams As New CspParameters()
        cspParams.KeyContainerName = "XML_ENC_RSA_KEY"
        ' Get the RSA key from the key container.  This key will decrypt
        ' a symmetric key that was imbedded in the XML document.
        Dim rsaKey As New RSACryptoServiceProvider(cspParams)
        Try

            ' Decrypt the elements.
            Decrypt(xmlDoc, rsaKey, "rsaKey")

            ' Save the XML document.
            xmlDoc.Save("test.xml")
            ' Display the encrypted XML to the console.
            Console.WriteLine()
            Console.WriteLine("Decrypted XML:")
            Console.WriteLine()
            Console.WriteLine(xmlDoc.OuterXml)
        Catch e As Exception
            Console.WriteLine(e.Message)
        Finally
            ' Clear the RSA key.
            rsaKey.Clear()
        End Try


        Console.ReadLine()

    End Sub



    Sub Decrypt(ByVal Doc As XmlDocument, ByVal Alg As RSA, ByVal KeyName As String)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If
        If Alg Is Nothing Then
            Throw New ArgumentNullException("Alg")
        End If
        If KeyName Is Nothing Then
            Throw New ArgumentNullException("KeyName")
        End If
        ' Create a new EncryptedXml object.
        Dim exml As New EncryptedXml(Doc)
        ' Add a key-name mapping.
        ' This method can only decrypt documents
        ' that present the specified key name.
        exml.AddKeyNameMapping(KeyName, Alg)
        ' Decrypt the element.
        exml.DecryptDocument()
    End Sub
End Module
```

Compilar el código

- Para compilar este ejemplo, debe incluir una referencia a `System.Security.dll`.

- Incluya los siguientes espacios de nombres: System.Xml, System.Security.Cryptography y System.Security.Cryptography.Xml.

Seguridad de .NET Framework

No almacene nunca una clave criptográfica simétrica en texto sin formato ni transfiera una clave simétrica entre equipos en texto sin formato. Tampoco debe almacenar ni transferir nunca la clave privada de un par de claves asimétricas en texto sin formato. Para obtener más información acerca de las claves criptográficas simétricas y asimétricas, vea generar claves para cifrado y descifrado.

No inserte nunca una clave directamente en el código fuente. Las claves incrustadas se pueden leer fácilmente desde un ensamblado mediante Ildasm. exe (desensamblador de IL) o abriendo el ensamblado en un editor de texto como el Bloc de notas.

Cuando termine de usar una clave criptográfica, bórrela de la memoria estableciendo cada byte en cero o llamando al método Clear de la clase criptográfica administrada. A veces, las claves criptográficas se pueden leer desde la memoria con un depurador o desde un disco duro si la ubicación de memoria se pagina en el disco.

Consulte también

- System.Security.Cryptography.Xml
- Procedimiento para cifrar elementos XML con claves asimétricas

Procedimiento para cifrar elementos XML con certificados X.509

02/07/2020 • 10 minutes to read • Edit Online

Puede usar las clases en el espacio de nombres System.Security.Cryptography.Xml para cifrar un elemento dentro de un documento XML. El cifrado XML es un método estándar para intercambiar o almacenar datos XML cifrados sin preocuparse de que los datos puedan leerse con facilidad. Para obtener más información sobre el estándar de cifrado XML, consulte la especificación de World Wide Web Consortium (W3C) para el cifrado XML ubicado en https://www.w3.org/TR/xmldsig-core/ .

Puede usar el cifrado de XML para reemplazar cualquier elemento o documento XML con un elemento < `EncryptedData` > que contenga los datos XML cifrados. El elemento < `EncryptedData` > puede contener subelementos con información sobre las claves y los procesos usados durante el cifrado. El cifrado XML permite que un documento contenga varios elementos cifrados y permite cifrar varias veces un elemento. El ejemplo de código de este procedimiento muestra cómo crear un elemento < `EncryptedData` > junto con otros subelementos que se pueden usar posteriormente durante el descifrado.

En este ejemplo se cifra un elemento XML mediante dos claves. Se genera un certificado X.509 de prueba mediante la herramienta de creación de certificados (Makecert.exe) y se guarda el certificado en un almacén de certificados. A continuación, en el ejemplo se recupera el certificado mediante programación y se usa para cifrar un elemento XML mediante el método Encrypt. Internamente, el método Encrypt crea una clave de sesión independiente y la usa para cifrar el documento XML. Este método cifra la clave de sesión y la guarda junto con el XML cifrado dentro de un nuevo elemento < `EncryptedData` >.

Para descifrar el elemento XML, basta con llamar al método DecryptDocument, que recupera automáticamente el certificado X.509 del almacén y realiza las operaciones de descifrado necesarias. Para más información sobre la manera de descifrar un elemento XML cifrado mediante este procedimiento, vea Cómo: Descifrar elementos XML con certificados X.509.

Este ejemplo resulta adecuado en aquellas situaciones en las que varias aplicaciones tienen que compartir datos cifrados o en las que una aplicación tiene que guardar datos cifrados entre los intervalos en los que se ejecuta.

Para cifrar un elemento XML con un certificado X.509

1. Use la herramienta de creación de certificados (Makecert.exe) para generar un certificado X.509 de prueba y colocarlo en el almacén de usuario local. Debe generar una clave de intercambio y debe hacerla exportable. Ejecute el siguiente comando:

   ```
   makecert -r -pe -n "CN=XML_ENC_TEST_CERT" -b 01/01/2005 -e 01/01/2010 -sky exchange -ss my
   ```

2. Cree un objeto X509Store e inicialícelo para abrir el almacén del usuario actual.

   ```
   X509Store store = new X509Store(StoreLocation.CurrentUser);
   ```

   ```
   Dim store As New X509Store(StoreLocation.CurrentUser)
   ```

3. Abra el almacén en modo de solo lectura.

```
store.Open(OpenFlags.ReadOnly);
```

```
store.Open(OpenFlags.ReadOnly)
```

4. Inicialice una X509Certificate2Collection con todos los certificados del almacén.

```
X509Certificate2Collection certCollection = store.Certificates;
```

```
Dim certCollection As X509Certificate2Collection = store.Certificates
```

5. Enumere los certificados del almacén y busque el certificado con el nombre adecuado. En este ejemplo, el certificado se llama `"CN=XML_ENC_TEST_CERT"`.

```
X509Certificate2 cert = null;

// Loop through each certificate and find the certificate
// with the appropriate name.
foreach (X509Certificate2 c in certCollection)
{
    if (c.Subject == "CN=XML_ENC_TEST_CERT")
    {
        cert = c;

        break;
    }
}
```

```
Dim cert As X509Certificate2 = Nothing

' Loop through each certificate and find the certificate
' with the appropriate name.
Dim c As X509Certificate2
For Each c In certCollection
    If c.Subject = "CN=XML_ENC_TEST_CERT" Then
        cert = c

        Exit For
    End If
Next c
```

6. Cierre el almacén después de localizar el certificado.

```
store.Close();
```

```
store.Close()
```

7. Cree un objeto XmlDocument cargando un archivo XML del disco. El objeto XmlDocument contiene el elemento XML que se va a cifrar.

```
XmlDocument xmlDoc = new XmlDocument();
```

```
Dim xmlDoc As New XmlDocument()
```

8. Busque el elemento especificado en el objeto XmlDocument y cree un objeto XmlElement nuevo para representar el elemento que desea cifrar. En este ejemplo, el elemento `"creditcard"` está cifrado.

```
XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementToEncrypt)[0] as XmlElement;
```

```
Dim elementToEncrypt As XmlElement = Doc.GetElementsByTagName(ElementToEncryptName)(0)
```

9. Cree una nueva instancia de la clase EncryptedXml y úsela para cifrar el elemento especificado mediante el certificado X.509. El método Encrypt devuelve el elemento cifrado como un objeto EncryptedData.

```
EncryptedXml eXml = new EncryptedXml();

// Encrypt the element.
EncryptedData edElement = eXml.Encrypt(elementToEncrypt, Cert);
```

```
Dim eXml As New EncryptedXml()

' Encrypt the element.
Dim edElement As EncryptedData = eXml.Encrypt(elementToEncrypt, Cert)
```

10. Reemplace el elemento del objeto XmlDocument original por el elemento EncryptedData.

```
EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);
```

```
EncryptedXml.ReplaceElement(elementToEncrypt, edElement, False)
```

11. Guarde el objeto XmlDocument.

```
xmlDoc.Save("test.xml");
```

```
xmlDoc.Save("test.xml")
```

Ejemplo

En este ejemplo se supone que un archivo llamado `"test.xml"` se encuentra en el mismo directorio que el programa compilado. También se supone que `"test.xml"` contiene un elemento `"creditcard"`. Puede colocar el siguiente código XML en un archivo llamado `test.xml` y usarlo con este ejemplo.

```
<root>
    <creditcard>
        <number>19834209</number>
        <expiry>02/02/2002</expiry>
    </creditcard>
</root>
```

```
using System;
using System.Xml;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
using System.Security.Cryptography.X509Certificates;

class Program
{
    static void Main(string[] args)
    {
        try
        {
            // Create an XmlDocument object.
            XmlDocument xmlDoc = new XmlDocument();

            // Load an XML file into the XmlDocument object.
            xmlDoc.PreserveWhitespace = true;
            xmlDoc.Load("test.xml");

            // Open the X.509 "Current User" store in read only mode.
            X509Store store = new X509Store(StoreLocation.CurrentUser);
            store.Open(OpenFlags.ReadOnly);

            // Place all certificates in an X509Certificate2Collection object.
            X509Certificate2Collection certCollection = store.Certificates;

            X509Certificate2 cert = null;

            // Loop through each certificate and find the certificate
            // with the appropriate name.
            foreach (X509Certificate2 c in certCollection)
            {
                if (c.Subject == "CN=XML_ENC_TEST_CERT")
                {
                    cert = c;

                    break;
                }
            }

            if (cert == null)
            {
                throw new CryptographicException("The X.509 certificate could not be found.");
            }

            // Close the store.
            store.Close();

            // Encrypt the "creditcard" element.
            Encrypt(xmlDoc, "creditcard", cert);

            // Save the XML document.
            xmlDoc.Save("test.xml");

            // Display the encrypted XML to the console.
            Console.WriteLine("Encrypted XML:");
            Console.WriteLine();
            Console.WriteLine(xmlDoc.OuterXml);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }

    public static void Encrypt(XmlDocument Doc, string ElementToEncrypt, X509Certificate2 Cert)
    {
        // Check the arguments.
        if (Doc == null)
```

```
        if (Doc == null)
            throw new ArgumentNullException("Doc");
        if (ElementToEncrypt == null)
            throw new ArgumentNullException("ElementToEncrypt");
        if (Cert == null)
            throw new ArgumentNullException("Cert");

        ////////////////////////////////////////////////
        // Find the specified element in the XmlDocument
        // object and create a new XmlElemnt object.
        ////////////////////////////////////////////////

        XmlElement elementToEncrypt = Doc.GetElementsByTagName(ElementToEncrypt)[0] as XmlElement;
        // Throw an XmlException if the element was not found.
        if (elementToEncrypt == null)
        {
            throw new XmlException("The specified element was not found");
        }

        ////////////////////////////////////////////////
        // Create a new instance of the EncryptedXml class
        // and use it to encrypt the XmlElement with the
        // X.509 Certificate.
        ////////////////////////////////////////////////

        EncryptedXml eXml = new EncryptedXml();

        // Encrypt the element.
        EncryptedData edElement = eXml.Encrypt(elementToEncrypt, Cert);

        ////////////////////////////////////////////////
        // Replace the element from the original XmlDocument
        // object with the EncryptedData element.
        ////////////////////////////////////////////////
        EncryptedXml.ReplaceElement(elementToEncrypt, edElement, false);
    }
}
```

```
Imports System.Xml
Imports System.Security.Cryptography
Imports System.Security.Cryptography.Xml
Imports System.Security.Cryptography.X509Certificates



Module Program

    Sub Main(ByVal args() As String)
        Try
            ' Create an XmlDocument object.
            Dim xmlDoc As New XmlDocument()
            ' Load an XML file into the XmlDocument object.
            xmlDoc.PreserveWhitespace = True
            xmlDoc.Load("test.xml")

            ' Open the X.509 "Current User" store in read only mode.
            Dim store As New X509Store(StoreLocation.CurrentUser)
            store.Open(OpenFlags.ReadOnly)
            ' Place all certificates in an X509Certificate2Collection object.
            Dim certCollection As X509Certificate2Collection = store.Certificates
            Dim cert As X509Certificate2 = Nothing

            ' Loop through each certificate and find the certificate
            ' with the appropriate name.
            Dim c As X509Certificate2
            For Each c In certCollection
                If c.Subject = "CN=XML_ENC_TEST_CERT" Then
                    cert = c
```

```
                    cert = c

                    Exit For
                End If
            Next c
            If cert Is Nothing Then
                Throw New CryptographicException("The X.509 certificate could not be found.")
            End If

            ' Close the store.
            store.Close()
            ' Encrypt the "creditcard" element.
            Encrypt(xmlDoc, "creditcard", cert)

            ' Save the XML document.
            xmlDoc.Save("test.xml")
            ' Display the encrypted XML to the console.
            Console.WriteLine("Encrypted XML:")
            Console.WriteLine()
            Console.WriteLine(xmlDoc.OuterXml)

        Catch e As Exception
            Console.WriteLine(e.Message)
        End Try

    End Sub


    Sub Encrypt(ByVal Doc As XmlDocument, ByVal ElementToEncryptName As String, ByVal Cert As
X509Certificate2)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If
        If ElementToEncryptName Is Nothing Then
            Throw New ArgumentNullException("ElementToEncrypt")
        End If
        If Cert Is Nothing Then
            Throw New ArgumentNullException("Cert")
        End If
        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' Find the specified element in the XmlDocument
        ' object and create a new XmlElemnt object.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        Dim elementToEncrypt As XmlElement = Doc.GetElementsByTagName(ElementToEncryptName)(0)

        ' Throw an XmlException if the element was not found.
        If elementToEncrypt Is Nothing Then
            Throw New XmlException("The specified element was not found")
        End If

        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' Create a new instance of the EncryptedXml class
        ' and use it to encrypt the XmlElement with the
        ' X.509 Certificate.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        Dim eXml As New EncryptedXml()

        ' Encrypt the element.
        Dim edElement As EncryptedData = eXml.Encrypt(elementToEncrypt, Cert)
        '''''''''''''''''''''''''''''''''''''''''''''''''
        ' Replace the element from the original XmlDocument
        ' object with the EncryptedData element.
        '''''''''''''''''''''''''''''''''''''''''''''''''
        EncryptedXml.ReplaceElement(elementToEncrypt, edElement, False)
    End Sub
End Module
```

Compilar el código

- Para compilar este ejemplo, debe incluir una referencia a `System.Security.dll`.
- Incluya los siguientes espacios de nombres: System.Xml, System.Security.Cryptography y System.Security.Cryptography.Xml.

Seguridad de .NET Framework

El certificado X.509 usado en este ejemplo se usa únicamente para pruebas. Las aplicaciones deben usar un certificado X.509 generado por una entidad de certificación de confianza o un certificado generado por Microsoft Windows Certificate Server.

Consulte también

- System.Security.Cryptography.Xml
- Procedimiento para descifrar elementos XML con certificados X.509

Procedimiento para descifrar elementos XML con certificados X.509

02/07/2020 • 5 minutes to read • Edit Online

Puede usar las clases en el espacio de nombres System.Security.Cryptography.Xml para cifrar y descifrar un elemento dentro de un documento XML. El cifrado XML es un método estándar para intercambiar o almacenar datos XML cifrados sin preocuparse de que los datos puedan leerse con facilidad. Para obtener más información sobre el estándar de cifrado XML, consulte la especificación de World Wide Web Consortium (W3C) para el cifrado XML ubicado en https://www.w3.org/TR/xmldsig-core/ .

En este ejemplo se descifra un elemento XML cifrado mediante los métodos descritos en: Cómo: cifrar elementos XML con certificados X. 509. Busca un elemento de `EncryptedData` > de <, descifra el elemento y, a continuación, reemplaza el elemento por el elemento XML de texto sin formato original.

El ejemplo de código de este procedimiento descifra un elemento XML mediante un certificado X.509 del almacén de certificados local de la cuenta de usuario actual. En el ejemplo se usa el DecryptDocument método para recuperar automáticamente el certificado X. 509 y descifrar una clave de sesión almacenada en el `EncryptedKey` elemento <> del `EncryptedData` elemento <>. Luego, el método DecryptDocument usa automáticamente la clave de sesión para descifrar el elemento XML.

Este ejemplo resulta adecuado en aquellas situaciones en las que varias aplicaciones tienen que compartir datos cifrados o en las que una aplicación tiene que guardar datos cifrados entre los intervalos en los que se ejecuta.

Para descifrar un elemento XML con un certificado X.509

1. Cree un objeto XmlDocument cargando un archivo XML del disco. El objeto XmlDocument contiene el elemento XML que se va a descifrar.

```
XmlDocument xmlDoc = new XmlDocument();
```

```
Dim xmlDoc As New XmlDocument()
```

2. Cree un objeto EncryptedXml nuevo pasando el objeto XmlDocument al constructor.

```
EncryptedXml exml = new EncryptedXml(Doc);
```

```
Dim exml As New EncryptedXml(Doc)
```

3. Descifre el documento XML mediante el método DecryptDocument.

```
exml.DecryptDocument();
```

```
exml.DecryptDocument()
```

4. Guarde el objeto XmlDocument.

```
xmlDoc.Save("test.xml");
```

```
xmlDoc.Save("test.xml")
```

Ejemplo

En este ejemplo se supone que un archivo llamado `"test.xml"` se encuentra en el mismo directorio que el programa compilado. También se supone que `"test.xml"` contiene un elemento `"creditcard"`. Puede colocar el siguiente código XML en un archivo llamado `test.xml` y usarlo con este ejemplo.

