

Tema 13. Programación segura para aplicaciones móviles

Formación específica, cursos verano 2008
ETS de Informática Aplicada
Universidad Politécnica de Valencia

Índice

- ◆ Seguridad en las implementaciones
- ◆ Salvar credenciales y secretos de forma segura
- ◆ Cifrado de información
- ◆ Comunicaciones seguras
- ◆ Uso de firmas en las aplicaciones

Vulnerabilidades y riesgos

- ◆ Entradas de usuario no validadas
- ◆ Usos de esquemas de autenticación y autorización débiles en el acceso a recursos y servicios web
- ◆ Uso de conexiones de red no seguras
- ◆ Codificación en texto plano de nombres de usuario, contraseñas u otros secretos
- ◆ Intercambio de información sensible no cifrada
- ◆ Almacenamiento de información sensible en el dispositivo durante más tiempo que el necesario

Información sensible en mi código ¿Qué problema hay?

- ◆ Las credenciales que aparezcan en el programa se pueden obtener con facilidad desde los ejecutables
 - Los ejecutables contienen MSIL (código intermedio)
 - Por cuestiones de espacio éste es preferible al código nativo
 - Cuando es requerido el código MSIL es transformado en nativo por el JIT
 - El uso de ofuscadores puede ayudar ...

Buenas y malas técnicas para guardar secretos

- ◆ Poner los secretos en el código pero luego ofuscar
 - ii Aunque utilicemos ofuscadores de código las constantes no se ofuscan !!
 - Demo: ildasm, dotfuscator, Roeder's reflector
 - ◆ Nota: <http://www.aisto.com/roeder/dotnet> (reflector)
- ◆ Ocultar el secreto en un bloque de código mayor
 - Mala praxis

Buenas y malas técnicas para guardar secretos

- ◆ Guardar el secreto en el registro del sistema
 - ¿Se cifra ese secreto?
 - Solución no segura, uso de Remote Registry Editor o aplicación equivalente para el dispositivo
 - Tal vez limitando el acceso al dispositivo a través de una contraseña robusta ...
- ◆ Guardar el secreto cifrado en el registro del sistema o en un fichero
 - Solución disuasoria para curiosos, pero ¿Y si alguien roba el fichero y lo intenta descifrar?
 - Mejor si la clave de cifrado se deduce a través de una contraseña que se solicita al usuario

Buenas y malas técnicas para guardar secretos

- ◆ Guardar el secreto en el registro del sistema
 - ¿Se cifra ese secreto?
 - Solución no segura, uso de Remote Registry Editor o aplicación equivalente para el dispositivo
 - Tal vez limitando el acceso al dispositivo a través de una contraseña robusta ...
- ◆ Guardar el secreto cifrado en el registro del sistema o en un fichero
 - Solución disuasoria para curiosos, pero ¿Y si alguien roba el fichero y lo intenta descifrar?
 - Mejor si la clave de cifrado se deduce a través de una contraseña que se solicita al usuario => ii Empezamos a hablar de seguridad !!

Buenas y malas técnicas para guardar secretos

- ◆ En la medida de lo posible solicitar al usuario cierta información (usuario y contraseña) que permita derivar las claves (o los hash) necesarias para descifrar la información sensible
 - ¿Con qué frecuencia se solicita al usuario su contraseña? => Evitar irritarlo
 - Si guardamos la contraseña ¿qué pasa si nos roban el dispositivo?
 - Borrar (reescribir) las variables que hayan contenido información sensible una vez que ésta no se está utilizando
- ◆ Moraleja: Una buena solución de seguridad requiere de la intervención del usuario

En resumen ...

- ◆ Si la aplicación es utilizada por varios usuarios ¿comparten todos la misma contraseña?
- ◆ ¿Cómo distribuir información cifrada a distintos dispositivos y permitir que cada usuario utilice su propia y única contraseña para descifrarla?
- ◆ ¿Y si los usuarios olvidan sus contraseñas? ¿Cómo cambiarlas?

La propuesta de Microsoft

- ◆ La gestión de la seguridad es compleja
- ◆ Microsoft propone utilizar bloques aplicativos definidos en el *Mobile Client Software Factory*
 - Descargable desde <http://www.microsoft.com/downloads/details.aspx?FamilyId=F9176708-9F57-4C0F-97FB-F9C65A9BBF22&displaylang=en>
 - ◆ Mobile Configuration Block: Lee ficheros de configuración de aplicaciones
 - ◆ Mobile Authentication Block: Proporciona métodos para validar usuarios (login y passwords)
 - ◆ Endpoint Catalog: API para exponer URLs y credenciales almacenadas en ficheros de configuración

