راه‌اندازی تجارت الکترونیکی

با

ASP.Net

(کدهای C#)

از مقدماتی تا پیشرفته

«جلد دوم»

مهندس مسلم افراشته مهر، رضا افراشته مهر

انتشارت پندار پارس

سرشناسه : افراشته‌مهر، مسلم، ۱۳۶۵ -، گردآورنده، مترجم
عنوان و نام پدیدآور : راه‌اندازی تجارت الکترونیکی با ASP.Net کدهای (#C) از مقدماتی تا پیشرفته/[گردآورنده و] مترجم مسلم افراشته‌مهر.
مشخصات نشر : تهران: پندار پارس : مانلی، ۱۳۹۰ -
مشخصات ظاهری : مصور، جدول. (ج۱: ۵۶۰ ص. ج۲: ۲۱۶ ص.)
شابک : دوره :978-964-2989-85-0 : ج۲: 978-964-2989-84-3: ۹۸۰۰۰ ریال
وضعیت فهرست نویسی : فیپا
یادداشت : کتابنامه.
موضوع : صفحه‌های سرور فعال
موضوع : بازرگانی الکترونیکی -- برنامه‌های کامپیوتری -- دستنامه‌ها
موضوع : مایکروسافت دات نت
رده بندی کنگره : HF۵۵۴۸/۱۳۹۰۳۲ ۲ر۶۶الف/
رده بندی دیویی : ۸۷۲۰۲۸۵/۶۵۸
شماره کتابشناسی ملی : ۰۹۱۲۹۴۲

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوچه رشتچی، شماره ۱٤، واحد ۱٦ **www.pendarepars.com**

تلفن: ٦٦٥٧٢٣٣٥ - تلفکس: ٦٦٩٢٦٥٧٨همراه: ۰۹۱۲۲٤٥٢٣٤٨ **info@pendarepars.com**

نام کتاب : راه‌اندازی تجارت الکترونیکی با ASP.Net (کدهای #C) (جلد ۲)

ناشر : انتشارات پندار پارس

ترجمه و تالیف : مسلم افراشته مهر، رضا افراشته مهر

چاپ نخست : پاییز ۹۱

شمارگان : ۱۰۰۰ نسخه

طرح جلد : فرزانه روزبهانی

لیتوگرافی، چاپ، صحافی : ترام‌سنج، صالحان، خیام

قیمت : ۹۸۰۰ تومان شابک : ۳-۸٤-۲۹۸۹-۹٦٤-۹۷۸ شابک دوره: ۰-۸۵-۲۹۸۹-۹٦٤-۹۷۸

فهرست

فصل 16

ایجاد حساب مشتریان

تا این قسمت از کتاب، یک سایت اولیه (اما کاربردی) ایجاد کرده‌اید و برای پرداخت هزینه‌ها و تأیید سفارش‌ها، آن را به سیستم پرداخت Paypal متصل نموده‌اید. در فاز آخر کتاب (فاز سوم)، قصد داریم ابزارهای دیگری را به پروژه اضافه کنیم. با قطع ارتباط سایت از سیستم پرداخت Paypal، می‌توان کنترل بهتری بر روی سایت داشت و سربار اضافی آن را کاهش داد. این عمل به آن پیچیدگی که فکر می‌کنید نیست، اما باید در نظر داشته باشید کارها باید به‌درستی انجام گیرد.

این فصل، با پیاده‌سازی یک سیستم حساب مشتری، مقدمه‌ای بر این عمل می‌باشد. برای ساخت سایت‌های تجارت الکترونیکی مشتری‌پسند، باید جزئیاتی از قبیل شماره‌ی کارت‌های اعتباری مربوط به آنها را در پایگاه‌داده ذخیره کنید، تا کاربر در هر بار سفارش، نیازی به ورود دوباره‌ی این اطلاعات نداشته باشد. سیستم حساب مشتری که پیاده‌سازی خواهید کرد این کارها را انجام داده و همه‌ی صفحه‌های وبی که برای ورود این اطلاعات موردنیاز می‌باشد را در اختیار دارد.