```
<root>
    <creditcard>
        <number>19834209</number>
        <expiry>02/02/2002</expiry>
    </creditcard>
</root>
```

```
using System;
using System.Xml;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
using System.Security.Cryptography.X509Certificates;

class Program
{
    static void Main(string[] args)
    {
        try
        {
            // Create an XmlDocument object.
            XmlDocument xmlDoc = new XmlDocument();

            // Load an XML file into the XmlDocument object.
            xmlDoc.PreserveWhitespace = true;
            xmlDoc.Load("test.xml");

            // Decrypt the document.
            Decrypt(xmlDoc);

            // Save the XML document.
            xmlDoc.Save("test.xml");

            // Display the decrypted XML to the console.
            Console.WriteLine("Decrypted XML:");
            Console.WriteLine();
            Console.WriteLine(xmlDoc.OuterXml);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }

    public static void Decrypt(XmlDocument Doc)
    {
        // Check the arguments.
        if (Doc == null)
            throw new ArgumentNullException("Doc");

        // Create a new EncryptedXml object.
        EncryptedXml exml = new EncryptedXml(Doc);

        // Decrypt the XML document.
        exml.DecryptDocument();
    }
}
```

```
Imports System.Xml
Imports System.Security.Cryptography
Imports System.Security.Cryptography.Xml
Imports System.Security.Cryptography.X509Certificates



Module Program

    Sub Main(ByVal args() As String)
        Try
            ' Create an XmlDocument object.
            Dim xmlDoc As New XmlDocument()
            ' Load an XML file into the XmlDocument object.
            xmlDoc.PreserveWhitespace = True
            xmlDoc.Load("test.xml")

            ' Decrypt the document.
            Decrypt(xmlDoc)

            ' Save the XML document.
            xmlDoc.Save("test.xml")
            ' Display the decrypted XML to the console.
            Console.WriteLine("Decrypted XML:")
            Console.WriteLine()
            Console.WriteLine(xmlDoc.OuterXml)

        Catch e As Exception
            Console.WriteLine(e.Message)
        End Try

    End Sub


    Sub Decrypt(ByVal Doc As XmlDocument)
        ' Check the arguments.
        If Doc Is Nothing Then
            Throw New ArgumentNullException("Doc")
        End If

        ' Create a new EncryptedXml object.
        Dim exml As New EncryptedXml(Doc)
        ' Decrypt the XML document.
        exml.DecryptDocument()
    End Sub
End Module
```

Compilar el código

- Para compilar este ejemplo, debe incluir una referencia a `System.Security.dll`.
- Incluya los siguientes espacios de nombres: System.Xml, System.Security.Cryptography y System.Security.Cryptography.Xml.

Seguridad de .NET Framework

El certificado X.509 usado en este ejemplo se usa únicamente para pruebas. Las aplicaciones deben usar un certificado X.509 generado por una entidad de certificación de confianza o un certificado generado por Microsoft Windows Certificate Server.

Consulte también

- System.Security.Cryptography.Xml
- Procedimiento para cifrar elementos XML con certificados X.509

Procedimiento para firmar documentos XML con firmas digitales

02/07/2020 • 8 minutes to read • Edit Online

Puede usar las clases del espacio de nombres System.Security.Cryptography.Xml para firmar un documento XML o parte de un documento XML con una firma digital. Las firmas XML digitales (XMLDSIG) permiten comprobar que los datos no se modificaron después de firmarlos. Para obtener más información sobre el estándar XMLDSIG, consulte la recomendación de la firma XMLdel World Wide Web Consortium (W3C) y el procesamiento.

En el ejemplo de código de este procedimiento se muestra cómo firmar digitalmente un documento XML completo y adjuntar la firma al documento en un <`Signature` elemento>. En el ejemplo se crea una clave de firma RSA, se agrega la clave a un contenedor de claves seguro y, a continuación, se usa la clave para firmar digitalmente un documento XML. La clave se puede recuperar para comprobar la firma digital XML, o bien se puede usar para firmar otro documento XML.

Para obtener información sobre cómo comprobar una firma digital XML creada con este procedimiento, consulte Cómo: comprobar las firmas digitales de documentos XML.

Para firmar digitalmente un documento XML

1. Cree un objeto CspParameters y especifique el nombre del contenedor de claves.

```
CspParameters cspParams = new CspParameters();
cspParams.KeyContainerName = "XML_DSIG_RSA_KEY";
```

```
Dim cspParams As New CspParameters()
cspParams.KeyContainerName = "XML_DSIG_RSA_KEY"
```

2. Genere una clave asimétrica mediante la clase RSACryptoServiceProvider. La clave se guarda automáticamente en el contenedor de claves al pasar el objeto CspParameters al constructor de la clase RSACryptoServiceProvider. Esta clave se usará para firmar el documento XML.

```
RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);
```

```
Dim rsaKey As New RSACryptoServiceProvider(cspParams)
```

3. Cree un objeto XmlDocument cargando un archivo XML del disco. El objeto XmlDocument contiene el elemento XML que se va a cifrar.

```
XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.
xmlDoc.PreserveWhitespace = true;
xmlDoc.Load("test.xml");
```

```
Dim xmlDoc As New XmlDocument()

' Load an XML file into the XmlDocument object.
xmlDoc.PreserveWhitespace = True
xmlDoc.Load("test.xml")
```

4. Cree un nuevo objeto SignedXml y pásele el objeto XmlDocument.

```
SignedXml signedXml = new SignedXml(xmlDoc);
```

```
Dim signedXml As New SignedXml(xmlDoc)
```

5. Agregue la clave RSA de firma al objeto SignedXml.

```
signedXml.SigningKey = rsaKey;
```

```
signedXml.SigningKey = rsaKey
```

6. Cree un objeto Reference que describa qué se va a firmar. Para firmar el documento completo, establezca la propiedad Uri en `""`.

```
// Create a reference to be signed.
Reference reference = new Reference();
reference.Uri = "";
```

```
' Create a reference to be signed.
Dim reference As New Reference()
reference.Uri = ""
```

7. Agregue un objeto XmlDsigEnvelopedSignatureTransform al objeto Reference. Una transformación permite al comprobador representar los datos XML de un modo idéntico al que usó el firmante. Los datos XML se pueden representar de maneras diferentes, por lo que este paso es vital para la comprobación.

```
XmlDsigEnvelopedSignatureTransform env = new XmlDsigEnvelopedSignatureTransform();
reference.AddTransform(env);
```

```
Dim env As New XmlDsigEnvelopedSignatureTransform()
reference.AddTransform(env)
```

8. Agregue el objeto Reference al objeto SignedXml.

```
signedXml.AddReference(reference);
```

```
signedXml.AddReference(reference)
```

9. Calcule la firma llamando al método ComputeSignature.

```
signedXml.ComputeSignature();
```

```
signedXml.ComputeSignature()
```

10. Recuperar la representación XML de la firma (un `Signature` elemento <>) y guardarla en un nuevo XmlElement objeto.

```
XmlElement xmlDigitalSignature = signedXml.GetXml();
```

```
Dim xmlDigitalSignature As XmlElement = signedXml.GetXml()
```

11. Anexe el elemento al objeto XmlDocument.

```
xmlDoc.DocumentElement.AppendChild(xmlDoc.ImportNode(xmlDigitalSignature, true));
```

```
xmlDoc.DocumentElement.AppendChild(xmlDoc.ImportNode(xmlDigitalSignature, True))
```

12. Guarde el documento.

```
xmlDoc.Save("test.xml");
```

```
xmlDoc.Save("test.xml")
```

Ejemplo

En este ejemplo se supone que un archivo llamado `test.xml` se encuentra en el mismo directorio que el programa compilado. Puede colocar el siguiente código XML en un archivo llamado `test.xml` y usarlo con este ejemplo.

```
<root>
    <creditcard>
        <number>19834209</number>
        <expiry>02/02/2002</expiry>
    </creditcard>
</root>
```

```
using System;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
using System.Xml;

public class SignXML
{
    public static void Main(String[] args)
    {
        try
        {
            // Create a new CspParameters object to specify
            // a key container.
            CspParameters cspParams = new CspParameters();
```

```
            cspParams.KeyContainerName = "XML_DSIG_RSA_KEY";

            // Create a new RSA signing key and save it in the container.
            RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

            // Create a new XML document.
            XmlDocument xmlDoc = new XmlDocument();

            // Load an XML file into the XmlDocument object.
            xmlDoc.PreserveWhitespace = true;
            xmlDoc.Load("test.xml");

            // Sign the XML document.
            SignXml(xmlDoc, rsaKey);

            Console.WriteLine("XML file signed.");

            // Save the document.
            xmlDoc.Save("test.xml");
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }

    // Sign an XML file.
    // This document cannot be verified unless the verifying
    // code has the key with which it was signed.
    public static void SignXml(XmlDocument xmlDoc, RSA rsaKey)
    {
        // Check arguments.
        if (xmlDoc == null)
            throw new ArgumentException(nameof(xmlDoc));
        if (rsaKey == null)
            throw new ArgumentException(nameof(rsaKey));

        // Create a SignedXml object.
        SignedXml signedXml = new SignedXml(xmlDoc);

        // Add the key to the SignedXml document.
        signedXml.SigningKey = rsaKey;

        // Create a reference to be signed.
        Reference reference = new Reference();
        reference.Uri = "";

        // Add an enveloped transformation to the reference.
        XmlDsigEnvelopedSignatureTransform env = new XmlDsigEnvelopedSignatureTransform();
        reference.AddTransform(env);

        // Add the reference to the SignedXml object.
        signedXml.AddReference(reference);

        // Compute the signature.
        signedXml.ComputeSignature();

        // Get the XML representation of the signature and save
        // it to an XmlElement object.
        XmlElement xmlDigitalSignature = signedXml.GetXml();

        // Append the element to the XML document.
        xmlDoc.DocumentElement.AppendChild(xmlDoc.ImportNode(xmlDigitalSignature, true));
    }
}
```

```
Imports System.Security.Cryptography
Imports System.Security.Cryptography.Xml
Imports System.Xml

Module SignXML
    Sub Main(ByVal args() As String)
        Try
            ' Create a new CspParameters object to specify
            ' a key container.
            Dim cspParams As New CspParameters()
            cspParams.KeyContainerName = "XML_DSIG_RSA_KEY"
            ' Create a new RSA signing key and save it in the container.
            Dim rsaKey As New RSACryptoServiceProvider(cspParams)
            ' Create a new XML document.
            Dim xmlDoc As New XmlDocument()

            ' Load an XML file into the XmlDocument object.
            xmlDoc.PreserveWhitespace = True
            xmlDoc.Load("test.xml")
            ' Sign the XML document.
            SignXml(xmlDoc, rsaKey)

            Console.WriteLine("XML file signed.")

            ' Save the document.
            xmlDoc.Save("test.xml")
        Catch e As Exception
            Console.WriteLine(e.Message)
        End Try
    End Sub

    ' Sign an XML file.
    ' This document cannot be verified unless the verifying
    ' code has the key with which it was signed.
    Sub SignXml(ByVal xmlDoc As XmlDocument, ByVal rsaKey As RSA)
        ' Check arguments.
        If xmlDoc Is Nothing Then
            Throw New ArgumentException(NameOf(xmlDoc))
        End If
        If rsaKey Is Nothing Then
            Throw New ArgumentException(NameOf(rsaKey))
        End If
        ' Create a SignedXml object.
        Dim signedXml As New SignedXml(xmlDoc)
        ' Add the key to the SignedXml document.
        signedXml.SigningKey = rsaKey
        ' Create a reference to be signed.
        Dim reference As New Reference()
        reference.Uri = ""
        ' Add an enveloped transformation to the reference.
        Dim env As New XmlDsigEnvelopedSignatureTransform()
        reference.AddTransform(env)
        ' Add the reference to the SignedXml object.
        signedXml.AddReference(reference)
        ' Compute the signature.
        signedXml.ComputeSignature()
        ' Get the XML representation of the signature and save
        ' it to an XmlElement object.
        Dim xmlDigitalSignature As XmlElement = signedXml.GetXml()
        ' Append the element to the XML document.
        xmlDoc.DocumentElement.AppendChild(xmlDoc.ImportNode(xmlDigitalSignature, True))
    End Sub
End Module
```

Compilar el código

- Para compilar este ejemplo, debe incluir una referencia a `System.Security.dll`.
- Incluya los siguientes espacios de nombres: System.Xml, System.Security.Cryptography y System.Security.Cryptography.Xml.

Seguridad de .NET Framework

Nunca almacene ni transfiera la clave privada de un par de claves asimétricas en texto sin formato. Para obtener más información acerca de las claves criptográficas simétricas y asimétricas, vea generar claves para cifrado y descifrado.

No inserte nunca una clave privada directamente en el código fuente. Las claves incrustadas se pueden leer fácilmente desde un ensamblado mediante Ildasm. exe (desensamblador de IL) o abriendo el ensamblado en un editor de texto como el Bloc de notas.

Vea también

- System.Security.Cryptography.Xml
- Procedimiento para comprobar las firmas digitales de documentos XML

Procedimiento para comprobar las firmas digitales de documentos XML

02/07/2020 • 7 minutes to read • Edit Online

Puede usar las clases del espacio de nombres System.Security.Cryptography.Xml para comprobar datos XML firmados con una firma digital. Las firmas XML digitales (XMLDSIG) permiten comprobar que los datos no se modificaron después de firmarlos. Para obtener más información sobre el estándar XMLDSIG, consulte la especificación de World Wide Web Consortium (W3C) en https://www.w3.org/TR/xmldsig-core/ .

En el ejemplo de código de este procedimiento se muestra cómo comprobar una firma digital XML incluida en un `<Signature` elemento>. En el ejemplo se recupera una clave pública RSA de un contenedor de claves y se usa la clave para comprobar la firma.

Para obtener información sobre cómo crear una firma digital que se pueda comprobar mediante esta técnica, consulte Cómo: firmar documentos XML con firmas digitales.

Para comprobar la firma digital de un documento XML

1. Para comprobar el documento, debe usar la misma clave asimétrica que se empleó para la firma. Cree un objeto CspParameters y especifique el nombre del contenedor de claves usado para la firma.

```
CspParameters cspParams = new CspParameters();
cspParams.KeyContainerName = "XML_DSIG_RSA_KEY";
```

```
Dim cspParams As New CspParameters()
cspParams.KeyContainerName = "XML_DSIG_RSA_KEY"
```

2. Recupere la clave pública mediante la clase RSACryptoServiceProvider. La clave se carga automáticamente desde el contenedor de claves por nombre al pasar el objeto CspParameters al constructor de la clase RSACryptoServiceProvider.

```
RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);
```

```
Dim rsaKey As New RSACryptoServiceProvider(cspParams)
```

3. Cree un objeto XmlDocument cargando un archivo XML del disco. El objeto XmlDocument contiene el documento XML firmado que se va a comprobar.

```
XmlDocument xmlDoc = new XmlDocument();

// Load an XML file into the XmlDocument object.
xmlDoc.PreserveWhitespace = true;
xmlDoc.Load("test.xml");
```

```
Dim xmlDoc As New XmlDocument()

' Load an XML file into the XmlDocument object.
xmlDoc.PreserveWhitespace = True
xmlDoc.Load("test.xml")
```

4. Cree un nuevo objeto SignedXml y pásele el objeto XmlDocument.

```
SignedXml signedXml = new SignedXml(xmlDoc);
```

```
Dim signedXml As New SignedXml(xmlDoc)
```

5. Busque el `signature` elemento <> y cree un nuevo XmlNodeList objeto.

```
XmlNodeList nodeList = xmlDoc.GetElementsByTagName("Signature");
```

```
Dim nodeList As XmlNodeList = xmlDoc.GetElementsByTagName("Signature")
```

6. Cargue el XML del primer < `signature` elemento> en el SignedXml objeto.

```
signedXml.LoadXml((XmlElement)nodeList[0]);
```

```
signedXml.LoadXml(CType(nodeList(0), XmlElement))
```

7. Compruebe la firma con el método CheckSignature y la clave pública RSA. Este método devuelve un valor booleano que indica éxito o error.

```
return signedXml.CheckSignature(key);
```

```
Return signedXml.CheckSignature(key)
```

Ejemplo

En este ejemplo se supone que un archivo llamado `"test.xml"` se encuentra en el mismo directorio que el programa compilado. El `"test.xml"` archivo se debe firmar mediante las técnicas descritas en Cómo: firmar documentos XML con firmas digitales.