Salvar credenciales y secretos de forma segura

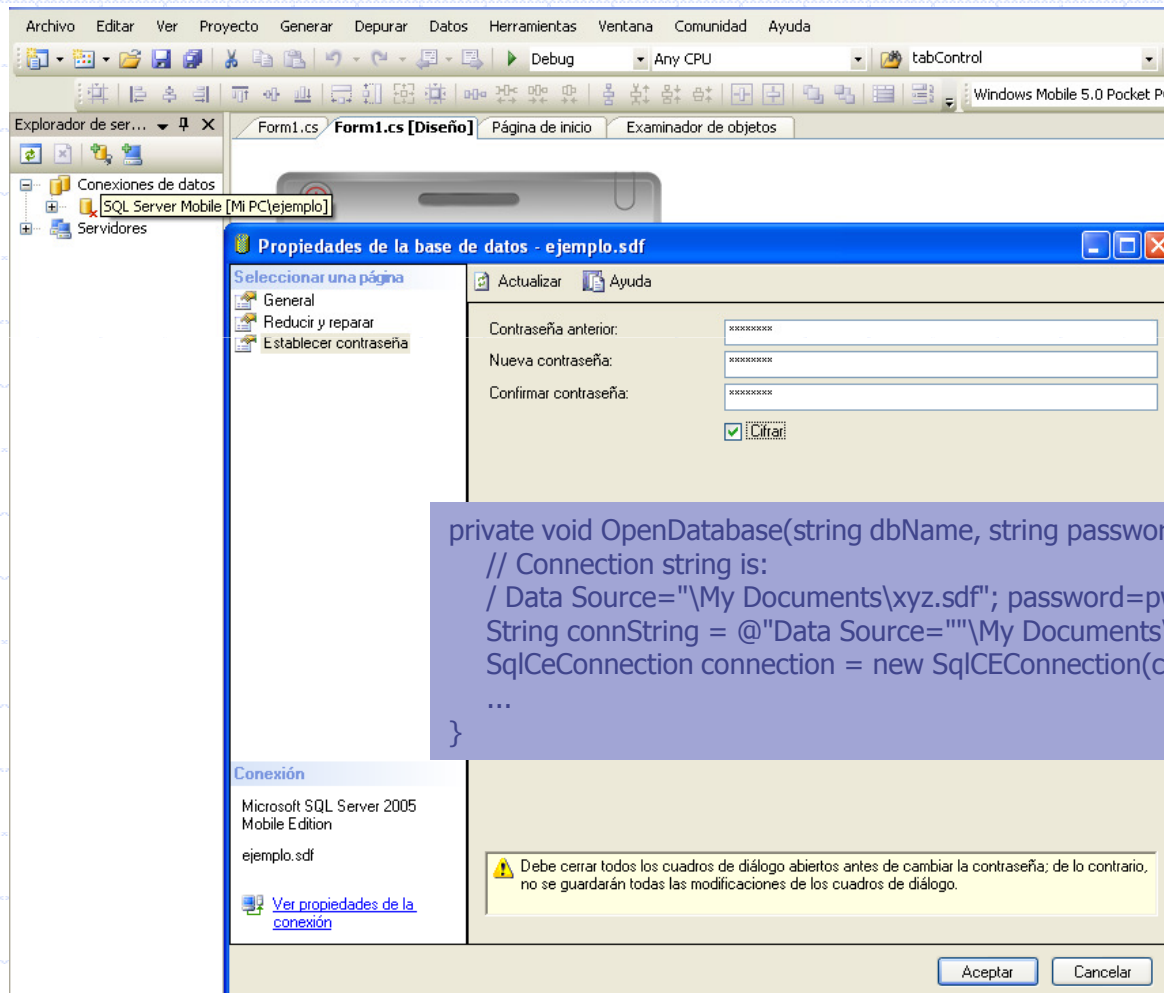
◆ Estudio de caso:

- Aplicación que almacena información en una BD SQL Server CE y comunica con un Web Service que autentica a los usuarios a través de un esquema basado en HTTP
- Vulnerabilidades y riesgos
 - ◆ BD
 - ◆ La URL del Web Service y las credenciales
 - ◆ Información transmitida

Mitigación de riesgos

- ◆ Cifrar el contenido de la base de datos
- ◆ Crear un fichero de configuración en la aplicación
 - Contenido: clave de la BD, información del usuario (login y contraseña) y la URL del servidor
 - Cifrar dicho contenido
- ◆ Si la aplicación tiene varios usuarios, utilizar el mismo fichero de configuración para todos
 - Necesidad de guardar de manera segura la clave para descifrar su contenido
 - Cada usuario tendrá su propio login y contraseña
 - ◆ Si se introducen OK => descifrar el contenido del fichero de configuración y la aplicación se hace funcional

Protección de datos en BD SQL CE



```
private void OpenDatabase(string dbName, string password) {
    // Connection string is:
    / Data Source="\\My Documents\xyz.sdf"; password=pwd
    String connString = @"Data Source=""\My Documents\" + dbName + """; Password=" + password;
    SqlCeConnection connection = new SqlCeConnection(connString);
    ...
}
```

- ☹ **Coste computacional importante => Consumo de batería**
- ☹ **Establecer 2 BDs**
 - **BD sensible**
 - **BD no sensible**

Definición del fichero de configuración

Identifica las clases que gestionan la información relativa al EndPoint y al SystemSettings

Ensamblado

Clase (tipo de datos)

