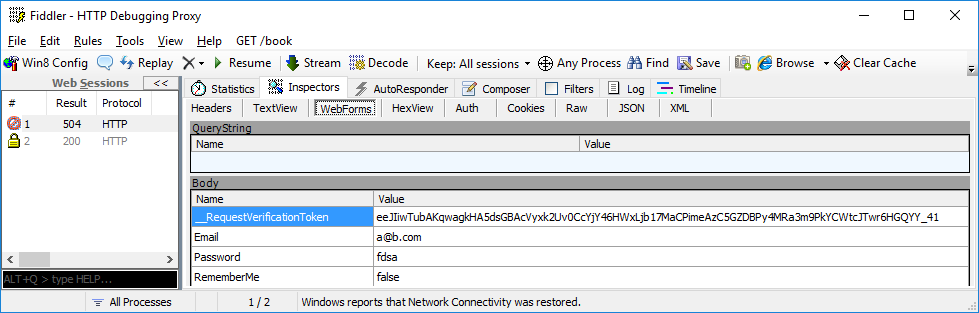
**ASP.NET MVC中使用JS实现不对称加密密码传输**

摘要： ASP.NET MVC中登录页面中点击登录后，用户名、密码将被明文传输到Controller中，使用Fiddler等工具可以轻松截获并获取密码， 这是不安全的。 使用对称加密，如AES，密钥将被暴露前端代码，也是不安全的。使用不对称加密能够较好解决这个问题。本文以RSA不对称加密的形式，在JS端通过公钥对密码进行加密，将密文传输到后端后通过密钥进行解密。

关键字： 不对称加密,对称加密,RSA 算法,AES, 密钥,公钥

**0 背景**

登录是最常见的需求之一，在这个环节，安全问题不可避免，明文传输很容易被截获并暴露密码原文。如下图使用Fiddle中出现的情况。



为了避免这种情况，通常办法有1 使用HTTPS形式解决； 2 使用公钥和不对称加密对密文进行加密; 3使用对称加密，比如AES。

这3种方案中，方案1是终极方案，但是需要克服证书获取和配置的问题， 本方案不是本文讨论重点，请有兴趣的自行查阅<https://letsencrypt.org/>。方案3， 以AES加密为例，必须把加密密钥存放在前端。 而前端对用户来说是开源的，很多开发者尝试把密钥藏的路径很深，但无疑这还是自欺欺人的。

方案2中，在JS端进行密码的RSA加密是有必要的，因为密码需要在用户点击“登录”按钮后被提交到服务器，这个过程被截获是很容易的。同时，防范CSRF类型攻击的特性也必须保留。这就要求：必须使用AJAX在JS端对密码加密，并向后台的Account Controller中的Login Action发起Post请求。而不能使用传统的Form Submit的方案。