```
using System;
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
using System.Xml;

public class VerifyXML
{

    public static void Main(String[] args)
    {
        try
        {
            // Create a new CspParameters object to specify
```

```
            // Create a new CspParameters object to specify
            // a key container.
            CspParameters cspParams = new CspParameters();
            cspParams.KeyContainerName = "XML_DSIG_RSA_KEY";

            // Create a new RSA signing key and save it in the container.
            RSACryptoServiceProvider rsaKey = new RSACryptoServiceProvider(cspParams);

            // Create a new XML document.
            XmlDocument xmlDoc = new XmlDocument();

            // Load an XML file into the XmlDocument object.
            xmlDoc.PreserveWhitespace = true;
            xmlDoc.Load("test.xml");

            // Verify the signature of the signed XML.
            Console.WriteLine("Verifying signature...");
            bool result = VerifyXml(xmlDoc, rsaKey);

            // Display the results of the signature verification to
            // the console.
            if (result)
            {
                Console.WriteLine("The XML signature is valid.");
            }
            else
            {
                Console.WriteLine("The XML signature is not valid.");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }

    // Verify the signature of an XML file against an asymmetric
    // algorithm and return the result.
    public static Boolean VerifyXml(XmlDocument xmlDoc, RSA key)
    {
        // Check arguments.
        if (xmlDoc == null)
             throw new ArgumentException("xmlDoc");
        if (key == null)
            throw new ArgumentException("key");

        // Create a new SignedXml object and pass it
        // the XML document class.
        SignedXml signedXml = new SignedXml(xmlDoc);

        // Find the "Signature" node and create a new
        // XmlNodeList object.
        XmlNodeList nodeList = xmlDoc.GetElementsByTagName("Signature");

        // Throw an exception if no signature was found.
        if (nodeList.Count <= 0)
        {
            throw new CryptographicException("Verification failed: No Signature was found in the document.");
        }

        // This example only supports one signature for
        // the entire XML document.  Throw an exception
        // if more than one signature was found.
        if (nodeList.Count >= 2)
        {
            throw new CryptographicException("Verification failed: More that one signature was found for the
document.");
        }

        // Load the first <signature> node.
```

```
        // Load the first <signature> node.
        signedXml.LoadXml((XmlElement)nodeList[0]);

        // Check the signature and return the result.
        return signedXml.CheckSignature(key);
    }
}
```

```
Imports System.Security.Cryptography
Imports System.Security.Cryptography.Xml
Imports System.Xml

Module VerifyXML
    Sub Main(ByVal args() As String)
        Try
            ' Create a new CspParameters object to specify
            ' a key container.
            Dim cspParams As New CspParameters()
            cspParams.KeyContainerName = "XML_DSIG_RSA_KEY"
            ' Create a new RSA signing key and save it in the container.
            Dim rsaKey As New RSACryptoServiceProvider(cspParams)
            ' Create a new XML document.
            Dim xmlDoc As New XmlDocument()

            ' Load an XML file into the XmlDocument object.
            xmlDoc.PreserveWhitespace = True
            xmlDoc.Load("test.xml")
            ' Verify the signature of the signed XML.
            Console.WriteLine("Verifying signature...")
            Dim result As Boolean = VerifyXml(xmlDoc, rsaKey)

            ' Display the results of the signature verification to
            ' the console.
            If result Then
                Console.WriteLine("The XML signature is valid.")
            Else
                Console.WriteLine("The XML signature is not valid.")
            End If

        Catch e As Exception
            Console.WriteLine(e.Message)
        End Try
    End Sub

    ' Verify the signature of an XML file against an asymmetric
    ' algorithm and return the result.
    Function VerifyXml(ByVal xmlDoc As XmlDocument, ByVal key As RSA) As [Boolean]
        ' Check arguments.
        If xmlDoc Is Nothing Then
            Throw New ArgumentException("xmlDoc")
        End If
        If key Is Nothing Then
            Throw New ArgumentException("key")
        End If
        ' Create a new SignedXml object and pass it
        ' the XML document class.
        Dim signedXml As New SignedXml(xmlDoc)
        ' Find the "Signature" node and create a new
        ' XmlNodeList object.
        Dim nodeList As XmlNodeList = xmlDoc.GetElementsByTagName("Signature")
        ' Throw an exception if no signature was found.
        If nodeList.Count <= 0 Then
            Throw New CryptographicException("Verification failed: No Signature was found in the document.")
        End If

        ' This example only supports one signature for
        ' the entire XML document.  Throw an exception
        ' if more than one signature was found.
```

```
        If nodeList.Count >= 2 Then
            Throw New CryptographicException("Verification failed: More that one signature was found for the
document.")
        End If

        ' Load the first <signature> node.
        signedXml.LoadXml(CType(nodeList(0), XmlElement))
        ' Check the signature and return the result.
        Return signedXml.CheckSignature(key)
    End Function
End Module
```

Compilar el código

- Para compilar este ejemplo, debe incluir una referencia a `System.Security.dll`.
- Incluya los siguientes espacios de nombres: System.Xml, System.Security.Cryptography y System.Security.Cryptography.Xml.

Seguridad de .NET Framework

Nunca almacene ni transfiera la clave privada de un par de claves asimétricas en texto sin formato. Para obtener más información acerca de las claves criptográficas simétricas y asimétricas, vea generar claves para cifrado y descifrado.

No inserte nunca una clave privada directamente en el código fuente. Las claves incrustadas se pueden leer fácilmente desde un ensamblado mediante Ildasm. exe (desensamblador de IL) o abriendo el ensamblado en un editor de texto como el Bloc de notas.

Consulte también

- System.Security.Cryptography.Xml
- Procedimiento para firmar documentos XML con firmas digitales

Procedimiento para usar la protección de datos

02/07/2020 • 10 minutes to read • Edit Online

.NET Framework proporciona acceso a la API de protección de datos (DPAPI), que permite cifrar datos usando la información del equipo o la cuenta del usuario actual. Cuando se usa la DPAPI, se alivia el difícil problema de generar y almacenar explícitamente una clave criptográfica.

Use la clase ProtectedMemory para cifrar una matriz de bytes en memoria. Esta funcionalidad está disponible en Microsoft Windows XP y en los sistemas operativos posteriores. Puede especificar que la memoria cifrada por el proceso actual pueda ser descifrada solo por el proceso actual, por todos los procesos o desde el mismo contexto de usuario. Consulte la enumeración MemoryProtectionScope para obtener una descripción detallada de las opciones de ProtectedMemory.

Use la clase ProtectedData para cifrar una copia de una matriz de bytes. Esta funcionalidad está disponible en Microsoft Windows 2000 y en los sistemas operativos posteriores. Puede especificar que los datos cifrados por la cuenta de usuario actual solo puedan ser descifrados por la misma cuenta de usuario o que puedan ser descifrados por cualquier cuenta del equipo. Consulte la enumeración DataProtectionScope para obtener una descripción detallada de las opciones de ProtectedData.

Para cifrar los datos en memoria mediante la protección de datos

1. Llame al método Protect estático mientras pasa la matriz de bytes que se vaya a cifrar, la entropía y el ámbito de la protección de memoria.

Para descifrar los datos en memoria mediante la protección de datos

1. Llame al método Unprotect estático mientras pasa la matriz de bytes que se vaya a descifrar y el ámbito de la protección de memoria.

Para cifrar datos en un archivo o una secuencia usando la protección de datos

1. Cree una entropía aleatoria.
2. Llame al método Protect estático mientras pasa la matriz de bytes que se vaya a cifrar, la entropía y el ámbito de la protección de datos.
3. Escriba los datos cifrados en un archivo o en una secuencia.

Para descifrar los datos de un archivo o secuencia usando la protección de datos

1. Lea los datos cifrados desde un archivo o una secuencia.
2. Llame al método Unprotect estático mientras pasa la matriz de bytes que se vaya a descifrar y el ámbito de la protección de datos.

Ejemplo

En el siguiente ejemplo de código se muestran dos formas de cifrado y descifrado. En primer lugar, el ejemplo de código cifra y luego descifra la matriz de bytes en memoria. A continuación, el ejemplo de código cifra una copia de una matriz de bytes, la guarda en un archivo, vuelve a cargar los datos del archivo y, a continuación, descifra los datos. El ejemplo muestra los datos originales, los datos cifrados y los datos descifrados.

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

```
public class MemoryProtectionSample
{
    public static void Main()
    {
        Run();
    }

    public static void Run()
    {
        try
        {

            ////////////////////////////////
            //
            // Memory Encryption - ProtectedMemory
            //
            ////////////////////////////////

            // Create the original data to be encrypted (The data length should be a multiple of 16).
            byte[] toEncrypt = UnicodeEncoding.ASCII.GetBytes("ThisIsSomeData16");

            Console.WriteLine("Original data: " + UnicodeEncoding.ASCII.GetString(toEncrypt));
            Console.WriteLine("Encrypting...");

            // Encrypt the data in memory.
            EncryptInMemoryData(toEncrypt, MemoryProtectionScope.SameLogon);

            Console.WriteLine("Encrypted data: " + UnicodeEncoding.ASCII.GetString(toEncrypt));
            Console.WriteLine("Decrypting...");

            // Decrypt the data in memory.
            DecryptInMemoryData(toEncrypt, MemoryProtectionScope.SameLogon);

            Console.WriteLine("Decrypted data: " + UnicodeEncoding.ASCII.GetString(toEncrypt));

            ////////////////////////////////
            //
            // Data Encryption - ProtectedData
            //
            ////////////////////////////////

            // Create the original data to be encrypted
            toEncrypt = UnicodeEncoding.ASCII.GetBytes("This is some data of any length.");

            // Create a file.
            FileStream fStream = new FileStream("Data.dat", FileMode.OpenOrCreate);

            // Create some random entropy.
            byte[] entropy = CreateRandomEntropy();

            Console.WriteLine();
            Console.WriteLine("Original data: " + UnicodeEncoding.ASCII.GetString(toEncrypt));
            Console.WriteLine("Encrypting and writing to disk...");

            // Encrypt a copy of the data to the stream.
            int bytesWritten = EncryptDataToStream(toEncrypt, entropy, DataProtectionScope.CurrentUser,
fStream);

            fStream.Close();

            Console.WriteLine("Reading data from disk and decrypting...");

            // Open the file.
            fStream = new FileStream("Data.dat", FileMode.Open);

            // Read from the stream and decrypt the data.
            byte[] decryptData = DecryptDataFromStream(entropy, DataProtectionScope.CurrentUser, fStream,
bytesWritten);
```

```
            fStream.Close();

            Console.WriteLine("Decrypted data: " + UnicodeEncoding.ASCII.GetString(decryptData));
        }
        catch (Exception e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }
    }

    public static void EncryptInMemoryData(byte[] Buffer, MemoryProtectionScope Scope )
    {
        if (Buffer == null)
            throw new ArgumentNullException("Buffer");
        if (Buffer.Length <= 0)
            throw new ArgumentException("Buffer");

        // Encrypt the data in memory. The result is stored in the same array as the original data.
        ProtectedMemory.Protect(Buffer, Scope);
    }

    public static void DecryptInMemoryData(byte[] Buffer, MemoryProtectionScope Scope)
    {
        if (Buffer == null)
            throw new ArgumentNullException("Buffer");
        if (Buffer.Length <= 0)
            throw new ArgumentException("Buffer");

        // Decrypt the data in memory. The result is stored in the same array as the original data.
        ProtectedMemory.Unprotect(Buffer, Scope);
    }

    public static byte[] CreateRandomEntropy()
    {
        // Create a byte array to hold the random value.
        byte[] entropy = new byte[16];

        // Create a new instance of the RNGCryptoServiceProvider.
        // Fill the array with a random value.
        new RNGCryptoServiceProvider().GetBytes(entropy);

        // Return the array.
        return entropy;
    }

    public static int EncryptDataToStream(byte[] Buffer, byte[] Entropy, DataProtectionScope Scope, Stream S)
    {
        if (Buffer == null)
            throw new ArgumentNullException("Buffer");
        if (Buffer.Length <= 0)
            throw new ArgumentException("Buffer");
        if (Entropy == null)
            throw new ArgumentNullException("Entropy");
        if (Entropy.Length <= 0)
            throw new ArgumentException("Entropy");
        if (S == null)
            throw new ArgumentNullException("S");

        int length = 0;

        // Encrypt the data and store the result in a new byte array. The original data remains unchanged.
        byte[] encryptedData = ProtectedData.Protect(Buffer, Entropy, Scope);

        // Write the encrypted data to a stream.
        if (S.CanWrite && encryptedData != null)
        {
            S.Write(encryptedData, 0, encryptedData.Length);

            length = encryptedData.Length;
```

```
        }

        // Return the length that was written to the stream.
        return length;
    }

    public static byte[] DecryptDataFromStream(byte[] Entropy, DataProtectionScope Scope, Stream S, int Length)
    {
        if (S == null)
            throw new ArgumentNullException("S");
        if (Length <= 0 )
            throw new ArgumentException("Length");
        if (Entropy == null)
            throw new ArgumentNullException("Entropy");
        if (Entropy.Length <= 0)
            throw new ArgumentException("Entropy");

        byte[] inBuffer = new byte[Length];
        byte[] outBuffer;

        // Read the encrypted data from a stream.
        if (S.CanRead)
        {
            S.Read(inBuffer, 0, Length);

            outBuffer = ProtectedData.Unprotect(inBuffer, Entropy, Scope);
        }
        else
        {
            throw new IOException("Could not read the stream.");
        }

        // Return the length that was written to the stream.
        return outBuffer;
    }
}
```

```
Imports System.IO
Imports System.Text
Imports System.Security.Cryptography



Public Module MemoryProtectionSample

    Sub Main()
        Run()

    End Sub


    Sub Run()
        Try

            '''''''''''''''''''''''''''''''''''
            '
            ' Memory Encryption - ProtectedMemory
            '
            '''''''''''''''''''''''''''''''''''
            ' Create the original data to be encrypted (The data length should be a multiple of 16).
            Dim toEncrypt As Byte() = UnicodeEncoding.ASCII.GetBytes("ThisIsSomeData16")

            Console.WriteLine("Original data: " + UnicodeEncoding.ASCII.GetString(toEncrypt))
            Console.WriteLine("Encrypting...")

            ' Encrypt the data in memory.
            EncryptInMemoryData(toEncrypt, MemoryProtectionScope.SameLogon)
```

```
            Console.WriteLine("Encrypted data: " + UnicodeEncoding.ASCII.GetString(toEncrypt))
            Console.WriteLine("Decrypting...")

            ' Decrypt the data in memory.
            DecryptInMemoryData(toEncrypt, MemoryProtectionScope.SameLogon)

            Console.WriteLine("Decrypted data: " + UnicodeEncoding.ASCII.GetString(toEncrypt))


            ''''''''''''''''''''''''''''''''''
            '
            ' Data Encryption - ProtectedData
            '
            ''''''''''''''''''''''''''''''''''
            ' Create the original data to be encrypted
            toEncrypt = UnicodeEncoding.ASCII.GetBytes("This is some data of any length.")

            ' Create a file.
            Dim fStream As New FileStream("Data.dat", FileMode.OpenOrCreate)

            ' Create some random entropy.
            Dim entropy As Byte() = CreateRandomEntropy()

            Console.WriteLine()
            Console.WriteLine("Original data: " + UnicodeEncoding.ASCII.GetString(toEncrypt))
            Console.WriteLine("Encrypting and writing to disk...")

            ' Encrypt a copy of the data to the stream.
            Dim bytesWritten As Integer = EncryptDataToStream(toEncrypt, entropy,
DataProtectionScope.CurrentUser, fStream)

            fStream.Close()

            Console.WriteLine("Reading data from disk and decrypting...")

            ' Open the file.
            fStream = New FileStream("Data.dat", FileMode.Open)

            ' Read from the stream and decrypt the data.
            Dim decryptData As Byte() = DecryptDataFromStream(entropy, DataProtectionScope.CurrentUser,
fStream, bytesWritten)

            fStream.Close()

            Console.WriteLine("Decrypted data: " + UnicodeEncoding.ASCII.GetString(decryptData))


        Catch e As Exception
            Console.WriteLine("ERROR: " + e.Message)
        End Try

    End Sub



    Sub EncryptInMemoryData(ByVal Buffer() As Byte, ByVal Scope As MemoryProtectionScope)
        If Buffer Is Nothing Then
            Throw New ArgumentNullException("Buffer")
        End If
        If Buffer.Length <= 0 Then
            Throw New ArgumentException("Buffer")
        End If

        ' Encrypt the data in memory. The result is stored in the same array as the original data.
        ProtectedMemory.Protect(Buffer, Scope)

    End Sub
```

```
    Sub DecryptInMemoryData(ByVal Buffer() As Byte, ByVal Scope As MemoryProtectionScope)
        If Buffer Is Nothing Then
            Throw New ArgumentNullException("Buffer")
        End If
        If Buffer.Length <= 0 Then
            Throw New ArgumentException("Buffer")
        End If

        ' Decrypt the data in memory. The result is stored in the same array as the original data.
        ProtectedMemory.Unprotect(Buffer, Scope)

    End Sub


    Function CreateRandomEntropy() As Byte()
        ' Create a byte array to hold the random value.
        Dim entropy(15) As Byte

        ' Create a new instance of the RNGCryptoServiceProvider.
        ' Fill the array with a random value.
        Dim RNG As New RNGCryptoServiceProvider()

        RNG.GetBytes(entropy)

        ' Return the array.
        Return entropy

    End Function 'CreateRandomEntropy



    Function EncryptDataToStream(ByVal Buffer() As Byte, ByVal Entropy() As Byte, ByVal Scope As
DataProtectionScope, ByVal S As Stream) As Integer
        If Buffer Is Nothing Then
            Throw New ArgumentNullException("Buffer")
        End If
        If Buffer.Length <= 0 Then
            Throw New ArgumentException("Buffer")
        End If
        If Entropy Is Nothing Then
            Throw New ArgumentNullException("Entropy")
        End If
        If Entropy.Length <= 0 Then
            Throw New ArgumentException("Entropy")
        End If
        If S Is Nothing Then
            Throw New ArgumentNullException("S")
        End If
        Dim length As Integer = 0

        ' Encrypt the data and store the result in a new byte array. The original data remains unchanged.
        Dim encryptedData As Byte() = ProtectedData.Protect(Buffer, Entropy, Scope)

        ' Write the encrypted data to a stream.
        If S.CanWrite AndAlso Not (encryptedData Is Nothing) Then
            S.Write(encryptedData, 0, encryptedData.Length)

            length = encryptedData.Length
        End If

        ' Return the length that was written to the stream.
        Return length

    End Function 'EncryptDataToStream


    Function DecryptDataFromStream(ByVal Entropy() As Byte, ByVal Scope As DataProtectionScope, ByVal S As
Stream, ByVal Length As Integer) As Byte()
```