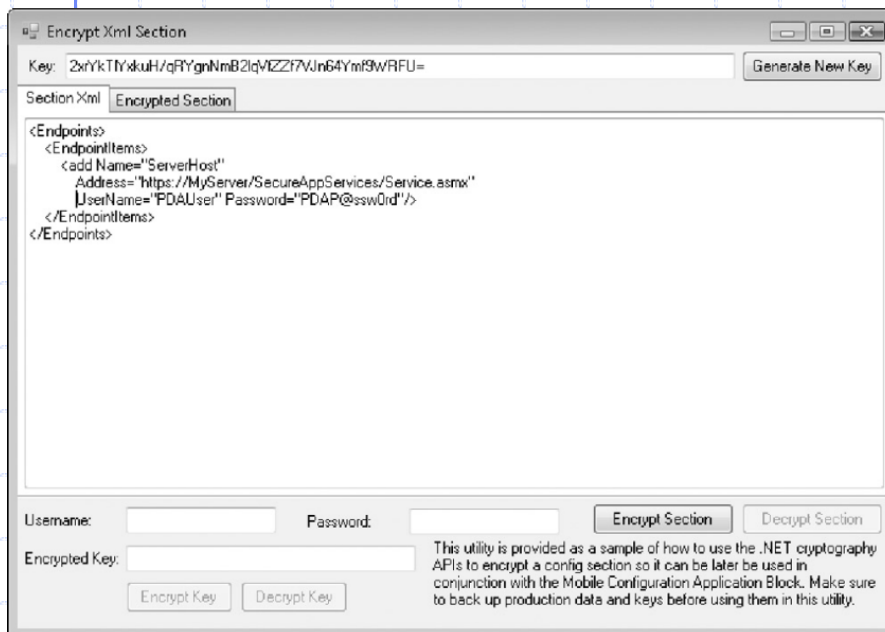
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="Endpoints" type=
      "Microsoft.Practices.Mobile.EndpointCatalog.Configuration.EndpointSection
,Microsoft.Practices.Mobile.EndpointCatalog" />
    <section name="SystemSettings"
      type="MobileDevelopersHandbook.SystemSettingsSection,
MobileDevelopersHandbook.ApplicationBlocksSecure"/>
  </configSections>

  <Endpoints>
    <EndpointItems>
      <add Name="ServerHost"
        Address="https://MyServer/SecureAppServices/Service.asmx"
        UserName="PDAUser" Password="PDAP@ssw0rd"/>
    </EndpointItems>
  </Endpoints>

  <SystemSettings>
    <SystemSettingsItems>
      <add Name="DatabasePassword" Value="MOBILEP@ssw0rd" />
    </SystemSettingsItems>
  </SystemSettings>
</configuration>
```

Cifrado del fichero de configuración

- ◆ Uso de la herramienta ConfigSectionEncrypt que viene con el MCSF
 - Asumimos que el MSCF ya está instalado
 - Inicio->Todos los programas->Microsoft patterns and practices->Mobile Client Software Factory->Tools



- Copiar el texto de cada sección y solicitar su cifrado
- Sustituir en el fichero original el resultado por el proporcionado por la herramienta

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="Endpoints" type=... />
    <section name="SystemSettings" type=.../>
  </configSections>

  <EncryptedSection name="Endpoints">AQAQNu3F... ..J2M+EN</EncryptedSection>
  <EncryptedSection name="SystemSettings">AQAQymI... ..8r7aR</EncryptedSection>
</configuration>
```

Detalle

- ◆ ConfigSectionEncrypt muestra una clave codificada en Base64 generada utilizando el algoritmo de cifrado simétrico de Rijndael

```
using System.Security.Cryptography;
...
public static byte[] GenerateKey()
{
    SymmetricAlgorithm algorithm = Rijndael.Create();
    return algorithm.Key;
}
```

(detalle de código de la herramienta)

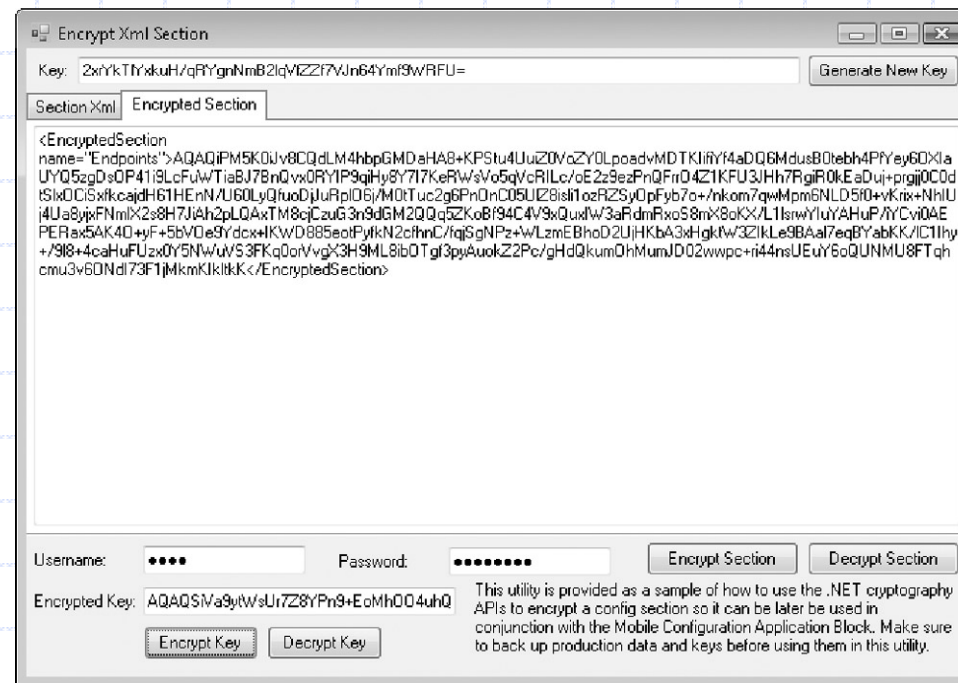
- ◆ Al ser simétrica la clave (clave maestra) que se utiliza para cifrar es la misma que se utiliza para descifrar