AJAX post请求中，需要注意问题: 由于需要防范CSRF攻击的同时保障密文传输安全。需要同时顾及如下问题

问题1： 如何通过AJAX向Controller发起ajax请求？

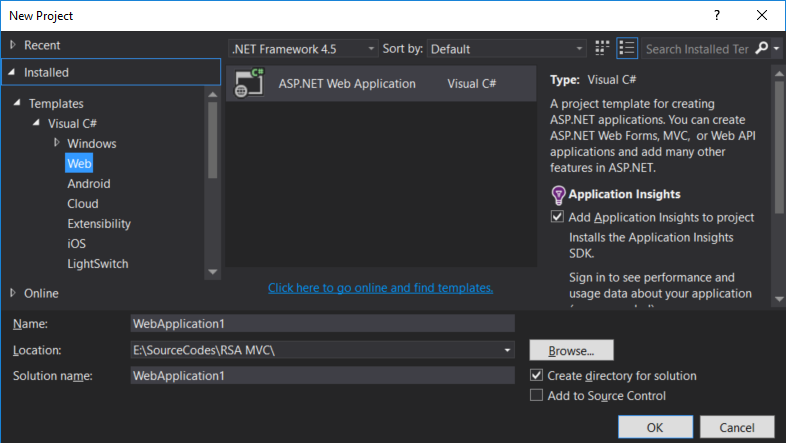
问题2：如何在ajax请求中加入AntiForgeryToken？

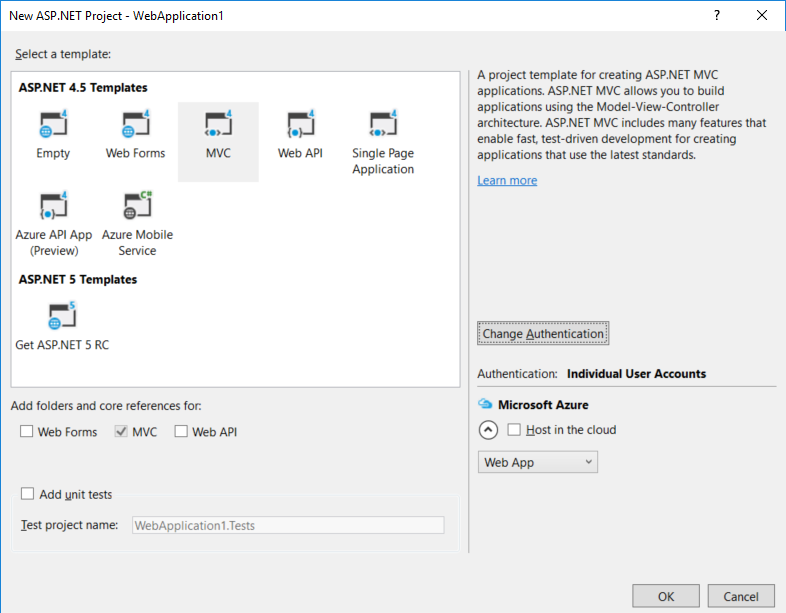
问题3： AJAX请求前，如何对密码进行RSA加密？

问题4： RSA的key format 有两种， pem格式和C#所支持的XML格式，通常JS支持pem, C#支持xml， 如何转换？

带着问题，进入操作步骤；

**1 操作步骤**

* 1. 新建Web Application
  2. 选择MVC， Authentication中选择”Individual User Accounts”



* 1. **Views** -> Account->Login.cshtml中代码修改如下

**@using WebApplicationRSA.Models**

**@model LoginViewModel**

**@{**

**ViewBag.Title = "Log in";**

**}**

**<h2>@ViewBag.Title.</h2>**

**<div class="row">**

**<div class="col-md-8">**

**<section id="loginForm">**

**<input type="hidden" id="AccountLoginURL" value=@Url.Action("Login", "Account") />**

**@Html.AntiForgeryToken()**

**<h4>Use a local account to log in.</h4>**

**<hr />**

**@Html.ValidationSummary(true, "", new { @class = "text-danger" })**

**<div class="form-group">**

**@Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })**

**<div class="col-md-10">**

**@Html.TextBoxFor(m => m.Email, new { @class = "form-control",id= "userNameTextBox" })**

**@Html.ValidationMessageFor(m => m.Email, "", new { @class = "text-danger" })**

**</div>**

**</div>**

**<div class="form-group">**

**@Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })**

**<div class="col-md-10">**

**@Html.PasswordFor(m => m.Password, new { @class = "form-control", id = "passwordTextBox" })**

**@Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })**

**</div>**

**</div>**

**<div class="form-group">**

**<div class="col-md-offset-2 col-md-10">**

**<div class="checkbox">**

**@Html.CheckBoxFor(m => m.RememberMe)**

**@Html.LabelFor(m => m.RememberMe)**

**</div>**

**</div>**

**</div>**

**<div class="form-group">**

**<div class="col-md-offset-2 col-md-10">**

**<input type="submit" value="Log in" class="btn btn-default" id="login" />**

**</div>**

**</div>**

**<p>**

**@Html.ActionLink("Register as a new user", "Register")**

**</p>**

**<p>**

**@Html.Label("PublicKey", File.ReadAllText(AppDomain.CurrentDomain.BaseDirectory + @"/Content/PublicKey.pem"), new { id= "PublicKey" })**

**</p>**

**</section>**

**</div>**

**<div class="col-md-4">**

**<section id="socialLoginForm">**

**@Html.Partial("\_ExternalLoginsListPartial", new ExternalLoginListViewModel { ReturnUrl = ViewBag.ReturnUrl })**

**</section>**

**</div>**

**</div>**

**@section Scripts {**

**@Scripts.Render("~/bundles/jqueryval")**

**}**

**<script src="~/Scripts/jquery-1.10.2.min.js"></script>**

**<script src="~/Scripts/jsencrypt.js"></script>**

**<script src="~/Scripts/Login.js"></script>**

* 1. **Scripts文件夹下新增 Login.js, jsencrypt.js文件**

jsencrypt.js文件请从“参考链接3” 中获取.