```
Stream, ByVal Length As Integer) As Byte()
        If S Is Nothing Then
            Throw New ArgumentNullException("S")
        End If
        If Length <= 0 Then
            Throw New ArgumentException("Length")
        End If
        If Entropy Is Nothing Then
            Throw New ArgumentNullException("Entropy")
        End If
        If Entropy.Length <= 0 Then
            Throw New ArgumentException("Entropy")
        End If


        Dim inBuffer(Length - 1) As Byte
        Dim outBuffer() As Byte

        ' Read the encrypted data from a stream.
        If S.CanRead Then
            S.Read(inBuffer, 0, Length)

            outBuffer = ProtectedData.Unprotect(inBuffer, Entropy, Scope)
        Else
            Throw New IOException("Could not read the stream.")
        End If

        ' Return the unencrypted data as byte array.
        Return outBuffer

    End Function 'DecryptDataFromStream
End Module 'MemoryProtectionSample
```

Compilar el código

- Incluya una referencia a `System.Security.dll`.
- Incluya el espacio de nombres System, System.IO, System.Security.Cryptography y System.Text.

Vea también

- ProtectedMemory
- ProtectedData

Cómo: Obtener acceso a los dispositivos de cifrado de hardware

10/01/2020 • 5 minutes to read • Edit Online

Puede usar la clase CspParameters para obtener acceso a los dispositivos de cifrado de hardware. Por ejemplo, puede usar esta clase para integrar la aplicación con una tarjeta inteligente, un generador de números aleatorios de hardware o una implementación de hardware de un determinado algoritmo criptográfico.

La clase CspParameters crea un proveedor de servicios criptográficos (CSP) que tiene acceso a un dispositivo de cifrado de hardware correctamente instalado. Puede comprobar la disponibilidad de un CSP inspeccionando la siguiente clave del registro con el Editor del registro (Regedit.exe): HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.

Para firmar datos mediante una tarjeta de claves

1. Cree una instancia nueva de la clase CspParameters; para ello, pase el tipo de proveedor entero y el nombre del proveedor al constructor.
2. Pase las marcas adecuadas a la propiedad Flags del objeto CspParameters recién creado. Por ejemplo, pase la marca UseDefaultKeyContainer.
3. Cree una nueva instancia de una clase AsymmetricAlgorithm (por ejemplo, la clase RSACryptoServiceProvider) pasando el objeto CspParameters al constructor.
4. Firme los datos mediante uno de los métodos `Sign` y compruebe los datos mediante uno de los métodos `Verify`.

Para generar un número aleatorio mediante un generador de números aleatorios de hardware

1. Cree una instancia nueva de la clase CspParameters; para ello, pase el tipo de proveedor entero y el nombre del proveedor al constructor.
2. Cree una nueva instancia del RNGCryptoServiceProvider pasando el objeto CspParameters al constructor.
3. Cree un valor aleatorio con el método GetBytes o GetNonZeroBytes.

Ejemplo

En el siguiente ejemplo de código se muestra cómo firmar datos con una tarjeta inteligente. El ejemplo de código crea un objeto CspParameters que expone una tarjeta inteligente e inicializa un objeto RSACryptoServiceProvider con el CSP. Después, el ejemplo de código firma y comprueba algunos datos.

```
using namespace System;
using namespace System::Security::Cryptography;
int main()
{

   // To idendify the Smart Card CryptoGraphic Providers on your
   // computer, use the Microsoft Registry Editor (Regedit.exe).
   // The available Smart Card CryptoGraphic Providers are listed
   // in HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.
   // Create a new CspParameters object that identifies a
   // Smart Card CryptoGraphic Provider.
   // The 1st parameter comes from HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider Types.
   // The 2nd parameter comes from HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.
   CspParameters^ csp = gcnew CspParameters( 1,L"Schlumberger Cryptographic Service Provider" );
   csp->Flags = CspProviderFlags::UseDefaultKeyContainer;

   // Initialize an RSACryptoServiceProvider object using
   // the CspParameters object.
   RSACryptoServiceProvider^ rsa = gcnew RSACryptoServiceProvider( csp );

   // Create some data to sign.
   array<Byte>^data = gcnew array<Byte>{
      0,1,2,3,4,5,6,7
   };
   Console::WriteLine( L"Data   : {0}", BitConverter::ToString( data ) );

   // Sign the data using the Smart Card CryptoGraphic Provider.
   array<Byte>^sig = rsa->SignData( data, L"SHA1" );
   Console::WriteLine( L"Signature : {0}", BitConverter::ToString( sig ) );

   // Verify the data using the Smart Card CryptoGraphic Provider.
   bool verified = rsa->VerifyData( data, L"SHA1", sig );
   Console::WriteLine( L"Verified  : {0}", verified );
}
```

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

namespace SmartCardSign
{
    class SCSign
    {
        static void Main(string[] args)
        {
            // To idendify the Smart Card CryptoGraphic Providers on your
            // computer, use the Microsoft Registry Editor (Regedit.exe).
            // The available Smart Card CryptoGraphic Providers are listed
            // in HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.

            // Create a new CspParameters object that identifies a
            // Smart Card CryptoGraphic Provider.
            // The 1st parameter comes from
HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider Types.
            // The 2nd parameter comes from
HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.
            CspParameters csp = new CspParameters(1, "Schlumberger Cryptographic Service Provider");
            csp.Flags = CspProviderFlags.UseDefaultKeyContainer;

            // Initialize an RSACryptoServiceProvider object using
            // the CspParameters object.
            RSACryptoServiceProvider rsa = new RSACryptoServiceProvider(csp);

            // Create some data to sign.
            byte[] data = new byte[] { 0, 1, 2, 3, 4, 5, 6, 7 };

            Console.WriteLine("Data   : " + BitConverter.ToString(data));

            // Sign the data using the Smart Card CryptoGraphic Provider.
            byte[] sig = rsa.SignData(data, "SHA1");

            Console.WriteLine("Signature : " + BitConverter.ToString(sig));

            // Verify the data using the Smart Card CryptoGraphic Provider.
            bool verified = rsa.VerifyData(data, "SHA1", sig);

            Console.WriteLine("Verified  : " + verified);
        }
    }
}
```

```
Imports System.Security.Cryptography



Module SCSign

    Sub Main(ByVal args() As String)
        ' To idendify the Smart Card CryptoGraphic Providers on your
        ' computer, use the Microsoft Registry Editor (Regedit.exe).
        ' The available Smart Card CryptoGraphic Providers are listed
        ' in HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.

        ' Create a new CspParameters object that identifies a
        ' Smart Card CryptoGraphic Provider.
        ' The 1st parameter comes from HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider
Types.
        ' The 2nd parameter comes from HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Defaults\Provider.
        Dim csp As New CspParameters(1, "Schlumberger Cryptographic Service Provider")
        csp.Flags = CspProviderFlags.UseDefaultKeyContainer

        ' Initialize an RSACryptoServiceProvider object using
        ' the CspParameters object.
        Dim rsa As New RSACryptoServiceProvider(csp)

        ' Create some data to sign.
        Dim data() As Byte = {0, 1, 2, 3, 4, 5, 6, 7}


        Console.WriteLine("Data   : " + BitConverter.ToString(data))

        ' Sign the data using the Smart Card CryptoGraphic Provider.
        Dim sig As Byte() = rsa.SignData(data, "SHA1")

        Console.WriteLine("Signature : " + BitConverter.ToString(sig))

        ' Verify the data using the Smart Card CryptoGraphic Provider.
        Dim verified As Boolean = rsa.VerifyData(data, "SHA1", sig)

        Console.WriteLine("Verified")

    End Sub

End Module
```

Compilar el código

- Incluya los espacios de nombres System y System.Security.Cryptography.
- Debe tener un lector de tarjetas inteligentes y los controladores correspondientes instalados en el equipo.
- Debe inicializar el objeto CspParameters con la información específica de su lector de tarjetas. Para obtener más información, consulte la documentación de su lector de tarjetas.

Tutorial: Crear una aplicación criptográfica

02/07/2020 • 25 minutes to read • Edit Online

En este tutorial se muestra cómo cifrar y descifrar contenido. Los ejemplos de código están diseñados para una aplicación de Windows Forms. Esta aplicación no muestra escenarios del mundo real, como el uso de tarjetas inteligentes. En su lugar, muestra los aspectos básicos del cifrado y el descifrado.

En este tutorial se usan las siguientes directrices para el cifrado:

- Use la clase RijndaelManaged, un algoritmo simétrico, para cifrar y descifrar datos mediante su Key y IV generados automáticamente.
- Use el RSACryptoServiceProvider, un algoritmo asimétrico, para cifrar y descifrar la clave en los datos cifrados por RijndaelManaged. Los algoritmos asimétricos son útiles para pequeñas cantidades de datos, como las claves.

> **NOTE**
> Si desea proteger los datos en el equipo en lugar de intercambiar contenido cifrado con otras personas, considere la posibilidad de usar la clase ProtectedData o ProtectedMemory.

En la tabla siguiente se resumen las tareas criptográficas de este tema.

TAREA	DESCRIPCIÓN
Crear una aplicación de Windows Forms	Enumera los controles necesarios para ejecutar la aplicación.
Declarar objetos globales	Declara variables de ruta de acceso de cadena, los CspParameters y el RSACryptoServiceProvider para disponer del contexto global de la clase Form.
Crear una clave asimétrica	Crea un par de valores de claves pública y privada asimétricas y le asigna un nombre de contenedor de claves.
Cifrar un archivo	Muestra un cuadro de diálogo para seleccionar un archivo para el cifrado y lo cifra.
Descifrar un archivo	Muestra un cuadro de diálogo para seleccionar un archivo cifrado para el descifrado y lo descifra.
Obtener una clave privada	Obtiene el par de claves completo con el nombre del contenedor de claves.
Exportar una clave pública	Guarda la clave en un archivo XML únicamente con parámetros públicos.
Importar una clave pública	Carga la clave desde un archivo XML en el contenedor de claves.
Probar la aplicación	Enumera los procedimientos para probar esta aplicación.

Requisitos previos

Necesitará los componentes siguientes para completar este tutorial:

- Referencias a los espacios de nombres System.IO y System.Security.Cryptography.

Crear una aplicación de Windows Forms

La mayoría de los ejemplos de código de este tutorial están diseñados para actuar como controladores de eventos de los controles de botón. En la tabla siguiente se enumeran los controles necesarios para que la aplicación de ejemplo y los nombres necesarios coincidan con los ejemplos de código.

CONTROL	NOMBRE	PROPIEDAD DE TEXTO (SEGÚN SEA NECESARIO)
Button	`buttonEncryptFile`	Cifrar archivo
Button	`buttonDecryptFile`	Descifrar archivo
Button	`buttonCreateAsmKeys`	Crear claves
Button	`buttonExportPublicKey`	Exportar la clave pública
Button	`buttonImportPublicKey`	Importar la clave pública
Button	`buttonGetPrivateKey`	Obtener la clave privada
Label	`label1`	Clave no establecida
OpenFileDialog	`openFileDialog1`	
OpenFileDialog	`openFileDialog2`	

Haga doble clic en los botones del Diseñador de Visual Studio para crear los controladores de eventos.

Declarar objetos globales

Agregue el siguiente código al constructor del formulario. Edite las variables de cadena para el entorno y las preferencias.

```
// Declare CspParmeters and RsaCryptoServiceProvider
// objects with global scope of your Form class.
CspParameters cspp = new CspParameters();
RSACryptoServiceProvider rsa;

// Path variables for source, encryption, and
// decryption folders. Must end with a backslash.
const string EncrFolder = @"c:\Encrypt\";
const string DecrFolder = @"c:\Decrypt\";
const string SrcFolder = @"c:\docs\";

// Public key file
const string PubKeyFile = @"c:\encrypt\rsaPublicKey.txt";

// Key container name for
// private/public key value pair.
const string keyName = "Key01";
```

```
' Declare CspParmeters and RsaCryptoServiceProvider
' objects with global scope of your Form class.
Dim cspp As CspParameters = New System.Security.Cryptography.CspParameters
Dim rsa As RSACryptoServiceProvider

' Path variables for source, encryption, and
' decryption folders. Must end with a backslash.
Dim EncrFolder As String = "c:\Encrypt\"
Dim DecrFolder As String = "c:\Decrypt\"
Dim SrcFolder As String = "c:\docs\"

' Public key file
Dim PubKeyFile As String = "c:\encrypt\rsaPublicKey.txt"

' Key container name for
' private/public key value pair.
Dim keyName As String = "Key01"
```

Crear una clave asimétrica

Esta tarea crea una clave asimétrica que cifra y descifra la clave RijndaelManaged. Esta clave se usó para cifrar el contenido y muestra el nombre del contenedor de claves en el control de etiqueta.

Agregue el siguiente código como controlador de eventos `Click` del botón `Create Keys` (`buttonCreateAsmKeys_Click`).

```
private void buttonCreateAsmKeys_Click(object sender, System.EventArgs e)
{
    // Stores a key pair in the key container.
    cspp.KeyContainerName = keyName;
    rsa = new RSACryptoServiceProvider(cspp);
    rsa.PersistKeyInCsp = true;
    if (rsa.PublicOnly == true)
        label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only";
    else
        label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair";
}
```

```
Private Sub buttonCreateAsmKeys_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
buttonCreateAsmKeys.Click
    ' Stores a key pair in the key container.
    cspp.KeyContainerName = keyName
    rsa = New RSACryptoServiceProvider(cspp)
    rsa.PersistKeyInCsp = True

    If rsa.PublicOnly = True Then
        Label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only"
    Else
        Label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair"
    End If

End Sub
```

Cifrar un archivo

Esta tarea implica dos métodos: el método de control de eventos para el `Encrypt File` botón (`buttonEncryptFile_Click`) y el `EncryptFile` método. El primer método muestra un cuadro de diálogo para seleccionar un archivo y pasa el nombre del archivo al segundo método, que lleva a cabo el cifrado.

El contenido cifrado, la clave y el vector de inicialización (IV) se guardan en un FileStream, conocido como paquete de cifrado.

El método `EncryptFile` hace lo siguiente:

1. Crea un algoritmo simétrico RijndaelManaged para cifrar el contenido.
2. Crea un objeto RSACryptoServiceProvider para cifrar la clave RijndaelManaged.
3. Usa un objeto CryptoStream para leer y cifrar el FileStream del archivo de código fuente, en bloques de bytes, en un objeto FileStream de destino para el archivo cifrado.
4. Determina la longitud de la clave cifrada y del IV y crea matrices de bytes de sus valores de longitud.
5. Escribe la clave, el IV y sus valores de longitud en el paquete cifrado.

El paquete de cifrado usa el siguiente formato:

- Longitud de clave, bytes 0 - 3
- Longitud del IV, bytes 4 - 7
- Clave cifrada
- IV
- Texto cifrado

Puede usar la longitud de la clave y del IV para determinar los puntos iniciales y la longitud de todas las partes del paquete de cifrado, que posteriormente se puede usar para descifrar el archivo.

Agregue el siguiente código como controlador de eventos `Click` del botón `Encrypt File` (`buttonEncryptFile_Click`).

```
private void buttonEncryptFile_Click(object sender, System.EventArgs e)
{
    if (rsa == null)
    {
        MessageBox.Show("Key not set.");
    }
    else
    {

        // Display a dialog box to select a file to encrypt.
        openFileDialog1.InitialDirectory = SrcFolder;
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            string fName = openFileDialog1.FileName;
            if (fName != null)
            {
                FileInfo fInfo = new FileInfo(fName);
                // Pass the file name without the path.
                string name = fInfo.FullName;
                EncryptFile(name);
            }
        }
    }
}
```

```
Private Sub buttonEncryptFile_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
buttonEncryptFile.Click
    If rsa Is Nothing Then
        MsgBox("Key not set.")
    Else
        ' Display a dialog box to select a file to encrypt.
        OpenFileDialog1.InitialDirectory = SrcFolder
        If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
            Try
                Dim fName As String = OpenFileDialog1.FileName
                If (Not (fName) Is Nothing) Then
                    Dim fInfo As FileInfo = New FileInfo(fName)
                    ' Use just the file name without path.
                    Dim name As String = fInfo.FullName
                    EncryptFile(name)
                End If
            Catch ex As Exception
                MsgBox(ex.Message)
            End Try
        End If
    End If
End Sub
```