Problema

- ◆ ¿Cómo distribuir la clave maestra a distintos dispositivos sin revelarla a eventuales usuarios maliciosos?
- ◆ Solución: cifrar la clave maestra con una clave específica que será función de un usuario-contraseña concreto
 - Se posibilita el envío de la clave maestra por canales de comunicación inseguros

¿Cómo generar una clave específica?

- ◆ Utilizando de nuevo la utilidad ConfigSectionEncrypt tool



Detalle

- ◆ Distintas pulsaciones al botón Encrypt Key conllevan la generación de distintas claves específicas
 - El mismo texto se cifrará de forma distintas en distintas ocasiones
 - ◆ Esto es porque el algoritmo de Rijndael cifra los datos a partir de una clave y una semilla (IV)
 - ◆ Esta semilla también se incluirá en el mensaje cifrado

```
<?xml version="1.0" encoding="utf-8" ?>
<Users>
  <User Name="Bill"
    Token="AQAkk90fWdPpq+Umc4tTmwy9u61JD+nPEPu6H1ke"
    EncKey="AQAQbQR3QmZ1wUNfJI73g1lp5tCt9gGjAn... ..04n14thG+FvwL/Ng==" />
  <User Name="Mary"
    Token="AQAYAVjIVA94wzLv2rXRRYSES14zJ3vjR2OFHrd04"
    EncKey="AQAQbqT2rGTfiwW/iZTBh85myiPHuu26eA0xuG99As... ..VgMPeaFoQ==" />
  <User Name="Joe"
    Token="AQAsocb1MW6WiOV/bzj1yqehrYAKP05dWq808gF05"
    EncKey="AQAQXG+00D408eajDnmoJ/... ..LAMGZ0KS0YquZQ7RdI9N/MjA==" />
</Users>
```

- Sólo los usuarios que conozcan la contraseña podrán obtener la clave maestra y descifrar el mensaje de configuración

Creación de tokens

- ◆ Uso de la clase AuthenticationToken class del Password Authentication Application Block del MCSF
 - Crear un token a partir de usuario+contraseña

```
using Microsoft.Practices.Mobile.PasswordAuthentication;
...
private void buttonShowToken_Click(object sender, EventArgs e)
{
    using (RsaAesCryptographyProvider provider =
        new RsaAesCryptographyProvider("MobileDevelopersHandbook"))
    {
        PasswordIdentity identity = new PasswordIdentity(
            textBoxUsername.Text, textBoxPassword.Text, provider);
        AuthenticationToken token = new AuthenticationToken(identity);
        textBoxToken.Text = token.TokenData;
    }
}
```

Autenticación del usuario

```
private void menuItemSubmit_Click(object sender, EventArgs e)
{
    for(int rowidx = 0; rowidx < users.Tables[0].Rows.Count; rowidx++)
    {
        DataRow userRow = users.Tables[0].Rows[rowidx];
        if ((string)userRow["Name"] == textBoxUsername.Text)
        {
            Cursor.Current = Cursors.WaitCursor;

            try
            {
                PasswordIdentity identity =
                    AuthenticateUser((string)userRow["Token"]);
                if (identity != null && identity.IsAuthenticated)
                {
                    // Success!! Decrypt the config file.
                    DecryptSettings(
                        identity, (string)userRow["encKey"]);

                    //Close this modal dialog box.
                    this.DialogResult = DialogResult.OK;
                }
            }
            finally
            {
                Cursor.Current = Cursors.Default;
            }
        }
    }

    // Show the login failed message.
    labelIncorrect.Visible = true;
}
```

```
private PasswordIdentity AuthenticateUser(string userToken)
{
    using (RsaAesCryptographyProvider provider =
        new RsaAesCryptographyProvider("MobileDevelopersHandbook"))
    {
        // Create AuthenticationToken using existing token.
        AuthenticationToken token =
            new AuthenticationToken(userToken);
        PasswordIdentity identity = token.Authenticate(
            textBoxUsername.Text, textBoxPassword.Text, provider);
        // return result - caller checks Authenticated property to
        // see authentication result
        return identity;
    }
}
```