افزون بر پیاده‌سازی این صفحه‌های وب، چندین فاکتور دیگر را نیز باید در نظر داشت: نخست اینکه اطلاعات حساسی مانند شماره‌ی کارت‌های اعتباری، تاریخ انقضا و غیره را باید در پایگاه‌داده، ذخیره کنید و ذخیره‌کردن آنها در یک فایل متنی ساده، ایده‌ی جالبی نیست. به‌جای اعمال سیاست دسترسی محدود به این قبیل داده‌ها، بهتر است این اطلاعات حساس را رمزگذاری کرده و با برنامه‌ریزی دقیق در هر زمان که موردنیاز باشند، آنها را بازیابی کنید. برای راحتی کار، می‌توان یک کتابخانه‌ی امنیتی ساخت.

دوم، از آنجایی‌که قرار است این اطلاعات حساس را به‌وسیله‌ی اینترنت از مشتریان دریافت نمایید، بنابراین داشتن یک ارتباط امن، امری لازم و ضروری می‌باشد. نمی‌توانید تنها یک فرم ساده ایجاد کنید و به مشتری اجازه دهید به‌وسیله‌ی پروتکل HTTP به آن دسترسی پیدا کرده، آن را پر نموده و برای شما ارسال کند. خواهید آموخت که چگونه با استفاده از SSL بر روی ارتباطات HTTPS، این مشکل را حل کنید.

در این فصل می‌آموزید که چگونه:

- حساب مشتریان را ذخیره کنید،
- کلاس‌های امن و مطمئن، پیاده‌سازی نمایید،
- عملیات حساب مشتری را به Balloonshop اضافه کنید،
- صفحه‌ی Checkout را ایجاد نمایید.

ذخیره‌سازی حساب مشتریان

به روش‌های زیادی می‌توان عملیات حساب مشتری را پیاده‌سازی نمود، ولی در هر صورت، عملیات زیر در همه‌ی روش‌ها مشترک است:

- مشتریان برای دسترسی به قسمت‌های محافظت شده‌ی سایت، نخست باید وارد سایت شوند (Login).
- به‌خاطر سپردن مشتریان پس از نخستین ورود آنها به سایت، تا زمانی‌که از سایت خارج شوند (مشتری زمانی از سایت خارج می‌شود، که به‌شکل دستی بر روی دکمه‌ی Logout کلیک کند، به‌شکل خودکار زمان Session تمام شود یا خطایی بر روی سرور رخ دهد).
- تمام صفحه‌های محافظت شده‌ی سایت، پیش از اینکه به مشتریان اجازه‌ی دسترسی به قسمت‌های گوناگون سایت را دهند، باید از ورود آنها به سایت، اطمینان حاصل کنند.

نخست اجازه دهید نگاهی به جزئیات پیاده‌سازی عمومی سایت تجارت الکترونیکی Balloonshop داشته باشیم.

ایجاد یک طرح[1] حساب مشتری برای Balloonshop

تا اینجا عملیات زیادی انجام داده‌اید. به فصل‌های 11 و 12 برگردید: یک سیستم تعیین هویت، ایجاد کردید به‌گونه‌ای‌که مدیران سایت به‌وسیله‌ی آن سیستم، شناسایی شده و قادر به دست‌کاری محصولات بودند. برای انجام این عمل، یک صفحه‌ی ورود به سایت (Login.aspx)، ایجاد نمودید تا کاربرانی که در نقش[2] Administrators قرار دارند، به‌وسیله‌ی آن وارد سایت شوند و به قسمت‌های مدیریتی آن دسترسی داشته باشند. وضعیت کاربر جاری (اینکه آیا کاربر، وارد سیستم شده است یا نه) با استفاده از یک کنترل کاربری[3]، که ایجاد نموده‌اید (Login.ascx)، نمایش داده می‌شود.

---

[1] Scheme
[2] Role
[3] user control

در این فصل، این سیستم را به‌گونه‌ای توسعه می‌دهیم تا برای مشتریان نیز قابل استفاده باشد. تغییرهای زیادی برای فعال‌سازی این بخش باید انجام داد; از جمله اینکه باید یک نقش جدید (با نام Customers)، برای مشتریان اضافه کنید. مشتریان برای ورود به سایت، از یک صفحه‌ی ورود (همانند صفحه‌ی استفاده‌شده برای ورود مدیران) استفاده می‌کنند ولی نقش آنها متفاوت است (مدیران به قسمت‌های مدیریتی سایت دسترسی دارند و می‌توانند کارهایی از قبیل ایجاد دسته، قراردادن محصول‌ها در دسته‌ها، حذف محصول‌ها و ... را انجام دهند ولی مشتریان پس از ورود به سایت می‌توانند جزئیات مربوط به سفارش از قبیل آدرس، اطلاعات تماس، شماره‌ی کارت اعتباری و ... را دیده و در صورت نیاز، تغییر دهند). عمل دیگری که در این فصل انجام خواهیم داد، این است که یک صفحه‌ی ثبت‌نام ایجاد می‌کنیم تا مشتریان جدید بتوانند در سایت، ثبت‌نام نمایند.