Agregue el siguiente método `EncryptFile` al formulario.

```
private void EncryptFile(string inFile)
{

    // Create instance of Rijndael for
    // symetric encryption of the data.
    RijndaelManaged rjndl = new RijndaelManaged();
    rjndl.KeySize = 256;
    rjndl.BlockSize = 256;
    rjndl.Mode = CipherMode.CBC;
    ICryptoTransform transform = rjndl.CreateEncryptor();

    // Use RSACryptoServiceProvider to
    // enrypt the Rijndael key.
```

```
    // rsa is previously instantiated:
    //    rsa = new RSACryptoServiceProvider(cspp);
    byte[] keyEncrypted = rsa.Encrypt(rjndl.Key, false);

    // Create byte arrays to contain
    // the length values of the key and IV.
    byte[] LenK = new byte[4];
    byte[] LenIV = new byte[4];

    int lKey = keyEncrypted.Length;
    LenK = BitConverter.GetBytes(lKey);
    int lIV = rjndl.IV.Length;
    LenIV = BitConverter.GetBytes(lIV);

    // Write the following to the FileStream
    // for the encrypted file (outFs):
    // - length of the key
    // - length of the IV
    // - ecrypted key
    // - the IV
    // - the encrypted cipher content

    int startFileName = inFile.LastIndexOf("\\") + 1;
    // Change the file's extension to ".enc"
    string outFile = EncrFolder + inFile.Substring(startFileName, inFile.LastIndexOf(".") - startFileName) +
".enc";

    using (FileStream outFs = new FileStream(outFile, FileMode.Create))
    {

        outFs.Write(LenK, 0, 4);
        outFs.Write(LenIV, 0, 4);
        outFs.Write(keyEncrypted, 0, lKey);
        outFs.Write(rjndl.IV, 0, lIV);

        // Now write the cipher text using
        // a CryptoStream for encrypting.
        using (CryptoStream outStreamEncrypted = new CryptoStream(outFs, transform, CryptoStreamMode.Write))
        {

            // By encrypting a chunk at
            // a time, you can save memory
            // and accommodate large files.
            int count = 0;
            int offset = 0;

            // blockSizeBytes can be any arbitrary size.
            int blockSizeBytes = rjndl.BlockSize / 8;
            byte[] data = new byte[blockSizeBytes];
            int bytesRead = 0;

            using (FileStream inFs = new FileStream(inFile, FileMode.Open))
            {
                do
                {
                    count = inFs.Read(data, 0, blockSizeBytes);
                    offset += count;
                    outStreamEncrypted.Write(data, 0, count);
                    bytesRead += blockSizeBytes;
                }
                while (count > 0);
                inFs.Close();
            }
            outStreamEncrypted.FlushFinalBlock();
            outStreamEncrypted.Close();
        }
        outFs.Close();
    }
}
```

```
Private Sub EncryptFile(ByVal inFile As String)

    ' Create instance of Rijndael for
    ' symetric encryption of the data.
    Dim rjndl As RijndaelManaged = New RijndaelManaged
    rjndl.KeySize = 256
    rjndl.BlockSize = 256
    rjndl.Mode = CipherMode.CBC
    Dim transform As ICryptoTransform = rjndl.CreateEncryptor

    ' Use RSACryptoServiceProvider to
    ' enrypt the Rijndael key.
    Dim keyEncrypted() As Byte = rsa.Encrypt(rjndl.Key, False)

    ' Create byte arrays to contain
    ' the length values of the key and IV.
    Dim LenK() As Byte = New Byte((4) - 1) {}
    Dim LenIV() As Byte = New Byte((4) - 1) {}
    Dim lKey As Integer = keyEncrypted.Length
    LenK = BitConverter.GetBytes(lKey)
    Dim lIV As Integer = rjndl.IV.Length
    LenIV = BitConverter.GetBytes(lIV)

    ' Write the following to the FileStream
    ' for the encrypted file (outFs):
    ' - length of the key
    ' - length of the IV
    ' - ecrypted key
    ' - the IV
    ' - the encrypted cipher content
    ' Change the file's extension to ".enc"
    Dim startFileName As Integer = inFile.LastIndexOf("\") + 1
    Dim outFile As String = (EncrFolder _
                + (inFile.Substring(startFileName, inFile.LastIndexOf(".") - startFileName) + ".enc"))

    Using outFs As FileStream = New FileStream(outFile, FileMode.Create)

        outFs.Write(LenK, 0, 4)
        outFs.Write(LenIV, 0, 4)
        outFs.Write(keyEncrypted, 0, lKey)
        outFs.Write(rjndl.IV, 0, lIV)

        ' Now write the cipher text using
        ' a CryptoStream for encrypting.
        Using outStreamEncrypted As CryptoStream = New CryptoStream(outFs, transform, CryptoStreamMode.Write)
            ' By encrypting a chunk at
            ' a time, you can save memory
            ' and accommodate large files.
            Dim count As Integer = 0
            Dim offset As Integer = 0

            ' blockSizeBytes can be any arbitrary size.
            Dim blockSizeBytes As Integer = (rjndl.BlockSize / 8)
            Dim data() As Byte = New Byte((blockSizeBytes) - 1) {}
            Dim bytesRead As Integer = 0
            Using inFs As FileStream = New FileStream(inFile, FileMode.Open)
                Do
                    count = inFs.Read(data, 0, blockSizeBytes)
                    offset = (offset + count)
                    outStreamEncrypted.Write(data, 0, count)
                    bytesRead = (bytesRead + blockSizeBytes)
                Loop Until (count = 0)

                outStreamEncrypted.FlushFinalBlock()
                inFs.Close()
            End Using
```

```
            outStreamEncrypted.Close()
        End Using
        outFs.Close()
    End Using
End Sub
```

Descifrar un archivo

Esta tarea implica dos métodos: el método del controlador de eventos del botón `Decrypt File` (`buttonDecryptFile_Click`) y el método `DecryptFile`. El primer método muestra un cuadro de diálogo para seleccionar un archivo y pasa su nombre al segundo método, que lleva a cabo el descifrado.

El método `Decrypt` hace lo siguiente:

1. Crea un algoritmo simétrico RijndaelManaged para descifrar el contenido.
2. Lee los primeros ocho bytes del FileStream del paquete de cifrado en matrices de bytes para obtener la longitud de la clave cifrada y del IV.
3. Extrae la clave y el IV del paquete de cifrado en matrices de bytes.
4. Crea un objeto RSACryptoServiceProvider para descifrar la clave RijndaelManaged.
5. Usa un objeto CryptoStream para leer y descifrar la sección de texto cifrado del paquete de cifrado FileStream, en bloques de bytes, en el objeto FileStream del archivo descifrado. Al finalizar, se habrá completado el descifrado.

Agregue el siguiente código como controlador de eventos `Click` del botón `Decrypt File`.

```
private void buttonDecryptFile_Click(object sender, EventArgs e)
{
    if (rsa == null)
    {
        MessageBox.Show("Key not set.");
    }
    else
    {
        // Display a dialog box to select the encrypted file.
        openFileDialog2.InitialDirectory = EncrFolder;
        if (openFileDialog2.ShowDialog() == DialogResult.OK)
        {
            string fName = openFileDialog2.FileName;
            if (fName != null)
            {
                FileInfo fi = new FileInfo(fName);
                string name = fi.Name;
                DecryptFile(name);
            }
        }
    }
}
```

```
Private Sub buttonDecryptFile_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
buttonDecryptFile.Click
    If rsa Is Nothing Then
        MsgBox("Key not set.")
    Else
        ' Display a dialog box to select the encrypted file.
        OpenFileDialog2.InitialDirectory = EncrFolder
        If (OpenFileDialog2.ShowDialog = Windows.Forms.DialogResult.OK) Then
            Try
                Dim fName As String = OpenFileDialog2.FileName
                If (Not (fName) Is Nothing) Then
                    Dim fi As FileInfo = New FileInfo(fName)
                    Dim name As String = fi.Name
                    DecryptFile(name)
                End If
            Catch ex As Exception
                MessageBox.Show(ex.Message)
            End Try
        End If
    End If
End Sub
```

Agregue el siguiente método `DecryptFile` al formulario.

```
private void DecryptFile(string inFile)
{

    // Create instance of Rijndael for
    // symetric decryption of the data.
    RijndaelManaged rjndl = new RijndaelManaged();
    rjndl.KeySize = 256;
    rjndl.BlockSize = 256;
    rjndl.Mode = CipherMode.CBC;

    // Create byte arrays to get the length of
    // the encrypted key and IV.
    // These values were stored as 4 bytes each
    // at the beginning of the encrypted package.
    byte[] LenK = new byte[4];
    byte[] LenIV = new byte[4];

    // Consruct the file name for the decrypted file.
    string outFile = DecrFolder + inFile.Substring(0, inFile.LastIndexOf(".")) + ".txt";

    // Use FileStream objects to read the encrypted
    // file (inFs) and save the decrypted file (outFs).
    using (FileStream inFs = new FileStream(EncrFolder + inFile, FileMode.Open))
    {

        inFs.Seek(0, SeekOrigin.Begin);
        inFs.Seek(0, SeekOrigin.Begin);
        inFs.Read(LenK, 0, 3);
        inFs.Seek(4, SeekOrigin.Begin);
        inFs.Read(LenIV, 0, 3);

        // Convert the lengths to integer values.
        int lenK = BitConverter.ToInt32(LenK, 0);
        int lenIV = BitConverter.ToInt32(LenIV, 0);

        // Determine the start postition of
        // the ciphter text (startC)
        // and its length(lenC).
        int startC = lenK + lenIV + 8;
        int lenC = (int)inFs.Length - startC;

        // Create the byte arrays for
```

```
        // the encrypted Rijndael key,
        // the IV, and the cipher text.
        byte[] KeyEncrypted = new byte[lenK];
        byte[] IV = new byte[lenIV];

        // Extract the key and IV
        // starting from index 8
        // after the length values.
        inFs.Seek(8, SeekOrigin.Begin);
        inFs.Read(KeyEncrypted, 0, lenK);
        inFs.Seek(8 + lenK, SeekOrigin.Begin);
        inFs.Read(IV, 0, lenIV);
        Directory.CreateDirectory(DecrFolder);
        // Use RSACryptoServiceProvider
        // to decrypt the Rijndael key.
        byte[] KeyDecrypted = rsa.Decrypt(KeyEncrypted, false);

        // Decrypt the key.
        ICryptoTransform transform = rjndl.CreateDecryptor(KeyDecrypted, IV);

        // Decrypt the cipher text from
        // from the FileSteam of the encrypted
        // file (inFs) into the FileStream
        // for the decrypted file (outFs).
        using (FileStream outFs = new FileStream(outFile, FileMode.Create))
        {

            int count = 0;
            int offset = 0;

            // blockSizeBytes can be any arbitrary size.
            int blockSizeBytes = rjndl.BlockSize / 8;
            byte[] data = new byte[blockSizeBytes];

            // By decrypting a chunk a time,
            // you can save memory and
            // accommodate large files.

            // Start at the beginning
            // of the cipher text.
            inFs.Seek(startC, SeekOrigin.Begin);
            using (CryptoStream outStreamDecrypted = new CryptoStream(outFs, transform,
CryptoStreamMode.Write))
            {
                do
                {
                    count = inFs.Read(data, 0, blockSizeBytes);
                    offset += count;
                    outStreamDecrypted.Write(data, 0, count);
                }
                while (count > 0);

                outStreamDecrypted.FlushFinalBlock();
                outStreamDecrypted.Close();
            }
            outFs.Close();
        }
        inFs.Close();
    }
}
```

```
Private Sub DecryptFile(ByVal inFile As String)
    ' Create instance of Rijndael for
    ' symetric decryption of the data.
    Dim rjndl As RijndaelManaged = New RijndaelManaged
    rjndl.KeySize = 256
    rjndl.BlockSize = 256
```

```
    rjndl.Mode = CipherMode.CBC

    ' Create byte arrays to get the length of
    ' the encrypted key and IV.
    ' These values were stored as 4 bytes each
    ' at the beginning of the encrypted package.
    Dim LenK() As Byte = New Byte(4 - 1) {}
    Dim LenIV() As Byte = New Byte(4 - 1) {}

    ' Construct the file name for the decrypted file.
    Dim outFile As String = (DecrFolder _
                + (inFile.Substring(0, inFile.LastIndexOf(".")) + ".txt"))

    ' Use FileStream objects to read the encrypted
    ' file (inFs) and save the decrypted file (outFs).
    Using inFs As FileStream = New FileStream((EncrFolder + inFile), FileMode.Open)

        inFs.Seek(0, SeekOrigin.Begin)
        inFs.Read(LenK, 0, 3)
        inFs.Seek(4, SeekOrigin.Begin)
        inFs.Read(LenIV, 0, 3)

        Dim lengthK As Integer = BitConverter.ToInt32(LenK, 0)
        Dim lengthIV As Integer = BitConverter.ToInt32(LenIV, 0)
        Dim startC As Integer = (lengthK + lengthIV + 8)
        Dim lenC As Integer = (CType(inFs.Length, Integer) - startC)
        Dim KeyEncrypted() As Byte = New Byte(lengthK - 1) {}
        Dim IV() As Byte = New Byte(lengthIV - 1) {}

        ' Extract the key and IV
        ' starting from index 8
        ' after the length values.
        inFs.Seek(8, SeekOrigin.Begin)
        inFs.Read(KeyEncrypted, 0, lengthK)
        inFs.Seek(8 + lengthK, SeekOrigin.Begin)
        inFs.Read(IV, 0, lengthIV)
        Directory.CreateDirectory(DecrFolder)
        ' User RSACryptoServiceProvider
        ' to decryt the Rijndael key
        Dim KeyDecrypted() As Byte = rsa.Decrypt(KeyEncrypted, False)

        ' Decrypt the key.
        Dim transform As ICryptoTransform = rjndl.CreateDecryptor(KeyDecrypted, IV)
        ' Decrypt the cipher text from
        ' from the FileSteam of the encrypted
        ' file (inFs) into the FileStream
        ' for the decrypted file (outFs).
        Using outFs As FileStream = New FileStream(outFile, FileMode.Create)
            Dim count As Integer = 0
            Dim offset As Integer = 0

            ' blockSizeBytes can be any arbitrary size.
            Dim blockSizeBytes As Integer = (rjndl.BlockSize / 8)
            Dim data() As Byte = New Byte(blockSizeBytes - 1) {}
            ' By decrypting a chunk a time,
            ' you can save memory and
            ' accommodate large files.
            ' Start at the beginning
            ' of the cipher text.
            inFs.Seek(startC, SeekOrigin.Begin)
            Using outStreamDecrypted As CryptoStream = New CryptoStream(outFs, transform,
CryptoStreamMode.Write)
                Do
                    count = inFs.Read(data, 0, blockSizeBytes)
                    offset = (offset + count)
                    outStreamDecrypted.Write(data, 0, count)
                Loop Until (count = 0)

                outStreamDecrypted.FlushFinalBlock()
```

```
                outStreamDecrypted.Close()
            End Using
            outFs.Close()
        End Using
        inFs.Close()
    End Using
End Sub
```

Exportar una clave pública

Esta tarea guarda en un archivo la clave creada por el botón `Create Keys`. Solo exporta los parámetros públicos.

Esta tarea simula un escenario en que Alicia da a Roberto su clave pública para que pueda cifrar archivos por ella. Roberto y otras personas que tengan esa clave pública no podrán descifrarlos porque no tienen el par de claves completo con los parámetros privados.

Agregue el siguiente código como controlador de eventos `Click` del botón `Export Public Key` (`buttonExportPublicKey_Click`).

```
void buttonExportPublicKey_Click(object sender, System.EventArgs e)
{
    // Save the public key created by the RSA
    // to a file. Caution, persisting the
    // key to a file is a security risk.
    Directory.CreateDirectory(EncrFolder);
    StreamWriter sw = new StreamWriter(PubKeyFile, false);
    sw.Write(rsa.ToXmlString(false));
    sw.Close();
}
```

```
Private Sub buttonExportPublicKey_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
buttonExportPublicKey.Click
    ' Save the public key created by the RSA
    ' to a file. Caution, persisting the
    ' key to a file is a security risk.
    Directory.CreateDirectory(EncrFolder)
    Dim sw As StreamWriter = New StreamWriter(PubKeyFile)
    sw.Write(rsa.ToXmlString(False))
    sw.Close()
End Sub
```

Importar una clave pública

Esta tarea carga la clave únicamente con parámetros públicos, tal como se creó con el botón `Export Public Key`, y la establece con el nombre del contenedor de claves.

Esta tarea simula un escenario en que Roberto carga la clave de Alicia que solo tiene parámetros públicos, de manera que pueda cifrar archivos por ella.

Agregue el siguiente código como controlador de eventos `Click` del botón `Import Public Key` (`buttonImportPublicKey_Click`).

```
void buttonImportPublicKey_Click(object sender, System.EventArgs e)
{
    StreamReader sr = new StreamReader(PubKeyFile);
    cspp.KeyContainerName = keyName;
    rsa = new RSACryptoServiceProvider(cspp);
    string keytxt = sr.ReadToEnd();
    rsa.FromXmlString(keytxt);
    rsa.PersistKeyInCsp = true;
    if (rsa.PublicOnly == true)
        label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only";
    else
        label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair";
    sr.Close();
}
```

```
Private Sub buttonImportPublicKey_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
buttonImportPublicKey.Click
    Dim sr As StreamReader = New StreamReader(PubKeyFile)
    cspp.KeyContainerName = keyName
    rsa = New RSACryptoServiceProvider(cspp)
    Dim keytxt As String = sr.ReadToEnd
    rsa.FromXmlString(keytxt)
    rsa.PersistKeyInCsp = True
    If rsa.PublicOnly = True Then
        Label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only"
    Else
        Label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair"
    End If
    sr.Close()
End Sub
```

Obtener una clave privada

Esta tarea establece el nombre del contenedor de claves con el nombre de la clave creada mediante el botón `Create Keys`. El contenedor de claves contendrá el par de claves completo con los parámetros privados.

Esta tarea simula un escenario en que Alicia usa su clave privada para descifrar archivos cifrados por Roberto.

Agregue el siguiente código como controlador de eventos `Click` del botón `Get Private Key` (`buttonGetPrivateKey_Click`).

```
private void buttonGetPrivateKey_Click(object sender, EventArgs e)
{
    cspp.KeyContainerName = keyName;

    rsa = new RSACryptoServiceProvider(cspp);
    rsa.PersistKeyInCsp = true;

    if (rsa.PublicOnly == true)
        label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only";
    else
        label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair";
}
```

```
Private Sub buttonGetPrivateKey_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles buttonGetPrivateKey.Click
    cspp.KeyContainerName = keyName
    rsa = New RSACryptoServiceProvider(cspp)
    rsa.PersistKeyInCsp = True
    If rsa.PublicOnly = True Then
        Label1.Text = "Key: " + cspp.KeyContainerName + " - Public Only"
    Else
        Label1.Text = "Key: " + cspp.KeyContainerName + " - Full Key Pair"
    End If
End Sub
```

Prueba de la aplicación

Después de haber compilado la aplicación, ejecute los siguientes escenarios de prueba.