Descrifar el fichero

```
private void DecryptSettings(
    PasswordIdentity identity, string userEncryptedConfigurationKey)
{
    // Obtain the user's key.
    byte[] userKeyBytes = identity.CryptoKey;

    // Create an instance of the CryptographyBlock class.
    SymmetricAlgorithm symmetric = Rijndael.Create();
    CryptographyBlock block =
        new CryptographyBlock(symmetric, userKeyBytes);

    // Get the user's encrypted configuration key.
    byte[] configKeyBytes =
        Convert.FromBase64String(userEncryptedConfigurationKey);

    // Decrypt the key
    byte[] configurationKey = block.Decrypt(configKeyBytes);

    // Create a configuration provider to decrypt the config file.
    RijndaelConfigurationProvider configProvider =
        new RijndaelConfigurationProvider(configurationKey);

    // Assign the new instance of the RijndaelConfigurationProvider
    // class to the ConfigurationManager.
    ConfigurationManager.ProtectedConfigurationProvider
        = configProvider;

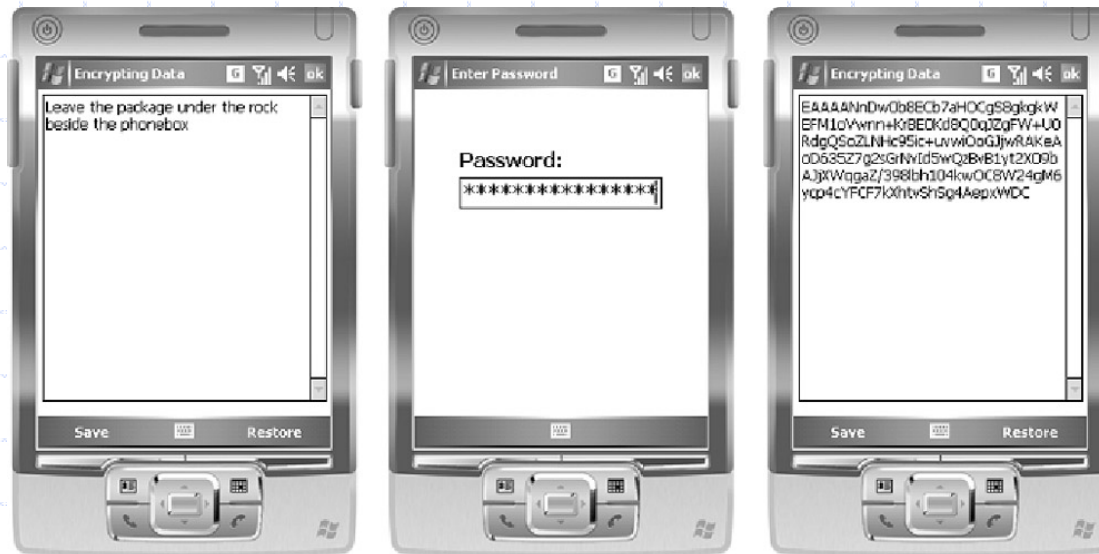
    // Finally, use the GetSection method of the ConfigurationManager
    // to retrieve the section you want.
    String sectionName = "SystemSettings";
    SystemSettingsSection configSection =
        ConfigurationManager.GetSection(sectionName)
        as SystemSettingsSection;

    // Store the decrypted data in the settings collection.
    foreach (SystemSettingsItemElement item in
        configSection.SystemSettingsItems)
    {
        configSettings.Add(item.Name, item.Value);
    }
}
```

EJEMPLO: CIFRADO DE DATOS USANDO UN ALGORITMO DE CIFRADO SIMETRICO AES

Ejemplo

- ◆ El código del ejemplo está en el directorio Encryption de los ejemplos suministrados



Ejemplo

- ◆ Derivar la clave de cifrado del string proporcionado por el usuario

```
using Microsoft.Practices.Mobile.PasswordAuthentication;
...
private byte[] DeriveKeyFromPassword(string password)
{
    byte[] key;
    using (RsaAesCryptographyProvider provider =
        new RsaAesCryptographyProvider("DevelopersHandbook"))
    {
        CryptNativeHelper crypto = new CryptNativeHelper(provider);
        key = crypto.GetPasswordDerivedKey(password);
    }

    return key;
}
```

Ejemplo

◆ Cifrar los datos

```
private string EncryptData(byte[] key, string plainText)
{
    // Get the bytes to encrypt.
    byte[] plaintextByte =
        System.Text.Encoding.Unicode.GetBytes(plainText);

    // Create a Rijndael instance.
    RijndaelManaged rijndael = new RijndaelManaged();

    // Set encryption mode.
    rijndael.Mode = CipherMode.ECB;
    rijndael.Padding = PaddingMode.PKCS7;

    // Create a random initialization vector.
    rijndael.GenerateIV();
    byte[] iv = rijndael.IV;

    string encodedText = "";

    // Define memory stream that will be used to hold encrypted data.
    MemoryStream memStrm = new MemoryStream();

    // Write the IV length and the IV.

    memStrm.Write(BitConverter.GetBytes(iv.Length), 0, 4);
    memStrm.Write(iv, 0, iv.Length);

    // Create a symmetric encryptor.
    using (ICryptoTransform encryptor =
        rijndael.CreateEncryptor(key, iv))
    {
        // Create a CryptoStream to write to the output file.
        CryptoStream cryptStrm = new CryptoStream(
            memStrm, encryptor, CryptoStreamMode.Write);

        // Write the content to be encrypted.
        cryptStrm.Write(plaintextByte, 0, plaintextByte.Length);
        cryptStrm.FlushFinalBlock();

        // Convert encrypted data from memory stream into byte array.
        byte[] cipherTextBytes = memStrm.ToArray();

        // Close the streams.
        memStrm.Close();
        cryptStrm.Close();

        // Convert encrypted byte array into a base64-encoded string.
        encodedText = Convert.ToBase64String(cipherTextBytes);
    }
    return encodedText;
}
```