Login.js代码如下：

$(function () {

$('#login').click(function () {

var crypt = new JSEncrypt();

var publicKey = $('#PublicKey').text();

var antiForgeryToken = $("input[name = '\_\_RequestVerificationToken']").val();

var password = $('#passwordTextBox').val();

var userName = $('#userNameTextBox').val();

var AccountLoginURL = $('#AccountLoginURL').val();

var encryptedPassword;

crypt.setPublicKey(publicKey);

encryptedPassword = crypt.encrypt(password);

console.log(encryptedPassword);

//data to be transported

var LoginModel = {

\_RequestVerificationToken: antiForgeryToken, //Reading text box values using Jquery

email: userName,

plainTextPassword: password,

encryptedPassword: encryptedPassword,

rememberMe: 'false'

};

//add antiForgeryToken

LoginModel.\_\_RequestVerificationToken = antiForgeryToken;

$.ajax({

type: "POST",

url: AccountLoginURL,

data: LoginModel,

success: function (result) {

console.log('succeed:' + result);

},

dataType: 'json',

error: function (err) {

alert("error - " + err);

}

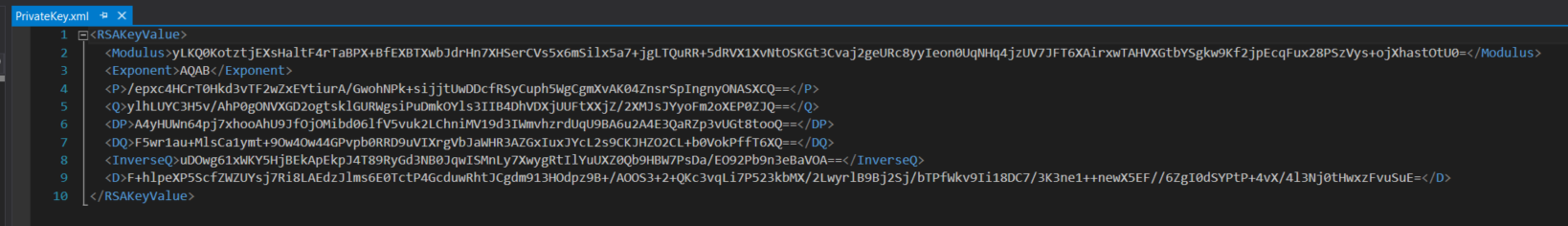
});

});

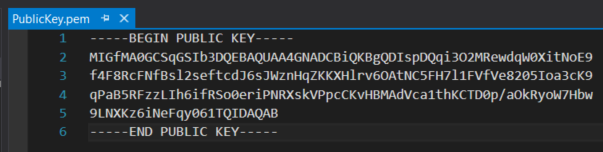
})

* 1. **配置publicKey和PrivateKey**

通过“3参考链接“中连接2，生成公钥和私钥。 公钥保持pem格式，因为JS类库使用的需要。 把私钥通过“3参考链接“中连接4（转换器）转换成XML格式，因为.NET 能够识别XML格式的私钥。



Xml格式私钥



Pem格式公钥

* 1. **Controller端**

Controllers->AccountController->Login()

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Login()

{

var form = Request.Form;

string plainTextPassword = form["plainTextPassword"].ToString();

string encryptedPassword = form["encryptedPassword"].ToString();

//decrypt

String privateKeyPathFile = AppDomain.CurrentDomain.BaseDirectory + @"\Content\PrivateKey.xml";

string RSAprivateKey = System.IO.File.ReadAllText(privateKeyPathFile);