همان‌گونه‌که می‌بینید، اکنون که می‌خواهید این عملیات را برای مشتریان انجام دهید، میزان داده‌هایی که باید ذخیره شوند در حال افزایش است. خوشبختانه ASP.Net برای حل اینگونه مشکل‌ها، مفهومی به اسم پروفایل کاربران[1] (سیستم ذخیره‌سازی نرمش‌پذیری که مناسب این نیاز است) را معرفی کرده است. در ادامه‌ی فصل، خواهید دید چگونه پروفایل کاربران به‌سرعت با استفاده از فایل web.config، پیکربندی شده و می‌توانید از کد خودتان به این اطلاعات متصل شوید.

کار مهم دیگری که در سیستم حساب مشتریان انجام خواهید داد (همان‌گونه که در فصل 11، برای مدیران انجام دادید) این است که رمز عبور مربوط به مشتریان را، به‌شکل درهم‌برهم (Hash) ذخیره کنید. از آنجایی‌که رمز عبور مربوط به مشتریان، همواره مورد حمله‌ی نفوذگران قرار می‌گیرد، بنابراین ذخیره‌کردن آنها به‌شکل ساده در پایگاه‌داده‌ها، ایده‌ی خوبی نیست.

کد درهم‌برهم، رشته‌ی منحصربه‌فردی می‌باشد که نشان‌دهنده‌ی یک رمز عبور است. برای اعتبارسنجی رمز عبور وارد شده به‌وسیله‌ی مشتری، کافی است یک کد درهم‌برهم برای رمز عبور واردشده ایجاد و سپس آن را با کد درهم‌برهم شده‌ی موجود در پایگاه‌داده‌ها، مقایسه کنید. در صورت هماهنگی این دو رمز عبور، می‌توان اطمینان داشت که رمز عبور وارد شده، درست است. این کار را با استفاده از ASP.NET forms authentication system انجام می‌دهیم.

درهم‌برهم کردن رمز عبور، یکی از روش‌های امنیتی کارت اعتباری است. کار دیگری که باید انجام دهید این است که اطلاعات کارت اعتباری را رمزگذاری نمایید. این کار سبب می‌شود این اطلاعات با امنیت بیشتری ذخیره شود و در زمان لازم، مورد دسترسی قرار گیرد.

به‌شکل ویژه، برای پیاده‌سازی این مفهوم در برنامه، عملیات زیر را انجام خواهیم داد:

---

[1] user profile

- یک شمای[1] پروفایل کاربری، به برنامه اضافه می‌کنید،
- سایت را به‌گونه‌ای تغییر می‌دهید تا مشتریان بتوانند جزئیات مربوط به سفارش را مشاهده و دست‌کاری نمایند، و همچنین مشتریان جدید بتوانند ثبت‌نام کنند،
- صفحه‌ی ShoppingCart.ascx را، به‌گونه‌ای تغییر می‌دهید تا مشتریان را به صفحه‌ی Checkout.aspx هدایت کند.

کلاس‌های SecurityLib

تاکنون دو کار امنیتی در سایت انجام داده‌ایم:

- درهم برهم کردن رمزهای ورود
- رمزگذاری کارت‌های اعتباری

کار کدهای مربوط به هر بخش را در کلاس‌هایی درون پوشه‌ای به‌نام SecurityLib (که خود در پوشه‌ی App_Code جای گرفته است) قرار می‌دهیم. از آنجاکه ممکن است از این کدها در دیگر برنامه‌ها استفاده کنید، بنابراین کدهای این بخش را مستقل از کد اصلی برنامه می‌نویسیم. داشتن تمام فایل‌های وابسته به‌هم در یک مکان، این امکان را فراهم می‌سازد که به‌راحتی بتوانید آنها را در هر مکانی کپی و یا حتی آنها را استخراج کنید و در یک کتابخانه‌ی کلاس اشتراکی، قرار دهید. برای استفاده‌ی بهتر، تمام کلاس‌های موجود در این پوشه را در یک فضای نام[2] جداگانه به‌نام SecurityLib قرار می‌دهیم. از آنجاکه Visual Web Developer Express اجازه‌ی ایجاد کتابخانه‌ی کلاس‌ها[3] را به شما نمی‌دهد، بنابراین برای اشتراکی کردن کد موجود در یک کتابخانه‌ی کلاس، باید از Visual C# Express یا نسخه‌ی کامل Visual Studio استفاده کنید.