Ejemplo

◆ Describir el contenido resultante

```
using System.IO;
using System.Security.Cryptography;
...
private string DecryptData(byte[] key, string encryptedData)
{
    string retStr = "";

    // Create a symmetric decryptor.
    RijndaelManaged rijndael = new RijndaelManaged();
    rijndael.Mode = CipherMode.ECB;
    rijndael.Padding = PaddingMode.PKCS7;

    // Convert the ciphertext into a byte array.
    byte[] cipherTextBytes = Convert.FromBase64String(encryptedData);
    // Define memory stream to use to read encrypted data.
    MemoryStream inStream = new MemoryStream(cipherTextBytes);

    // Read the IV length from the buffer.
    int ivLength = BitConverter.ToInt32(cipherTextBytes, 0);
    // Reposition to after 'length' bytes in stream.
    inStream.Position = 4;

    // Read the IV from the input stream.
    byte[] iv = new byte[ivLength];
    inStream.Read(iv, 0, ivLength);

    using (ICryptoTransform decryptor =
        rijndael.CreateDecryptor(key, iv))
    {
        // Create a CryptStream to read from the file.
        CryptoStream cryptStrm = new CryptoStream(
            inStream, decryptor, CryptoStreamMode.Read);

        // Create another MemoryStream for the output.
        MemoryStream memStrm = new MemoryStream();
        byte[] buffer = new byte[2048];
        int totalbytes = 0;
        do
        {
            int bytesRead = cryptStrm.Read(buffer, 0, buffer.Length);
            if (bytesRead == 0)
                break;
            memStrm.Write(buffer, 0, bytesRead);
            totalbytes += bytesRead;
        } while (true);

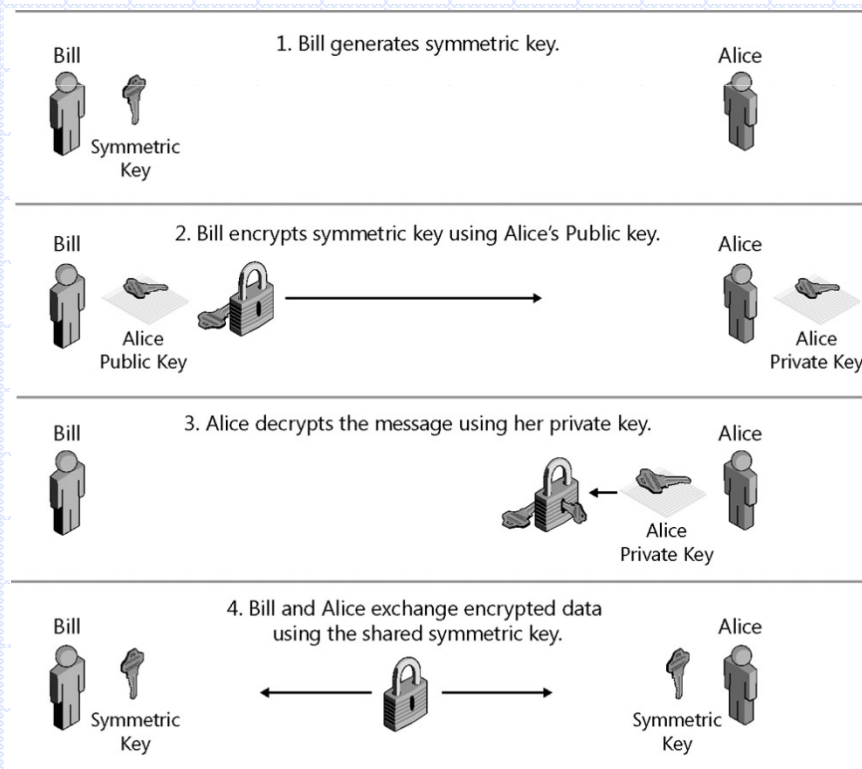
        // Write the content to be encrypted.
        memStrm.Flush();
        memStrm.Seek(0, SeekOrigin.Begin);

        // Get the string from the bytes you read.
        retStr = System.Text.Encoding.Unicode.GetString(
            memStrm.GetBuffer(), 0, totalbytes);
        cryptStrm.Close();
    }
    return retStr;
}
```

EJEMPLO: CIFRADO DE DATOS USANDO UN ALGORITMO DE CIFRADO ASIMÉTRICO RSA

Concepto de cifrado asimétrico

- ◆ Cifrado simétrico es más eficaz, pero ¿cómo intercambiar la clave?