RSAEncryption rsaCryption = new RSAEncryption();

string decryptedPwd = rsaCryption.RSADecrypt(RSAprivateKey, encryptedPassword);

return View();

}

* 1. **RSAEncryption**

/// <summary>

/// RSA加密解密及RSA签名和验证

/// </summary>

public class RSAEncryption

{

public RSAEncryption()

{

}

#region RSA 加密解密

#region RSA 的密钥产生

/// <summary>

/// RSA 的密钥产生 产生私钥 和公钥

/// </summary>

/// <param name="xmlKeys"></param>

/// <param name="xmlPublicKey"></param>

public void RSAKey(out string xmlKeys, out string xmlPublicKey)

{

System.Security.Cryptography.RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

xmlKeys = rsa.ToXmlString(true);

xmlPublicKey = rsa.ToXmlString(false);

}

#endregion

#region RSA的加密函数

//##############################################################################

//RSA 方式加密

//说明KEY必须是XML的行式,返回的是字符串

//在有一点需要说明！！该加密方式有 长度 限制的！！

//##############################################################################

//RSA的加密函数 string

public string RSAEncrypt(string xmlPublicKey, string m\_strEncryptString)

{

byte[] PlainTextBArray;

byte[] CypherTextBArray;

string Result;

RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

rsa.FromXmlString(xmlPublicKey);

PlainTextBArray = (new UnicodeEncoding()).GetBytes(m\_strEncryptString);

CypherTextBArray = rsa.Encrypt(PlainTextBArray, false);

Result = Convert.ToBase64String(CypherTextBArray);

return Result;

}

//RSA的加密函数 byte[]

public string RSAEncrypt(string xmlPublicKey, byte[] EncryptString)

{

byte[] CypherTextBArray;

string Result;

RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

rsa.FromXmlString(xmlPublicKey);

CypherTextBArray = rsa.Encrypt(EncryptString, false);

Result = Convert.ToBase64String(CypherTextBArray);

return Result;

}

#endregion

#region RSA的解密函数

//RSA的解密函数 string

public string RSADecrypt(string xmlPrivateKey, string m\_strDecryptString)

{

byte[] PlainTextBArray;

byte[] DypherTextBArray;

string Result;

System.Security.Cryptography.RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

rsa.FromXmlString(xmlPrivateKey);

PlainTextBArray = Convert.FromBase64String(m\_strDecryptString);

DypherTextBArray = rsa.Decrypt(PlainTextBArray, false);

Result = (new ASCIIEncoding()).GetString(DypherTextBArray);

return Result;

}

//RSA的解密函数 byte

public string RSADecrypt(string xmlPrivateKey, byte[] DecryptString)

{

byte[] DypherTextBArray;

string Result;

System.Security.Cryptography.RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();

rsa.FromXmlString(xmlPrivateKey);

DypherTextBArray = rsa.Decrypt(DecryptString, false);

Result = (new UnicodeEncoding()).GetString(DypherTextBArray);

return Result;

}

#endregion

#endregion

#region RSA数字签名

#region 获取Hash描述表

//获取Hash描述表

public bool GetHash(string m\_strSource, ref byte[] HashData)

{

//从字符串中取得Hash描述

byte[] Buffer;

System.Security.Cryptography.HashAlgorithm MD5 = System.Security.Cryptography.HashAlgorithm.Create("MD5");

Buffer = System.Text.Encoding.GetEncoding("GB2312").GetBytes(m\_strSource);

HashData = MD5.ComputeHash(Buffer);

return true;

}

//获取Hash描述表

public bool GetHash(string m\_strSource, ref string strHashData)

{

//从字符串中取得Hash描述

byte[] Buffer;

byte[] HashData;

System.Security.Cryptography.HashAlgorithm MD5 = System.Security.Cryptography.HashAlgorithm.Create("MD5");

Buffer = System.Text.Encoding.GetEncoding("GB2312").GetBytes(m\_strSource);

HashData = MD5.ComputeHash(Buffer);

strHashData = Convert.ToBase64String(HashData);

return true;

}

//获取Hash描述表

public bool GetHash(System.IO.FileStream objFile, ref byte[] HashData)