Para crear claves, cifrar y descifrar

1. Haga clic en el botón `Create Keys`. La etiqueta muestra el nombre de la clave e indica que se trata de un par de claves completo.
2. Haga clic en el botón `Export Public Key`. Tenga en cuenta que al exportar los parámetros de la clave pública no se cambia la clave actual.
3. Haga clic en el botón `Encrypt File` y seleccione un archivo.
4. Haga clic en el botón `Decrypt File` y seleccione el archivo que acaba de cifrar.
5. Examine el archivo que acaba de descifrar.
6. Cierre la aplicación y reiníciela para intentar recuperar los contenedores de claves persistentes en el siguiente escenario.

Para cifrar con la clave pública

1. Haga clic en el botón `Import Public Key`. La etiqueta muestra el nombre de la clave e indica que solo es pública.
2. Haga clic en el botón `Encrypt File` y seleccione un archivo.
3. Haga clic en el botón `Decrypt File` y seleccione el archivo que acaba de cifrar. Esto generará un error porque debe disponer de la clave privada para efectuar el descifrado.

En este escenario se muestra una situación en la que solo se dispone de la clave pública para cifrar un archivo para otra persona. Normalmente, esa persona debe proporcionarle únicamente la clave pública y conservar la clave privada para el descifrado.

Para descifrar con la clave privada

1. Haga clic en el botón `Get Private Key`. La etiqueta muestra el nombre de la clave e indica si se trata del par de claves completo.
2. Haga clic en el botón `Decrypt File` y seleccione el archivo que acaba de cifrar. Esta acción se llevará a cabo correctamente porque dispone del par de claves completo para efectuar el descifrado.

Vea también

- Servicios criptográficos

Directrices de codificación segura

19/03/2020 • 8 minutes to read • Edit Online

La seguridad basada en pruebas y la seguridad de acceso por código ofrecen mecanismos explícitos y muy eficaces para implementar la seguridad. La mayoría del código de aplicación simplemente puede usar la infraestructura implementada por .NET. En algunos casos, se requiere la seguridad específica de aplicación adicional, creada ampliando el sistema de seguridad o mediante nuevos métodos ad hoc.

Con los permisos forzados de .NET y otras aplicaciones en el código, debe levantar barreras para evitar que el código malintencionado tenga acceso a información que no desea que tenga o realice otras acciones no deseadas. Además, debe lograr un equilibrio entre la seguridad y la facilidad de uso en todos los escenarios esperados mediante código de confianza.

Esta información general describe las diferentes formas en que se puede diseñar código para que funcione con el sistema de seguridad.

Protección del acceso a los recursos

Al diseñar y escribir el código, debe proteger y limitar el acceso del código a los recursos, especialmente cuando se usa o se invoca código de origen desconocido. Por lo tanto, tenga en cuenta las siguientes técnicas para asegurarse de que el código sea seguro:

- No utilice la seguridad de acceso del código (CAS).
- No utilice el código de confianza parcial.
- No utilice el atributo AllowPartiallyTrustedCaller (APTCA).
- No utilice .NET Remoting.
- No utilice el modelo de objetos de componente distribuido (DCOM).
- No utilice los formateadores binarios.

La seguridad de acceso de código y el código transparente de seguridad no se admiten como límite de seguridad con código de confianza parcial. Le aconsejamos que no cargue ni ejecute código de orígenes desconocidos sin contar con medidas de seguridad alternativas. Las medidas de seguridad alternativas son:

- Virtualización
- AppContainers
- Usuarios y permisos de sistema operativo (SO)
- Contenedores de Hyper-V

Código de seguridad neutral

El código neutral respecto a la seguridad no hace nada explícito con el sistema de seguridad. Se ejecuta con los permisos que recibe. Aunque las aplicaciones que no detectan las excepciones de seguridad asociadas con operaciones protegidas (como el uso de archivos, redes, etc.) pueden dar lugar a una excepción no controlada, el código neutro en seguridad sigue aprovechando las tecnologías de seguridad de .NET .

Una biblioteca neutral respecto a la seguridad tiene características especiales que debe conocer. Supongamos que la biblioteca proporciona elementos de API que usan archivos o llaman a código no administrado. Si el código no

tiene el permiso correspondiente, no se ejecutará como se describe. Sin embargo, incluso si el código tiene el permiso, cualquier código de aplicación que le llame debe tener el mismo permiso para poder funcionar. Si el código de llamada no tiene SecurityException el permiso correcto, aparece un como resultado del recorrido de la pila de seguridad de acceso de código.

Código de aplicación que no es un componente reutilizable

Si el código forma parte de una aplicación a la que no llamará otro código, la seguridad es sencilla y es posible que no se requiera una codificación especial. Sin embargo, recuerde que el código malintencionado puede llamar a su código. Aunque la seguridad de acceso a código puede evitar que código malintencionado obtenga acceso a recursos, dicho código todavía podría leer valores de sus campos o propiedades que puedan contener información confidencial.

Además, si el código acepta la entrada del usuario desde Internet o de otras fuentes no confiables, debe tener cuidado con la entrada malintencionada.

Contenedor administrado para la implementación de código nativo

Normalmente en este escenario se implementa alguna funcionalidad de utilidad en código nativo que quiere que esté disponible para código administrado. Los contenedores administrados son sencillos de escribir mediante la invocación de plataforma o interoperabilidad COM. Sin embargo, si lo hace, los llamadores de los contenedores deben tener derechos de código no administrado para ser correctos. En la directiva predeterminada, esto significa que el código descargado de una intranet o Internet no funcionará con los contenedores.

En lugar de conceder derechos de código no administrado a todas las aplicaciones que usan estos contenedores, es mejor conceder estos derechos solo al código contenedor. Si la funcionalidad subyacente no expone ningún recurso y la implementación es igual de segura, el contenedor solo necesita imponer sus derechos, lo que permite a cualquier código llamar a través de él. Cuando se trate de recursos, la codificación de seguridad debe ser la mismo que el caso de código de biblioteca descrito en la siguiente sección. Dado que el contenedor puede exponer a los llamadores a esos recursos, se necesita una cuidadosa comprobación de la seguridad del código nativo, lo que es responsabilidad del contenedor.

Código de biblioteca que expone recursos protegidos

El siguiente enfoque es el más eficaz y, por lo tanto, potencialmente peligroso (si se hace incorrectamente) para la codificación de seguridad: la biblioteca sirve como interfaz para que otro código tenga acceso a determinados recursos que de otro modo no están disponibles, del igual que las clases .NET aplican la aplicación de la aplicación. permisos para los recursos que utilizan. Siempre que exponga un recurso, su código debe requerir primero el permiso adecuado para el recurso (es decir, debe realizar una comprobación de seguridad) y luego declarar sus derechos para llevar a cabo la operación real.

Temas relacionados

TÍTULO	DESCRIPCIÓN
Proteger los datos de estado	Se describe cómo proteger los miembros privados.
Seguridad e introducción de datos por el usuario	Se describen cuestiones de seguridad para las aplicaciones que aceptan la entrada del usuario.
Seguridad y condiciones de carrera	Se describe cómo evitar condiciones de anticipación en el código.

TÍTULO	DESCRIPCIÓN
Seguridad y generación de código inmediata	Se describen cuestiones de seguridad para las aplicaciones que generan código dinámico.
Seguridad basada en roles	Describe detalladamente la seguridad basada en roles de .NET y proporciona instrucciones para usarla en el código.

Proteger los datos de estado

02/07/2020 • 2 minutes to read • Edit Online

Las aplicaciones que administran datos confidenciales o realizan cualquier tipo de decisiones de seguridad necesitan mantener los datos bajo control y no pueden permitir que otro código potencialmente malintencionado acceda directamente a los datos. La mejor manera de proteger los datos en memoria es declarar los datos como variables privadas o internas (con el ámbito limitado al mismo ensamblado). Sin embargo, incluso estos datos están sujetos a acceso, por lo que debe tenerse en cuenta lo siguiente:

- Al usar mecanismos de reflexión, se puede obtener y establecer miembros privados en el código de plena confianza al que puede hacer referencia el objeto.
- Con la serialización, el código de plena confianza puede obtener y establecer eficazmente miembros privados si pueden acceder a los datos correspondientes en el formulario serializado del objeto.
- En la depuración, se pueden leer estos datos.

Asegúrese de que ninguno de sus métodos o propiedades expone estos valores de manera involuntaria.

Consulte también

- Instrucciones de codificación segura

Seguridad e introducción de datos por el usuario

02/07/2020 • 4 minutes to read • Edit Online

Los datos de usuario, que consisten en cualquier tipo de entrada (datos de una solicitud web o dirección URL, entrada a los controles de una aplicación de Microsoft Windows Forms etc.), pueden afectar negativamente al código, porque a menudo se utilizan esos datos directamente como parámetros para llamar a otro código. Esta situación es análoga al código malintencionado que llama a su código con parámetros extraños, y se deben tomar las mismas precauciones. La entrada del usuario es en realidad más difícil de proteger, porque no hay ningún marco de pila para realizar el seguimiento de la presencia de datos que no sean de confianza.

Estos son los errores de seguridad más sutiles y más difíciles de localizar porque, aunque pueden existir en el código aparentemente no relacionado con la seguridad, son una puerta de enlace para que pasen datos maliciosos a otro código. Para buscar estos errores, siga cualquier tipo de datos de entrada, imagine cuál puede ser el intervalo de valores posibles y considere si el código que ve estos datos puede controlar todos esos casos. Puede corregir estos errores mediante la comprobación del intervalo y el rechazo de cualquier entrada que no puede controlar el código.

A continuación se indican algunas consideraciones importantes que afectan a los datos de usuario:

- Los datos de usuario de una respuesta del servidor se ejecutan en el contexto del sitio del servidor en el cliente. Si el servidor Web toma los datos de usuario y los inserta en la página web devuelta, podría incluir, por ejemplo, una **<script>** etiqueta y ejecutar como si desde el servidor.
- Recuerde que el cliente puede solicitar cualquier dirección URL.
- Tenga en cuenta las rutas de acceso complicadas o no válidas:
  - .., rutas de acceso extremadamente largas.
  - Uso de caracteres comodín (*).
  - Expansión de token (%token%).
  - Formatos extraños de rutas de acceso con un significado especial.
  - Nombres de flujos de sistema de archivo alternativos, como `filename::$DATA`.
  - Versiones cortas de los nombres de archivo como `longfi~1` para `longfilename`.
- Recuerde que Eval(userdata) puede hacer cualquier cosa.
- Tenga cuidado de los enlaces tardíos en un nombre que incluye algunos datos de usuario.
- Si está trabajando con datos de web, tenga en cuenta las distintas secuencias de escape permitidas, como:
  - Secuencias de escape hexadecimales (%nn).
  - Secuencias de escape Unicode (%nnn).
  - Secuencias de escape UTF-8 excesivamente largas (%nn%nn).
  - Secuencias de escape dobles (%nn se convierte en %mmnn, donde %mm es el escape para '%').
- Tenga cuidado con los nombres de usuario que pueden tener más de un formato canónico. Por ejemplo, a menudo puede utilizar cualquier forma de MIDOMINIO*nombreUsuario* o de *nombreUsuario*@mydomain.example.com.

Consulte también

- Instrucciones de codificación segura

Seguridad y condiciones de carrera

02/07/2020 • 4 minutes to read • Edit Online

Otro área de preocupación es el potencial de los agujeros de seguridad aprovechados por las condiciones de carrera. Hay varias maneras en las que esto puede ocurrir. En los subtemas siguientes se describen algunos de los principales problemas que debe evitar el desarrollador.

Condiciones de carrera en el método Dispose

Si el método **Dispose** de una clase (para obtener más información, consulte recolección de elementos no utilizados) no está sincronizado, es posible que el código de limpieza dentro de **Dispose** se ejecute más de una vez, tal como se muestra en el ejemplo siguiente.

```
Sub Dispose()
    If Not (myObj Is Nothing) Then
       Cleanup(myObj)
       myObj = Nothing
    End If
End Sub
```

```
void Dispose()
{
    if (myObj != null)
    {
        Cleanup(myObj);
        myObj = null;
    }
}
```

Dado que esta implementación de **Dispose** no está sincronizada, es posible que la `Cleanup` llame primero a un subproceso y, después, un segundo subproceso antes de `_myObj` que se establezca en **null**. Tanto si se trata de un problema de seguridad depende de lo que ocurre cuando `Cleanup` se ejecuta el código. Un problema importante con las implementaciones de **Dispose** no sincronizadas implica el uso de identificadores de recursos como los archivos. Una eliminación incorrecta puede provocar que se use un identificador incorrecto, lo que a menudo conduce a vulnerabilidades de seguridad.

Condiciones de carrera en constructores

En algunas aplicaciones, es posible que otros subprocesos accedan a los miembros de clase antes de que sus constructores de clase se ejecuten por completo. Debe revisar todos los constructores de clase para asegurarse de que no hay ningún problema de seguridad si esto ocurre, o sincronizar los subprocesos si es necesario.

Condiciones de carrera con objetos almacenados en caché

El código que almacena en memoria caché la información de seguridad o usa la operación de aserción de seguridad de acceso del código también podría ser vulnerable a las condiciones de carrera si otras partes de la clase no se sincronizan correctamente, como se muestra en el ejemplo siguiente.

```
Sub SomeSecureFunction()
    If SomeDemandPasses() Then
        fCallersOk = True
        DoOtherWork()
        fCallersOk = False
    End If
End Sub

Sub DoOtherWork()
    If fCallersOK Then
        DoSomethingTrusted()
    Else
        DemandSomething()
        DoSomethingTrusted()
    End If
End Sub
```

```
void SomeSecureFunction()
{
    if (SomeDemandPasses())
    {
        fCallersOk = true;
        DoOtherWork();
        fCallersOk = false;
    }
}
void DoOtherWork()
{
    if (fCallersOK)
    {
        DoSomethingTrusted();
    }
    else
    {
        DemandSomething();
        DoSomethingTrusted();
    }
}
```

Si hay otras rutas de acceso a `DoOtherWork` las que se puede llamar desde otro subproceso con el mismo objeto, un llamador que no es de confianza puede posponerse después de una solicitud.

Si el código almacena en caché la información de seguridad, asegúrese de revisarla para ver esta vulnerabilidad.

Condiciones de carrera en finalizadores

También se pueden producir condiciones de carrera en un objeto que hace referencia a un recurso estático o no administrado que, a continuación, libera en su finalizador. Si varios objetos comparten un recurso que se manipula en el finalizador de una clase, los objetos deben sincronizar todo el acceso a ese recurso.

Consulte también

- Instrucciones de codificación segura

Seguridad y generación de código inmediata

02/07/2020 • 2 minutes to read • Edit Online

Algunas bibliotecas funcionan generando código y ejecutándolo para realizar algunas operaciones para el llamador. El problema fundamental es generar código en nombre de código de menor confianza y ejecutarlo con una confianza superior. El problema empeora cuando el llamador puede influir en la generación de código, por lo que debe asegurarse de generar solo código que considere seguro.

Necesitará saber exactamente qué código genera en todo momento. Esto significa que debe tener controles estrictos sobre los valores que obtiene de un usuario, ya sean cadenas entre comillas (que se deben escribir entre caracteres de escape para que no puedan incluir elementos de código imprevistos), identificadores (que se deben comprobar para verificar que son identificadores válidos), o cualquier otra cosa. Los identificadores pueden ser peligrosos porque un ensamblado compilado puede modificarse para que sus identificadores contengan caracteres extraños que probablemente lo interrumpirán (aunque esta es una vulnerabilidad de seguridad muy poco frecuente).

Es conveniente generar código con emisión de la reflexión porque suele ayudar a evitar muchos de estos problemas.

Al compilar el código, tenga en cuenta si hay alguna manera de que un programa malintencionado lo modifique. ¿Existe algún momento, por breve que sea, durante el cual un código malintencionado puede cambiar el código fuente en el disco antes de que el compilador lo lea o antes de que el código cargue el archivo .dll? Si es así, debe proteger el directorio que contiene estos archivos usando una lista de control de acceso en el sistema de archivos, según corresponda.

Consulte también

- Instrucciones de codificación segura

Vulnerabilidades de temporalización con descifrado simétrico en modo CBC al usar el relleno

19/03/2020 • 38 minutes to read • Edit Online

Microsoft cree que ya no es seguro descifrar los datos cifrados con el modo Cipher-Block-Chaining (CBC) de cifrado simétrico cuando se ha aplicado relleno verificable sin garantizar primero la integridad del texto cifrado, excepto para muy específicos Circunstancias. Este juicio se basa en la investigación criptográfica actualmente conocida.

Introducción

Un ataque de oráculo de relleno es un tipo de ataque contra datos cifrados que permite al atacante descifrar el contenido de los datos, sin conocer la clave.

Un oráculo hace referencia a un "tell" que proporciona a un atacante información sobre si la acción que está ejecutando es correcta o no. Imagínese jugar un juego de mesa o de cartas con un niño. Cuando su cara se ilumina con una gran sonrisa porque piensan que están a punto de hacer un buen movimiento, eso es un oráculo. Usted, como el oponente, puede utilizar este oráculo para planificar su próximo movimiento apropiadamente.

El relleno es un término criptográfico específico. Algunos cifrados, que son los algoritmos utilizados para cifrar los datos, funcionan en bloques de datos donde cada bloque tiene un tamaño fijo. Si los datos que desea cifrar no son del tamaño adecuado para rellenar los bloques, los datos se rellenan hasta que lo hacen. Muchas formas de relleno requieren que el relleno esté siempre presente, incluso si la entrada original era del tamaño correcto. Esto permite que el relleno siempre se elimine de forma segura al descifrar.

Al poner las dos cosas juntas, una implementación de software con un oráculo de relleno revela si los datos descifrados tienen un relleno válido. El oráculo podría ser algo tan simple como devolver un valor que dice "Relleno no válido" o algo más complicado como tomar un tiempo mediblemente diferente para procesar un bloque válido en lugar de un bloque no válido.