Generación de claves

```
using System.Security.Cryptography;
...
private string GetRSAKeyPair(string containerName)
{
    //Create a CryptoServiceProvider Parameter object.
    CspParameters cspParams = new CspParameters();
    // Set the key container name that has the RSA key pair.
    cspParams.KeyContainerName = containerName;
    //Set the CSP Provider Type PROV_RSA_FULL.
    cspParams.ProviderType = 1;
    //Set the CSP Provider Name
    cspParams.ProviderName =
        "Microsoft Enhanced Cryptographic Provider v1.0";

    //Create a new RSA provider, pass CspParameters to the constructor.
    //If specified key container doesn't exist, creates a new key pair
    rsaProvider = new RSACryptoServiceProvider(cspParams);

    //Indicate that you would like the new key pair to be persisted in
    // the key container specified.
    rsaProvider.PersistKeyInCsp = true;

    //Return the PUBLIC key info.
    return rsaProvider.ToXmlString(false);
}
```

Almacenar la clave

```
<?xml version="1.0" encoding="utf-8" ?>
<RSAKeyValue>
  <Modulus>ziN2zzR30Xnn7w+... ..o2aCq+ObHeZF41f18=</Modulus>
  <Exponent>AQAB</Exponent>
</RSAKeyValue>
```

```
public void SavePassword(string cleartext)
{
    RSACryptoServiceProvider rsaProvider =
        new RSACryptoServiceProvider();
    //Create a new instance of RSAParameters.
    RSAParameters RSAKeyInfo = new RSAParameters();

    RSAKeyInfo = ReadPublicKeyXML(RSAKeyInfo);

    //Import key parameters into RSA.
    rsaProvider.ImportParameters(RSAKeyInfo);

    //Encrypt the supplied data.
    byte[] cipherText = rsaProvider.Encrypt(
        new UnicodeEncoding().GetBytes(cleartext), false);

    //Save the ciphertext.
    using (System.IO.StreamWriter sw =
        new System.IO.StreamWriter(GetApplicationDirectory() +
            @"\UserKeyCipher.txt", false))
    {
        sw.Write(Convert.ToBase64String(cipherText));
        sw.Flush();
    }
}
```

Leer la clave pública

```
private RSAParameters ReadPublicKeyXML(RSAParameters RSAKeyInfo)
{
    // Read the public key from the XML file by using a reader.
    using (Stream stm = File.OpenRead(GetApplicationDirectory() +
        @"\PublicKey.xml"))
    {
        XmlTextReader reader = new XmlTextReader(stm);
        // Search the XML for the required element.
        string ev = reader.NameTable.Add("RSAKeyValue");
        while (reader.Read())
        {
            if (reader.LocalName == ev)
            {
                // Process it!
                int eventDepth = reader.Depth;
                reader.Read();
                while (reader.Depth > eventDepth)
                {
                    if (reader.MoveToContent() == XmlNodeType.Element)
                    {
                        switch (reader.Name)
                        {
                            case "Modulus":
                                RSAKeyInfo.Modulus =
                                    Convert.FromBase64String(reader.ReadString());
                                break;
                            case "Exponent":
                                RSAKeyInfo.Exponent =
                                    Convert.FromBase64String(reader.ReadString());
                                break;
                        }
                    }
                    reader.Read();
                }
            }
        }
    }
    return RSAKeyInfo;
}
```

Descifrar utilizando la clave RSA privada

```
private void textBoxFilePath_TextChanged(object sender, EventArgs e)
{
    if (System.IO.File.Exists(textBoxFilePath.Text))
    {
        string cipherBase64 =
            System.IO.File.ReadAllText(textBoxFilePath.Text);
        byte[] cipherText = Convert.FromBase64String(cipherBase64);

        // Decrypt the file using the PRIVATE key.
        byte[] decipheredText =
            rsaprovider.Decrypt(cipherText, false);

        //Display the deciphered message.
        labelResult.Text =
            new UnicodeEncoding().GetString(decipheredText);
    }
}
```

Comunicaciones seguras

◆ SSL (Secure Sockets Layer)

- Gestiona el intercambio de claves y el cifrado con clave simétrica de las comunicaciones
- Uso de certificados X.509 para autenticar a los extremos de una comunicación
- Necesidad de utilizar certificados expedidos por autoridades de certificación autorizadas

Certificados raíz en dispositivos windows mobile

- Class 2 Public Primary Certification Authority (VeriSign, Inc.)
- Class 3 Public Primary Certification Authority (VeriSign, Inc.)
- Entrust.net Certification Authority (2048)
- Entrust.net Secure Server Certification Authority
- Equifax Secure Certification Authority
- GlobalSign Root CA
- GTE CyberTrust Global Root
- GTE CyberTrust Root
- Secure Server Certification Authority (RSA)
- Thawte Premium Server CA
- Thawte Server CA

Almacenes de certificados

- ◆ Priviledge
- ◆ Normal
 - Utilizados por el cargador de seguridad para controlar la ejecución de aplicaciones
 - En el segundo caso la aplicación se considerará unprivileged
- ◆ SPC : Gobierna la instalación de ficheros cab.
 - El instalador de cab intenta asociar la firma del cab con uno de los certificados del almacén si no coincide no se instala