{

//从文件中取得Hash描述

System.Security.Cryptography.HashAlgorithm MD5 = System.Security.Cryptography.HashAlgorithm.Create("MD5");

HashData = MD5.ComputeHash(objFile);

objFile.Close();

return true;

}

//获取Hash描述表

public bool GetHash(System.IO.FileStream objFile, ref string strHashData)

{

//从文件中取得Hash描述

byte[] HashData;

System.Security.Cryptography.HashAlgorithm MD5 = System.Security.Cryptography.HashAlgorithm.Create("MD5");

HashData = MD5.ComputeHash(objFile);

objFile.Close();

strHashData = Convert.ToBase64String(HashData);

return true;

}

#endregion

#region RSA签名

//RSA签名

public bool SignatureFormatter(string p\_strKeyPrivate, byte[] HashbyteSignature, ref byte[] EncryptedSignatureData)

{

System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptography.RSACryptoServiceProvider();

RSA.FromXmlString(p\_strKeyPrivate);

System.Security.Cryptography.RSAPKCS1SignatureFormatter RSAFormatter = new System.Security.Cryptography.RSAPKCS1SignatureFormatter(RSA);

//设置签名的算法为MD5

RSAFormatter.SetHashAlgorithm("MD5");

//执行签名

EncryptedSignatureData = RSAFormatter.CreateSignature(HashbyteSignature);

return true;

}

//RSA签名

public bool SignatureFormatter(string p\_strKeyPrivate, byte[] HashbyteSignature, ref string m\_strEncryptedSignatureData)

{

byte[] EncryptedSignatureData;

System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptography.RSACryptoServiceProvider();

RSA.FromXmlString(p\_strKeyPrivate);

System.Security.Cryptography.RSAPKCS1SignatureFormatter RSAFormatter = new System.Security.Cryptography.RSAPKCS1SignatureFormatter(RSA);

//设置签名的算法为MD5

RSAFormatter.SetHashAlgorithm("MD5");

//执行签名

EncryptedSignatureData = RSAFormatter.CreateSignature(HashbyteSignature);

m\_strEncryptedSignatureData = Convert.ToBase64String(EncryptedSignatureData);

return true;

}

//RSA签名

public bool SignatureFormatter(string p\_strKeyPrivate, string m\_strHashbyteSignature, ref byte[] EncryptedSignatureData)

{

byte[] HashbyteSignature;

HashbyteSignature = Convert.FromBase64String(m\_strHashbyteSignature);

System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptography.RSACryptoServiceProvider();

RSA.FromXmlString(p\_strKeyPrivate);

System.Security.Cryptography.RSAPKCS1SignatureFormatter RSAFormatter = new System.Security.Cryptography.RSAPKCS1SignatureFormatter(RSA);

//设置签名的算法为MD5

RSAFormatter.SetHashAlgorithm("MD5");

//执行签名

EncryptedSignatureData = RSAFormatter.CreateSignature(HashbyteSignature);

return true;

}

//RSA签名

public bool SignatureFormatter(string p\_strKeyPrivate, string m\_strHashbyteSignature, ref string m\_strEncryptedSignatureData)

{

byte[] HashbyteSignature;

byte[] EncryptedSignatureData;

HashbyteSignature = Convert.FromBase64String(m\_strHashbyteSignature);

System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptography.RSACryptoServiceProvider();

RSA.FromXmlString(p\_strKeyPrivate);

System.Security.Cryptography.RSAPKCS1SignatureFormatter RSAFormatter = new System.Security.Cryptography.RSAPKCS1SignatureFormatter(RSA);

//设置签名的算法为MD5

RSAFormatter.SetHashAlgorithm("MD5");

//执行签名

EncryptedSignatureData = RSAFormatter.CreateSignature(HashbyteSignature);

m\_strEncryptedSignatureData = Convert.ToBase64String(EncryptedSignatureData);

return true;

}

#endregion

#region RSA 签名验证

public bool SignatureDeformatter(string p\_strKeyPublic, byte[] HashbyteDeformatter, byte[] DeformatterData)

{

System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptography.RSACryptoServiceProvider();