Los cifrados basados en bloques tienen otra propiedad, denominada modo, que determina la relación de datos del primer bloque con los datos del segundo bloque, etc. Uno de los modos más utilizados es CBC. CBC introduce un bloque aleatorio inicial, conocido como vector de inicialización (IV), y combina el bloque anterior con el resultado del cifrado estático para que sea de tal manera que cifrar el mismo mensaje con la misma clave no siempre produce la misma salida cifrada.

Un atacante puede usar un oráculo de relleno, en combinación con cómo se estructuran los datos de CBC, para enviar mensajes ligeramente modificados al código que expone el oráculo y seguir enviando datos hasta que el oráculo les diga que los datos son correctos. A partir de esta respuesta, el atacante puede descifrar el mensaje byte a byte.

Las redes informáticas modernas son de tan alta calidad que un atacante puede detectar diferencias muy pequeñas (menos de 0,1 ms) en el tiempo de ejecución en sistemas remotos.Las aplicaciones que asumen que un descifrado correcto solo puede ocurrir cuando los datos no se manipulan pueden ser vulnerables a ataques de herramientas diseñadas para observar diferencias en el descifrado exitoso y sin éxito. Si bien esta diferencia de tiempo puede ser más significativa en algunos idiomas o bibliotecas que en otros, ahora se cree que se trata de una amenaza práctica para todos los lenguajes y bibliotecas cuando se tiene en cuenta la respuesta de la aplicación al error.

Este ataque se basa en la capacidad de cambiar los datos cifrados y probar el resultado con el oráculo. La única manera de mitigar completamente el ataque es detectar cambios en los datos cifrados y negarse a realizar cualquier acción en él. La forma estándar de hacerlo es crear una firma para los datos y validar esa firma antes de realizar cualquier operación. La firma debe ser verificable, no puede ser creada por el atacante, de lo contrario

cambiarían los datos cifrados y, a continuación, calcularían una nueva firma en función de los datos modificados. Un tipo común de firma adecuada se conoce como código de autenticación de mensajes hash con clave (HMAC). Un HMAC difiere de una suma de comprobación en que toma una clave secreta, conocida sólo por la persona que produce el HMAC y por la persona que lo valida. Sin la posesión de la llave, no se puede producir un HMAC correcto. Cuando reciba los datos, tomaría los datos cifrados, calcularía de forma independiente el HMAC con la clave secreta que usted y el remitente comparten y, a continuación, compararía el HMAC que enviaron con el que calculó. Esta comparación debe ser tiempo constante, de lo contrario ha agregado otro oráculo detectable, lo que permite un tipo diferente de ataque.

En resumen, para utilizar cifrados de bloques CBC acolchados de forma segura, debe combinarlos con un HMAC (u otra comprobación de integridad de datos) que valide mediante una comparación de tiempo constante antes de intentar descifrar los datos. Puesto que todos los mensajes alterados tardan la misma cantidad de tiempo en producir una respuesta, se evita el ataque.

Quién es vulnerable

Esta vulnerabilidad se aplica tanto a las aplicaciones administradas como a las nativas que realizan su propio cifrado y descifrado. Esto incluye, por ejemplo:

- Una aplicación que cifra una cookie para su posterior descifrado en el servidor.
- Una aplicación de base de datos que proporciona a los usuarios la capacidad de insertar datos en una tabla cuyas columnas se descifran más adelante.
- Una aplicación de transferencia de datos que se basa en el cifrado mediante una clave compartida para proteger los datos en tránsito.
- Una aplicación que cifra y descifra los mensajes "dentro" del túnel TLS.

Tenga en cuenta que el uso de TLS por sí solo puede no protegerle en estos escenarios.

Una aplicación vulnerable:

- Descifra los datos mediante el modo de cifrado CBC con un modo de relleno verificable, como PKCS-7 o ANSI X.923.
- Realiza el descifrado sin haber realizado una comprobación de integridad de datos (a través de un MAC o una firma digital asimétrica).

Esto también se aplica a las aplicaciones creadas sobre abstracciones sobre estos primitivos, como la estructura EnvelopedData de sintaxis de mensajes criptográficos (PKCS-7/CMS).

Temas de preocupación relacionados

La investigación ha llevado a Microsoft a preocuparse más por los mensajes CBC que están acolchados con relleno equivalente a ISO 10126 cuando el mensaje tiene una estructura de pie de página conocida o predecible. Por ejemplo, contenido preparado bajo las reglas de la recomendación de procesamiento y sintaxis de cifrado XML de W3C (xmlenc, EncryptedXml). Aunque la guía de W3C para firmar el mensaje y luego cifrar se consideraba adecuada en ese momento, Microsoft ahora recomienda realizar siempre el cifrado y luego firmar.

Los desarrolladores de aplicaciones siempre deben tener en cuenta la comprobación de la aplicabilidad de una clave de firma asimétrica, ya que no hay ninguna relación de confianza inherente entre una clave asimétrica y un mensaje arbitrario.

Detalles

Históricamente, ha habido consenso en que es importante cifrar y autenticar datos importantes, utilizando medios como firmas HMAC o RSA. Sin embargo, ha habido una guía menos clara sobre cómo secuenciar las operaciones de cifrado y autenticación. Debido a la vulnerabilidad detallada en este artículo, la guía de Microsoft es ahora usar

siempre el paradigma "cifrar entonces-firmar". Es decir, primero cifre los datos mediante una clave simétrica y, a continuación, calcule una firma MAC o asimétrica sobre el texto cifrado (datos cifrados). Al descifrar datos, realice lo contrario. Primero, confirme el MAC o la firma del texto cifrado, después descifrelo.

Se sabe que existe una clase de vulnerabilidades conocidas como "ataques de oráculos de relleno" durante más de 10 años. Estas vulnerabilidades permiten a un atacante descifrar datos cifrados por algoritmos de bloques simétricos, como AES y 3DES, utilizando no más de 4096 intentos por bloque de datos. Estas vulnerabilidades hacen uso del hecho de que los cifrados de bloques se utilizan con más frecuencia con datos de relleno verificables al final. Se encontró que si un atacante puede manipular el texto cifrado y averiguar si la manipulación causó un error en el formato del relleno al final, el atacante puede descifrar los datos.

Inicialmente, los ataques prácticos se basaban en servicios que devolverían diferentes códigos de error en función de si el relleno era válido, como la vulnerabilidad ASP.NET MS10-070. Sin embargo, Microsoft ahora cree que es práctico llevar a cabo ataques similares utilizando solo las diferencias de tiempo entre el procesamiento de relleno válido y no válido.

Siempre que el esquema de cifrado emplee una firma y que la verificación de la firma se realice con un tiempo de ejecución fijo para una longitud determinada de datos (independientemente del contenido), la integridad de los datos se puede verificar sin emitir ninguna información a un atacante a través de un canal lateral. Puesto que la comprobación de integridad rechaza los mensajes manipulados, se mitiga la amenaza de oráculo de relleno.

Guía

En primer lugar, Microsoft recomienda que los datos que tienen necesidades de confidencialidad se transmitan a través de Transport Layer Security (TLS), el sucesor de Secure Sockets Layer (SSL).

A continuación, analice la aplicación para:

- Entienda con precisión qué cifrado está realizando y qué cifrado proporcionan las plataformas y las API que está utilizando.
- Asegúrese de que cada uso en cada capa de un algoritmo de cifrado de bloquessimétricos, como AES y 3DES, en modo CBC incorpore el uso de una comprobación de integridad de datos con clave secreta (una firma asimétrica, una HMAC, o para cambiar el modo de cifrado a un modo de cifrado autenticado (AE) como GCM o CCM).

Basado en la investigación actual, generalmente se cree que cuando los pasos de autenticación y cifrado se realizan de forma independiente para los modos de cifrado que no son AE, autenticar el texto cifrado (cifrar-entonces-signo) es la mejor opción general. Sin embargo, no hay una respuesta única correcta a la criptografía y esta generalización no es tan buena como el consejo dirigido de un criptógrafo profesional.

Se recomienda a las aplicaciones que no pueden cambiar su formato de mensajería pero realizar el descifrado CBC no autenticado que intenten incorporar mitigaciones como:

- Descifrar sin permitir que el descifrador para verificar o eliminar el relleno:
  - Cualquier relleno que se aplicó todavía debe eliminarse o ignorarse, está moviendo la carga a la aplicación.
  - La ventaja es que la verificación y eliminación de relleno se puede incorporar a otra lógica de verificación de datos de aplicación. Si la verificación de relleno y la verificación de datos se pueden hacer en tiempo constante, la amenaza se reduce.
  - Puesto que la interpretación del relleno cambia la longitud del mensaje percibida, todavía puede haber información de sincronización emitida por este enfoque.
- Cambie el modo de relleno de descifrado a ISO10126:
  - El relleno de descifrado ISO10126 es compatible con el relleno de cifrado PKCS7 y el relleno de cifrado ANSIX923.
  - Al cambiar el modo, se reduce el conocimiento del oráculo de relleno a 1 byte en lugar de a todo el

bloque. Sin embargo, si el contenido tiene un pie de página conocido, como un elemento XML de cierre, los ataques relacionados pueden seguir atacando el resto del mensaje.
    - Esto tampoco impide la recuperación de texto no cifrado en situaciones en las que el atacante puede coaccionar el mismo texto sin formato para que se cifre varias veces con un desplazamiento de mensaje diferente.
- Puerta de la evaluación de una llamada de descifrado para amortiguar la señal de sincronización:
    - El cálculo del tiempo de espera debe tener un mínimo que exceda la cantidad máxima de tiempo que la operación de descifrado tardaría para cualquier segmento de datos que contenga relleno.
    - Los cálculos de tiempo deben realizarse de acuerdo con las instrucciones Environment.TickCount de Adquisición de marcas de tiempo de alta resolución,no mediante el uso (sujeto a roll-over/overflow) o restando dos marcas de tiempo del sistema (sujeto a errores de ajuste NTP).
    - Los cálculos de tiempo deben incluir la operación de descifrado, incluidas todas las posibles excepciones en aplicaciones administradas o C++, no solo rellenadas al final.
    - Si el éxito o el error se ha determinado todavía, la puerta de sincronización necesita devolver el error cuando expira.
- Los servicios que realizan el descifrado no autenticado deben tener supervisión para detectar que ha llegado a través de una avalancha de mensajes "no válidos".
    - Tenga en cuenta que esta señal lleva tanto falsos positivos (datos legítimamente dañados) como falsos negativos (difundiendo el ataque durante un tiempo suficientemente largo como para evadir la detección).

Búsqueda de código vulnerable - aplicaciones nativas

Para programas creados en la biblioteca criptográfica de Windows: próxima generación (CNG):

- La llamada de descifrado es a BCryptDecrypt, especificando la `BCRYPT_BLOCK_PADDING` marca.
- El identificador de clave se ha inicializado BCRYPT_CHAINING_MODE llamando `BCRYPT_CHAIN_MODE_CBC` a BCryptSetProperty con BCRYPT_CHAINING_MODE establecido en .
    - Dado `BCRYPT_CHAIN_MODE_CBC` que es el valor predeterminado, es `BCRYPT_CHAINING_MODE` posible que el código afectado no haya asignado ningún valor para .

Para los programas creados con la API criptográfica de Windows anterior:

- La llamada de descifrado es `Final=TRUE` a CryptDecrypt con .
- El identificador de clave se ha inicializado KP_MODE llamando a `CRYPT_MODE_CBC` CryptSetKeyParam con KP_MODE establecido en .
    - Dado `CRYPT_MODE_CBC` que es el valor predeterminado, es `KP_MODE` posible que el código afectado no haya asignado ningún valor para .

Búsqueda de código vulnerable - aplicaciones administradas

- La llamada de descifrado CreateDecryptor() CreateDecryptor(Byte[], Byte[]) es System.Security.Cryptography.SymmetricAlgorithma los métodos o en .
    - Esto incluye los siguientes tipos derivados dentro de .NET, pero también puede incluir tipos de terceros:
        - Aes
        - AesCng
        - AesCryptoServiceProvider
        - AesManaged
        - DES
        - DESCryptoServiceProvider
        - RC2

    - RC2CryptoServiceProvider
    - Rijndael
    - RijndaelManaged
    - TripleDES
    - TripleDESCng
    - TripleDESCryptoServiceProvider
- La SymmetricAlgorithm.Padding propiedad se PaddingMode.PKCS7 PaddingMode.ANSIX923estableció PaddingMode.ISO10126en , , o .
    - Dado PaddingMode.PKCS7 que es el valor predeterminado, SymmetricAlgorithm.Padding es posible que el código afectado nunca haya asignado la propiedad.
- La SymmetricAlgorithm.Mode propiedad se estableció enCipherMode.CBC
    - Dado CipherMode.CBC que es el valor predeterminado, SymmetricAlgorithm.Mode es posible que el código afectado nunca haya asignado la propiedad.

Búsqueda de código vulnerable: sintaxis de mensajes criptográficos

Un mensaje envuelto CMS no autenticado cuyo contenido cifrado utiliza el modo CBC de AES (2.16.840.1.101.3.4.1.2, 2.16.840.1.101.3.4.1.22, 2.16.840.1.101.3.4.1.42), DES (1.3.14.3.2.7), 3DES (1.2.840.113549.3.7) o RC2 (1.2.840.113549.3.2) es vulnerables, así como mensajes que utilizan cualquier otro algoritmo de cifrado de bloques en modo CBC.

Aunque los cifrados de secuencia sin vulnerabilidad a esta vulnerabilidad en particular, Microsoft recomienda autenticar siempre los datos sobre la inspección del valor ContentEncryptionAlgorithm.

Para las aplicaciones administradas, se puede detectar un blob EndData de CMS como cualquier valor que se pasa a System.Security.Cryptography.Pkcs.EnvelopedCms.Decode(Byte[]).

Para aplicaciones nativas, un blob EndData de CMS se puede detectar como cualquier `CMSG_ENVELOPED` valor proporcionado a un identificador de `CMSG_CTRL_DECRYPT` CMS a través de CryptMsgUpdate cuyo CMSG_TYPE_PARAM resultante es y/o el identificador cmS se envía más adelante una instrucción a través de CryptMsgControl.

Ejemplo de código vulnerable - administrado

Este método lee una cookie y la descifra y no hay ninguna comprobación de integridad de datos visible. Por lo tanto, el contenido de una cookie que es leído por este método puede ser atacado por el usuario que la recibió, o por cualquier atacante que haya obtenido el valor de cookie cifrado.

```
private byte[] DecryptCookie(string cookieName)
{
    HttpCookie cookie = Request.Cookies[cookieName];

    if (cookie == null)
    {
        return null;
    }

    using (ICryptoTransform decryptor = _aes.CreateDecryptor())
    using (MemoryStream memoryStream = new MemoryStream())
    using (CryptoStream cryptoStream =
        new CryptoStream(memoryStream, decryptor, CryptoStreamMode.Write))
    {
        byte[] readCookie = Convert.FromBase64String(cookie.Value);
        cryptoStream.Write(readCookie, 0, readCookie.Length);
        cryptoStream.FlushFinalBlock();
        return memoryStream.ToArray();
    }
}
```

Ejemplo de código que sigue a las prácticas recomendadas - administrado

El siguiente código de ejemplo utiliza un formato de mensaje no estándar de

`cipher_algorithm_id || hmac_algorithm_id || hmac_tag || iv || ciphertext`

donde `cipher_algorithm_id` los `hmac_algorithm_id` identificadores de algoritmo y los identificadores de algoritmo son representaciones locales de la aplicación (no estándar) de esos algoritmos. Estos identificadores pueden tener sentido en otras partes del protocolo de mensajería existente en lugar de como una secuencia de bytes concatenada desnuda.

Este ejemplo también utiliza una sola clave maestra para derivar una clave de cifrado y una clave HMAC. Esto se proporciona tanto como una comodidad para convertir una aplicación con clave escote en una aplicación de doble clave, y para fomentar mantener las dos claves como valores diferentes. Además, garantiza que la clave HMAC y la clave de cifrado no pueden salir de la sincronización.