Validación de un certificado

◆ Esquema general a nivel de programa

```
using System.Windows.Forms;
using System.Net;
using System.Security.Cryptography.X509Certificates;

public class Form1: Form
{
    ...

    private void SetPolicyAndCallWebService()
    {
        System.Net.ServicePointManager.CertificatePolicy =
            new TrustAllCertificatePolicy();
        // Call method to call the SSL-secured Web service.
        CallSecureWebService();
    }
}

public class TrustAllCertificatePolicy : System.Net.ICertificatePolicy
{
    public TrustAllCertificatePolicy()
    { }

    public bool CheckValidationResult(ServicePoint servicepoint,
        X509Certificate cert, WebRequest req, int problem)
    {
        return true;
    }
}
```

◆ Gestión de certificados con Exchange Server

Política de seguridad en WM

- ◆ La política de seguridad determina el nivel de confianza que el sistema operativo deposita en las aplicaciones
 - Aplicaciones sin firmar, sin privilegios y privilegiadas

Privileged, Unprivileged, and Unsigned Applications

The Windows Mobile security model recognizes three kinds of applications:

- **Privileged** An application that has been signed using a certificate that has a corresponding certificate in the Privileged Execution Trust Authorities certificate store
- **Unprivileged** An application that has a corresponding certificate in the Unprivileged Execution Trust Authorities certificate store
- **Unsigned** An application that is not signed

- Modos de ejecución normal y de confianza

The Windows Mobile security model also recognizes two different application execution modes:

- **Trusted** An application that is virtually unlimited in what it is allowed to do. It can write to any registry key and call any Windows API.
- **Normal** An application that is not allowed to call certain restricted APIs or modify restricted registry keys. Restricted items are typically APIs and registry entries used to control security functionality and other essential functions. For a list of restricted APIs and registry keys, see the topic titled "Trusted APIs" in the Windows Mobile software development kit (SDK) documentation.

Seguridad de 1 y 2 niveles

- ◆ Ante del WM5 no hay gestión de seguridad
- ◆ Pocket-PCs con WM5 o superior integran seguridad a 1 nivel (One-tier security)
 - Ejecución (Trusted == Normal) => Si una aplicación se ejecuta lo hace en modo Trusted
- ◆ Smartphones con WM5 o superior integran seguridad a 2 niveles (Two-tier security)
 - Diferencia entre ejecución normal y de confianza

Políticas y configuraciones de seguridad

Security Policy	Description
4102	Can unsigned apps run? 0-No, unsigned apps cannot run. 1-Yes, unsigned apps can run.
4122	Prompt user to run unsigned apps. 0-Yes, prompt user. 1-No, do not prompt user. Note that if policy 4102 = 0, the setting of 4122 is irrelevant because unsigned apps cannot run.
4123	Is the device two-tier or one-tier? 0-Two-tier 1-One-tier
4097	User rights when making Remote API (RAPI; an API used by desktop programs calling into a Microsoft ActiveSync-connected device) calls. 0-RAPI disabled. 1-RAPI allowed with full access rights. 2-Restricted so that the desktop application has the same rights as the device user. (Note: By default, many Windows Mobile 5.0-powered devices have RAPI disabled, so applications that used RAPI successfully with older versions of Pocket PC may not work unless you change this policy.)

Configuration	4102 (Unsigned apps can run?)	4122 (Prompt user for unsigned apps?)	4123 (One-tier or two-tier?)	4097 (RAPI allowed?)
Locked	0-No	1-No	Either	0-Disabled
Third-Party-Signed	0-No	1-No	Either	0-Disabled
Two-Tier-Prompt	1-Yes	0-Yes	0-Two-Tier	2-Restricted
One-Tier-Prompt	1-Yes	0-Yes	1-One-Tier	2-Restricted
Security-Off	1-Yes	1-No	1-One-Tier	1-Allowed

Ejemplo

- ◆ Evitar que las aplicaciones sin firmar se ejecuten
 - Crear fichero XML en el que se defina la política de seguridad

```
<wap-provisioningdoc>
  <characteristic type="SecurityPolicy">
    <parm name="4102" value="0" />
  </characteristic>
</wap-provisioningdoc>
```

- Guardar el fichero como `_setup.xml`
- Ejecutar el program MakeCab desde la línea de comando
(disponible en `¿?:\Program Files/Microsoft Visual Studio 8/SmartDevices/SDK/SDKTools`)
 - ◆ `Makecab _setup.xml DisableUnsignedApps.cpf`
(cpf es el acrónimo de cab provisioning format file, archivos específicamente definidos para modificar la configuración del dispositivo)

Consultar y modificar la configuración de seguridad

- ◆ Uso de Microsoft Device Security Manager Powertoy for Windows Mobile
 - <http://www.microsoft.com/downloads/details.aspx?FamilyID=7e92628c-d587-47e0-908b-09fee6ea517a&displaylang=en>
 - Laboratorio virtual de 90' disponible en <http://msdn.microsoft.com/en-us/virtuallabs/aa740452.aspx>