پوشه‌ی SecurityLib، دارای فایل‌های زیر می‌باشد:

- **PasswordHasher.cs**: شامل کلاس PasswordHasher است که درون آن، متد اشتراکی شده‌ای[4] به‌نام Hash قراردارد و وظیفه دارد برای رمز عبور وارد شده، یک کد درهم‌برهم فراهم نماید.
- **SecureCard.cs**: شامل کلاس SecureCard است که نشان‌دهنده‌ی یک کارت اعتباری می‌باشد. این کلاس می‌تواند با اطلاعات کارت اعتباری، مقداردهی آغازین شود که سپس با فرمت رمزگذاری

---

[1] Schema
[2] namespace
[3] class library
[4] Shared method

شده، قابل دسترسی است. به‌طور جایگزین می‌تواند با داده‌های کارت اعتباری رمزگذاری‌شده، مقداردهی شود و دسترسی به اطلاعات رمزگشایی‌شده‌ی درون آن را فراهم آورد.

- **SecureCardException.cs**: در روند رمزگذاری یا رمزگشایی، هر نوع مشکلی ممکن است به‌وجود آید. استثنای موجود در این فایل (SecureCardException)، به‌وسیله‌ی SecureCard استفاده می‌شود.
- **StringEncryptor.cs**: کلاس موجود در این فایل (StringEncryptor)، به‌وسیله‌ی SecureCard استفاده می‌شود تا اطلاعات را رمزگذاری و رمزگشایی کند. این به‌این مفهوم است که اگر قصد داشتید روش رمزگذاری را تغییر دهید، تنها باید کد موجود در این کلاس، تغییر پیدا کند (نیازی به تغییر کلاس SecureCard نیست).
- **StringEncryptorException.cs**: حاوی استثنای StringEncryptorException می‌باشد و زمانی فراخوانی می‌شود که در کلاس StringEncryptor خطایی رخ دهد.

نخست، به کدی که برای درهم‌برهم کردن رمزهای عبور استفاده می‌شود نگاهی داشته باشیم و در ادامه، کد مربوط به رمزگذاری را بررسی می‌کنیم.

درهم‌برهم کردن

همان‌گونه‌که پیش از این گفته شد، کد درهم‌برهم، یک رشته‌ی منحصربه‌فردی است که نشان‌دهنده‌ی رمز عبور می‌باشد. برای درهم‌برهم کردن رمز عبور، عملیات زیر انجام می‌گیرد:

- شیئی که قرار است درهم‌برهم شود را به آرایه‌ی بایتی[1] سریال‌سازی[2] می‌کند.
- آرایه‌ی بایتی را درهم‌برهم کرده و یک آرایه‌ی بایتی درهم‌برهم شده‌ی جدید به‌دست می‌آورد.
- آرایه‌ی بایتی درهم‌برهم شده را به فرمتی تبدیل می‌کند که برای ذخیره‌سازی موردنیاز است.

این کار برای رمزهای عبور راحت است، زیرا تبدیل یک رشته (که آرایه‌ای از کاراکترها است) به یک آرایه‌ی بایتی، کار مشکلی نیست. تبدیل آرایه‌ی بایتی درهم‌برهم شده به یک رشته، برای ذخیره‌شدن در پایگاه‌داده‌ها و مقایسه‌ی سریع آنها نیز، کار آسانی است.

متد واقعی استفاده شده برای تبدیل آرایه‌ی بایتی منبع، به یک آرایه‌ی بایتی درهم‌برهم شده می‌تواند متعدد باشد. فضای‌نام System.Security.Cryptography، دارای تعداد زیادی کلاس برای درهم‌برهم

[1] byte array
[2] Serializing

کردن است (البته وارد جزئیات کلاس‌های موجود در این فضای‌نام نمی‌شویم).SHA1 [1] و MD5 [2] دو الگوریتم اصلی برای درهم‌برهم کردن هستند، که در .NET Framework وجود دارند. SHA1، یک کد درهم‌برهم 160 بیتی تولید می‌کند در حالی‌که MD5، یک کد 128 بیتی ایجاد می‌نماید؛ به‌همین دلیل SHA1، امنیت بیشتری نسبت به MD5 دارد (اگرچه کندتر است). .NET Framework دارای نسخه‌های دیگری از SHA1 است که می‌تواند کدهای بزرگتری تولید کند (تا 512 بیت).