Este ejemplo no acepta Stream un para cifrado o descifrado. El formato de datos actual dificulta el cifrado de una pasada porque el `hmac_tag` valor precede al texto cifrado. Sin embargo, este formato se eligió porque mantiene todos los elementos de tamaño fijo al principio para mantener el analizador más simple. Con este formato de datos, el descifrado de una sola pasada es posible, aunque se advierte a un implementador que llame a GetHashAndReset y compruebe el resultado antes de llamar a TransformFinalBlock. Si el cifrado de streaming es importante, puede ser necesario un modo AE diferente.

```
// ==++==
//
//   Copyright (c) Microsoft Corporation.  All rights reserved.
//
//   Shared under the terms of the Microsoft Public License,
//   https://opensource.org/licenses/MS-PL
//
// ==--==

using System;
using System.Diagnostics;
using System.IO;
using System.Runtime.CompilerServices;
using System.Security.Cryptography;
```

```
namespace Microsoft.Examples.Cryptography
{
    public enum AeCipher : byte
    {
        Unknown,
        Aes256CbcPkcs7,
    }

    public enum AeMac : byte
    {
        Unknown,
        HMACSHA256,
        HMACSHA384,
    }

    /// <summary>
    /// Provides extension methods to make HashAlgorithm look like .NET Core's
    /// IncrementalHash
    /// </summary>
    internal static class IncrementalHashExtensions
    {
        public static void AppendData(this HashAlgorithm hash, byte[] data)
        {
            hash.TransformBlock(data, 0, data.Length, null, 0);
        }

        public static void AppendData(
            this HashAlgorithm hash,
            byte[] data,
            int offset,
            int length)
        {
            hash.TransformBlock(data, offset, length, null, 0);
        }

        public static byte[] GetHashAndReset(this HashAlgorithm hash)
        {
            hash.TransformFinalBlock(Array.Empty<byte>(), 0, 0);
            return hash.Hash;
        }
    }

    public static partial class AuthenticatedEncryption
    {
        /// <summary>
        /// Use <paramref name="masterKey"/> to derive two keys (one cipher, one HMAC)
        /// to provide authenticated encryption for <paramref name="message"/>.
        /// </summary>
        /// <param name="masterKey">The master key from which other keys derive.</param>
        /// <param name="message">The message to encrypt</param>
        /// <returns>
        /// A concatenation of
        /// [cipher algorithm+chainmode+padding][mac algorithm][authtag][IV][ciphertext],
        /// suitable to be passed to <see cref="Decrypt"/>.
        /// </returns>
        /// <remarks>
        /// <paramref name="masterKey"/> should be a 128-bit (or bigger) value generated
        /// by a secure random number generator, such as the one returned from
        /// <see cref="RandomNumberGenerator.Create()"/>.
        /// This implementation chooses to block deficient inputs by length, but does not
        /// make any attempt at discerning the randomness of the key.
        ///
        /// If the master key is being input by a prompt (like a password/passphrase)
        /// then it should be properly turned into keying material via a Key Derivation
        /// Function like PBKDF2, represented by Rfc2898DeriveBytes. A 'password' should
        /// never be simply turned to bytes via an Encoding class and used as a key.
        /// </remarks>
        public static byte[] Encrypt(byte[] masterKey, byte[] message)
        {
```

```
            if (masterKey == null)
                throw new ArgumentNullException(nameof(masterKey));
            if (masterKey.Length < 16)
                throw new ArgumentOutOfRangeException(
                    nameof(masterKey),
                    "Master Key must be at least 128 bits (16 bytes)");
            if (message == null)
                throw new ArgumentNullException(nameof(message));

            // First, choose an encryption scheme.
            AeCipher aeCipher = AeCipher.Aes256CbcPkcs7;

            // Second, choose an authentication (message integrity) scheme.
            //
            // In this example we use the master key length to change from HMACSHA256 to
            // HMACSHA384, but that is completely arbitrary. This mostly represents a
            // "cryptographic needs change over time" scenario.
            AeMac aeMac = masterKey.Length < 48 ? AeMac.HMACSHA256 : AeMac.HMACSHA384;

            // It's good to be able to identify what choices were made when a message was
            // encrypted, so that the message can later be decrypted. This allows for
            // future versions to add support for new encryption schemes, but still be
            // able to read old data. A practice known as "cryptographic agility".
            //
            // This is similar in practice to PKCS#7 messaging, but this uses a
            // private-scoped byte rather than a public-scoped Object IDentifier (OID).
            // Please note that the scheme in this example adheres to no particular
            // standard, and is unlikely to survive to a more complete implementation in
            // the .NET Framework.
            //
            // You may be well-served by prepending a version number byte to this
            // message, but may want to avoid the value 0x30 (the leading byte value for
            // DER-encoded structures such as X.509 certificates and PKCS#7 messages).
            byte[] algorithmChoices = { (byte)aeCipher, (byte)aeMac };
            byte[] iv;
            byte[] cipherText;
            byte[] tag;

            // Using our algorithm choices, open an HMAC (as an authentication tag
            // generator) and a SymmetricAlgorithm which use different keys each derived
            // from the same master key.
            //
            // A custom implementation may very well have distinctly managed secret keys
            // for the MAC and cipher, this example merely demonstrates the master to
            // derived key methodology to encourage key separation from the MAC and
            // cipher keys.
            using (HMAC tagGenerator = GetMac(aeMac, masterKey))
            {
                using (SymmetricAlgorithm cipher = GetCipher(aeCipher, masterKey))
                using (ICryptoTransform encryptor = cipher.CreateEncryptor())
                {
                    // Since no IV was provided, a random one has been generated
                    // during the call to CreateEncryptor.
                    //
                    // But note that it only does the auto-generation once. If the cipher
                    // object were used again, a call to GenerateIV would have been
                    // required.
                    iv = cipher.IV;

                    cipherText = Transform(encryptor, message, 0, message.Length);
                }

                // The IV and ciphertest both need to be included in the MAC to prevent
                // tampering.
                //
                // By including the algorithm identifiers, we have technically moved from
                // simple Authenticated Encryption (AE) to Authenticated Encryption with
                // Additional Data (AEAD). By including the algorithm identifiers in the
                // MAC, it becomes harder for an attacker to change them as an attempt to
```

```
            // perform a downgrade attack.
            //
            // If you've added a data format version field, it can also be included
            // in the MAC to further inhibit an attacker's options for confusing the
            // data processor into believing the tampered message is valid.
            tagGenerator.AppendData(algorithmChoices);
            tagGenerator.AppendData(iv);
            tagGenerator.AppendData(cipherText);
            tag = tagGenerator.GetHashAndReset();
        }

        // Build the final result as the concatenation of everything except the keys.
        int totalLength =
            algorithmChoices.Length +
            tag.Length +
            iv.Length +
            cipherText.Length;

        byte[] output = new byte[totalLength];
        int outputOffset = 0;

        Append(algorithmChoices, output, ref outputOffset);
        Append(tag, output, ref outputOffset);
        Append(iv, output, ref outputOffset);
        Append(cipherText, output, ref outputOffset);

        Debug.Assert(outputOffset == output.Length);
        return output;
    }

    /// <summary>
    /// Reads a message produced by <see cref="Encrypt"/> after verifying it hasn't
    /// been tampered with.
    /// </summary>
    /// <param name="masterKey">The master key from which other keys derive.</param>
    /// <param name="cipherText">
    /// The output of <see cref="Encrypt"/>: a concatenation of a cipher ID, mac ID,
    /// authTag, IV, and cipherText.
    /// </param>
    /// <returns>The decrypted content.</returns>
    /// <exception cref="ArgumentNullException">
    /// <paramref name="masterKey"/> is <c>null</c>.
    /// </exception>
    /// <exception cref="ArgumentNullException">
    /// <paramref name="cipherText"/> is <c>null</c>.
    /// </exception>
    /// <exception cref="CryptographicException">
    /// <paramref name="cipherText"/> identifies unknown algorithms, is not long
    /// enough, fails a data integrity check, or fails to decrypt.
    /// </exception>
    /// <remarks>
    /// <paramref name="masterKey"/> should be a 128-bit (or larger) value
    /// generated by a secure random number generator, such as the one returned from
    /// <see cref="RandomNumberGenerator.Create()"/>. This implementation chooses to
    /// block deficient inputs by length, but doesn't make any attempt at
    /// discerning the randomness of the key.
    ///
    /// If the master key is being input by a prompt (like a password/passphrase),
    /// then it should be properly turned into keying material via a Key Derivation
    /// Function like PBKDF2, represented by Rfc2898DeriveBytes. A 'password' should
    /// never be simply turned to bytes via an Encoding class and used as a key.
    /// </remarks>
    public static byte[] Decrypt(byte[] masterKey, byte[] cipherText)
    {
        // This example continues the .NET practice of throwing exceptions for
        // failures. If you consider message tampering to be normal (and thus
        // "not exceptional") behavior, you may like the signature
        // bool Decrypt(byte[] messageKey, byte[] cipherText, out byte[] message)
        // better.
```

```
            if (masterKey == null)
                throw new ArgumentNullException(nameof(masterKey));
            if (masterKey.Length < 16)
                throw new ArgumentOutOfRangeException(
                    nameof(masterKey),
                    "Master Key must be at least 128 bits (16 bytes)");
            if (cipherText == null)
                throw new ArgumentNullException(nameof(cipherText));

            // The format of this message is assumed to be public, so there's no harm in
            // saying ahead of time that the message makes no sense.
            if (cipherText.Length < 2)
            {
                throw new CryptographicException();
            }

            // Use the message algorithm headers to determine what cipher algorithm and
            // MAC algorithm are going to be used. Since the same Key Derivation
            // Functions (KDFs) are being used in Decrypt as Encrypt, the keys are also
            // the same.
            AeCipher aeCipher = (AeCipher)cipherText[0];
            AeMac aeMac = (AeMac)cipherText[1];

            using (SymmetricAlgorithm cipher = GetCipher(aeCipher, masterKey))
            using (HMAC tagGenerator = GetMac(aeMac, masterKey))
            {
                int blockSizeInBytes = cipher.BlockSize / 8;
                int tagSizeInBytes = tagGenerator.HashSize / 8;
                int headerSizeInBytes = 2;
                int tagOffset = headerSizeInBytes;
                int ivOffset = tagOffset + tagSizeInBytes;
                int cipherTextOffset = ivOffset + blockSizeInBytes;
                int cipherTextLength = cipherText.Length - cipherTextOffset;
                int minLen = cipherTextOffset + blockSizeInBytes;

                // Again, the minimum length is still assumed to be public knowledge,
                // nothing has leaked out yet. The minimum length couldn't just be calculated
                // without reading the header.
                if (cipherText.Length < minLen)
                {
                    throw new CryptographicException();
                }

                // It's very important that the MAC be calculated and verified before
                // proceeding to decrypt the ciphertext, as this prevents any sort of
                // information leaking out to an attacker.
                //
                // Don't include the tag in the calculation, though.

                // First, everything before the tag (the cipher and MAC algorithm ids)
                tagGenerator.AppendData(cipherText, 0, tagOffset);

                // Skip the data before the tag and the tag, then read everything that
                // remains.
                tagGenerator.AppendData(
                    cipherText,
                    tagOffset + tagSizeInBytes,
                    cipherText.Length - tagSizeInBytes - tagOffset);

                byte[] generatedTag = tagGenerator.GetHashAndReset();

                // The time it took to get to this point has so far been a function only
                // of the length of the data, or of non-encrypted values (e.g. it could
                // take longer to prepare the *key* for the HMACSHA384 MAC than the
                // HMACSHA256 MAC, but the algorithm choice wasn't a secret).
                //
                // If the verification of the authentication tag aborts as soon as a
                // difference is found in the byte arrays then your program may be
                // acting as a timing oracle which helps an attacker to brute-force the
```

```
        // right answer for the MAC. So, it's very important that every possible
        // "no" (2^256-1 of them for HMACSHA256) be evaluated in as close to the
        // same amount of time as possible. For this, we call CryptographicEquals
        if (!CryptographicEquals(
            generatedTag,
            0,
            cipherText,
            tagOffset,
            tagSizeInBytes))
        {
            // Assuming every tampered message (of the same length) took the same
            // amount of time to process, we can now safely say
            // "this data makes no sense" without giving anything away.
            throw new CryptographicException();
        }

        // Restore the IV into the symmetricAlgorithm instance.
        byte[] iv = new byte[blockSizeInBytes];
        Buffer.BlockCopy(cipherText, ivOffset, iv, 0, iv.Length);
        cipher.IV = iv;

        using (ICryptoTransform decryptor = cipher.CreateDecryptor())
        {
            return Transform(
                decryptor,
                cipherText,
                cipherTextOffset,
                cipherTextLength);
        }
    }
}

private static byte[] Transform(
    ICryptoTransform transform,
    byte[] input,
    int inputOffset,
    int inputLength)
{
    // Many of the implementations of ICryptoTransform report true for
    // CanTransformMultipleBlocks, and when the entire message is available in
    // one shot this saves on the allocation of the CryptoStream and the
    // intermediate structures it needs to properly chunk the message into blocks
    // (since the underlying stream won't always return the number of bytes
    // needed).
    if (transform.CanTransformMultipleBlocks)
    {
        return transform.TransformFinalBlock(input, inputOffset, inputLength);
    }

    // If our transform couldn't do multiple blocks at once, let CryptoStream
    // handle the chunking.
    using (MemoryStream messageStream = new MemoryStream())
    using (CryptoStream cryptoStream =
        new CryptoStream(messageStream, transform, CryptoStreamMode.Write))
    {
        cryptoStream.Write(input, inputOffset, inputLength);
        cryptoStream.FlushFinalBlock();
        return messageStream.ToArray();
    }
}

/// <summary>
/// Open a properly configured <see cref="SymmetricAlgorithm"/> conforming to the
/// scheme identified by <paramref name="aeCipher"/>.
/// </summary>
/// <param name="aeCipher">The cipher mode to open.</param>
/// <param name="masterKey">The master key from which other keys derive.</param>
/// <returns>
/// A SymmetricAlgorithm object with the right key, cipher mode, and padding
```

```
/// A SymmetricAlgorithm object with the right key, cipher mode, and padding
/// mode; or <c>null</c> on unknown algorithms.
/// </returns>
private static SymmetricAlgorithm GetCipher(AeCipher aeCipher, byte[] masterKey)
{
    SymmetricAlgorithm symmetricAlgorithm;

    switch (aeCipher)
    {
        case AeCipher.Aes256CbcPkcs7:
            symmetricAlgorithm = Aes.Create();
            // While 256-bit, CBC, and PKCS7 are all the default values for these
            // properties, being explicit helps comprehension more than it hurts
            // performance.
            symmetricAlgorithm.KeySize = 256;
            symmetricAlgorithm.Mode = CipherMode.CBC;
            symmetricAlgorithm.Padding = PaddingMode.PKCS7;
            break;
        default:
            // An algorithm we don't understand
            throw new CryptographicException();
    }

    // Instead of using the master key directly, derive a key for our chosen
    // HMAC algorithm based upon the master key.
    //
    // Since none of the symmetric encryption algorithms currently in .NET
    // support key sizes greater than 256-bit, we can use HMACSHA256 with
    // NIST SP 800-108 5.1 (Counter Mode KDF) to derive a value that is
    // no smaller than the key size, then Array.Resize to trim it down as
    // needed.

    using (HMAC hmac = new HMACSHA256(masterKey))
    {
        // i=1, Label=ASCII(cipher)
        byte[] cipherKey = hmac.ComputeHash(
            new byte[] { 1, 99, 105, 112, 104, 101, 114 });

        // Resize the array to the desired keysize. KeySize is in bits,
        // and Array.Resize wants the length in bytes.
        Array.Resize(ref cipherKey, symmetricAlgorithm.KeySize / 8);

        symmetricAlgorithm.Key = cipherKey;
    }

    return symmetricAlgorithm;
}

/// <summary>
/// Open a properly configured <see cref="HMAC"/> conforming to the scheme
/// identified by <paramref name="aeMac"/>.
/// </summary>
/// <param name="aeMac">The message authentication mode to open.</param>
/// <param name="masterKey">The master key from which other keys derive.</param>
/// <returns>
/// An HMAC object with the proper key, or <c>null</c> on unknown algorithms.
/// </returns>
private static HMAC GetMac(AeMac aeMac, byte[] masterKey)
{
    HMAC hmac;

    switch (aeMac)
    {
        case AeMac.HMACSHA256:
            hmac = new HMACSHA256();
            break;
        case AeMac.HMACSHA384:
            hmac = new HMACSHA384();
            break;
        default:
```

```
            default:
                // An algorithm we don't understand
                throw new CryptographicException();
        }

        // Instead of using the master key directly, derive a key for our chosen
        // HMAC algorithm based upon the master key.
        // Since the output size of the HMAC is the same as the ideal key size for
        // the HMAC, we can use the master key over a fixed input once to perform
        // NIST SP 800-108 5.1 (Counter Mode KDF):
        hmac.Key = masterKey;

        // i=1, Context=ASCII(MAC)
        byte[] newKey = hmac.ComputeHash(new byte[] { 1, 77, 65, 67 });

        hmac.Key = newKey;
        return hmac;
    }

    // A simple helper method to ensure that the offset (writePos) always moves
    // forward with new data.
    private static void Append(byte[] newData, byte[] combinedData, ref int writePos)
    {
        Buffer.BlockCopy(newData, 0, combinedData, writePos, newData.Length);
        writePos += newData.Length;
    }

    /// <summary>
    /// Compare the contents of two arrays in an amount of time which is only
    /// dependent on <paramref name="length"/>.
    /// </summary>
    /// <param name="a">An array to compare to <paramref name="b"/>.</param>
    /// <param name="aOffset">
    /// The starting position within <paramref name="a"/> for comparison.
    /// </param>
    /// <param name="b">An array to compare to <paramref name="a"/>.</param>
    /// <param name="bOffset">
    /// The starting position within <paramref name="b"/> for comparison.
    /// </param>
    /// <param name="length">
    /// The number of bytes to compare between <paramref name="a"/> and
    /// <paramref name="b"/>.</param>
    /// <returns>
    /// <c>true</c> if both <paramref name="a"/> and <paramref name="b"/> have
    /// sufficient length for the comparison and all of the applicable values are the
    /// same in both arrays; <c>false</c> otherwise.
    /// </returns>
    /// <remarks>
    /// An "insufficient data" <c>false</c> response can happen early, but otherwise
    /// a <c>true</c> or <c>false</c> response take the same amount of time.
    /// </remarks>
    [MethodImpl(MethodImplOptions.NoInlining | MethodImplOptions.NoOptimization)]
    private static bool CryptographicEquals(
        byte[] a,
        int aOffset,
        byte[] b,
        int bOffset,
        int length)
    {
        Debug.Assert(a != null);
        Debug.Assert(b != null);
        Debug.Assert(length >= 0);

        int result = 0;

        if (a.Length - aOffset < length || b.Length - bOffset < length)
        {
            return false;
        }
```

```
            unchecked
            {
                for (int i = 0; i < length; i++)
                {
                    // Bitwise-OR of subtraction has been found to have the most
                    // stable execution time.
                    //
                    // This cannot overflow because bytes are 1 byte in length, and
                    // result is 4 bytes.
                    // The OR propagates all set bytes, so the differences are only
                    // present in the lowest byte.
                    result = result | (a[i + aOffset] - b[i + bOffset]);
                }
            }

            return result == 0;
        }
    }
}
```