RSA.FromXmlString(p\_strKeyPublic);

System.Security.Cryptography.RSAPKCS1SignatureDeformatter RSADeformatter = new System.Security.Cryptography.RSAPKCS1SignatureDeformatter(RSA);

//指定解密的时候HASH算法为MD5

RSADeformatter.SetHashAlgorithm("MD5");

if (RSADeformatter.VerifySignature(HashbyteDeformatter, DeformatterData))

{

return true;

}

else

{

return false;

}

}

public bool SignatureDeformatter(string p\_strKeyPublic, string p\_strHashbyteDeformatter, byte[] DeformatterData)

{

byte[] HashbyteDeformatter;

HashbyteDeformatter = Convert.FromBase64String(p\_strHashbyteDeformatter);

System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptography.RSACryptoServiceProvider();

RSA.FromXmlString(p\_strKeyPublic);

System.Security.Cryptography.RSAPKCS1SignatureDeformatter RSADeformatter = new System.Security.Cryptography.RSAPKCS1SignatureDeformatter(RSA);

//指定解密的时候HASH算法为MD5

RSADeformatter.SetHashAlgorithm("MD5");

if (RSADeformatter.VerifySignature(HashbyteDeformatter, DeformatterData))

{

return true;

}

else

{

return false;

}

}

public bool SignatureDeformatter(string p\_strKeyPublic, byte[] HashbyteDeformatter, string p\_strDeformatterData)

{

byte[] DeformatterData;

System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptography.RSACryptoServiceProvider();

RSA.FromXmlString(p\_strKeyPublic);

System.Security.Cryptography.RSAPKCS1SignatureDeformatter RSADeformatter = new System.Security.Cryptography.RSAPKCS1SignatureDeformatter(RSA);

//指定解密的时候HASH算法为MD5

RSADeformatter.SetHashAlgorithm("MD5");

DeformatterData = Convert.FromBase64String(p\_strDeformatterData);

if (RSADeformatter.VerifySignature(HashbyteDeformatter, DeformatterData))

{

return true;

}

else

{

return false;

}

}

public bool SignatureDeformatter(string p\_strKeyPublic, string p\_strHashbyteDeformatter, string p\_strDeformatterData)

{

byte[] DeformatterData;

byte[] HashbyteDeformatter;

HashbyteDeformatter = Convert.FromBase64String(p\_strHashbyteDeformatter);

System.Security.Cryptography.RSACryptoServiceProvider RSA = new System.Security.Cryptography.RSACryptoServiceProvider();

RSA.FromXmlString(p\_strKeyPublic);

System.Security.Cryptography.RSAPKCS1SignatureDeformatter RSADeformatter = new System.Security.Cryptography.RSAPKCS1SignatureDeformatter(RSA);

//指定解密的时候HASH算法为MD5

RSADeformatter.SetHashAlgorithm("MD5");

DeformatterData = Convert.FromBase64String(p\_strDeformatterData);

if (RSADeformatter.VerifySignature(HashbyteDeformatter, DeformatterData))

{

return true;

}

else

{

return false;

}

}

#endregion

#endregion

}

**2 小结**

本文提出了在ASP.NET MVC中，密码传输安全问题，提出了3种可行解决方案。重点讲述了RSA不对称加密的实现方式，同时保留了微软自带的AntiForgeryToken, 以防止CSRF攻击。达到了密文传输密码的效果，即使被人截获，也无法得知密码明文。

作者知识和精力都有限，如有不足，欢迎指正。

**3 参考链接**

1 本文详细代码，请参考GitHub: <https://github.com/memoryfraction/CommonUsedFunctions/tree/master/WebApplication_JS_RSA>

2 JS Encrypt的Demo，连接: <http://travistidwell.com/jsencrypt/demo/>

3 JSEncrypt的主页：http://travistidwell.com/jsencrypt/

4 Key的格式转换: <https://superdry.apphb.com/tools/online-rsa-key-converter>

5 《[什么是CSRF攻击，如何在ASP.NET MVC网站中阻止这种攻击？](http://blog.csdn.net/fanrong1985/article/details/71701301)》http://blog.csdn.net/fanrong1985/article/details/71701301