در پیاده‌سازی SecureLib، از الگوریتم SHA1 استفاده می‌کنید (البته اگر به امنیت بیشتری نیاز داشته باشید به‌راحتی می‌توانید آن را تغییر دهید). کد لازم برای دست‌یابی به این قسمت را، در تمرین زیر (کلاس PasswordHasher) خواهید دید.

تمرین: پیاده‌سازی کلاس PasswordHasher

1. یک زیر پوشه‌ی جدید در App_Code ایجاد و آن را SecurityLib نام‌گذاری کنید.
2. یک فایل کلاس جدید به‌نام PasswordHasher.cs، به پوشه‌ی SecurityLib بیافزایید و کد زیر را به آن اضافه کنید:

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

namespace SecurityLib
{
  public static class PasswordHasher
  {
    private static SHA1Managed hasher = new SHA1Managed();

    public static string Hash(string password)
    {
      // convert password to byte array
      byte[] passwordBytes =
        System.Text.ASCIIEncoding.ASCII.GetBytes(password);

      // generate hash from byte array of password
      byte[] passwordHash = hasher.ComputeHash(passwordBytes);

      // convert hash to string
      return Convert.ToBase64String(passwordHash, 0, passwordHash.Length);
    }
  }
}
```

[1] Secure Hash Algorithm
[2] Message Digest

3. یک صفحه‌ی وب[1] با نام SecurityLibTester.aspx، به ریشه‌ی وب‌سایت BalloonShop اضافه کنید (صفحه‌ی مستر Ballonshop پیش‌فرض را انتخاب کنید).

4. کد زیر را به SecurityLibTester.aspx اضافه کنید:

```
<%@ Page Title="SecurityLib Test Page" Language="C#" MasterPageFile=
"~/BalloonShop.master" AutoEventWireup="true" CodeFile="SecurityLibTester
.aspx.cs" Inherits="SecurityLibTester" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlace
Holder1" runat="Server">
        Enter your password:<br />
        <asp:TextBox ID="pwdBox1" runat="server" />
        <br />
        Enter your password again:
        <br />
        <asp:TextBox ID="pwdBox2" runat="server" />
        <br />
        <asp:Button ID="processButton" runat="server" Text="Process"
      OnClick="processButton_Click" />
        <br />
        <asp:Label ID="result" runat="server" />
      </asp:Content>
```

5. SecurityLibTester.aspx.cs را همانند زیر، تغییر دهید:

```
using System;
using System.Text;
using SecurityLib;

public partial class SecurityLibTester : System.Web.UI.Page
{
  protected void processButton_Click(object sender, EventArgs e)
  {
    string hash1 = PasswordHasher.Hash(pwdBox1.Text);
    string hash2 = PasswordHasher.Hash(pwdBox2.Text);
    StringBuilder sb = new StringBuilder();
    sb.Append("The hash of the first password is: ");
    sb.Append(hash1);
    sb.Append("<br />The hash of the second password is: ");
    sb.Append(hash2);
    if (hash1 == hash2)
    {
      sb.Append("<br />The passwords match! Welcome!");
    }
    else
    {
      sb.Append("<br />Password invalid. "
```

[1] web page

```
            + "Armed guards are on their way.");
        }
        result.Text = sb.ToString();
    }
}
```

6. SecurityLibTester.aspx را بارگذاری، دو رمز عبور را وارد و بر روی دکمه‌ی Process کلیک کنید. نتیجه در شکل 1-16 نشان داده شده است.

شکل 1-16. نتیجه‌ی رمز عبور درهم‌برهم شده

روش کار: پیاده‌سازی کلاس PasswordHasher

کد موجود در کلاس PasswordHasher، گام‌هایی که پیش از این برای درهم‌برهم کردن رمز عبور توضیح دادیم را دنبال می‌کند.

ابتدا با استفاده از متد کمکی System.Text.ASCIIEncoding.ASCII.GetBytes، رشته‌ی رمز عبور وارد شده را به یک آرایه‌ی بایتی تبدیل می‌کند:

```
// convert password to byte array
byte[] passwordBytes =
  System.Text.ASCIIEncoding.ASCII.GetBytes(password);
```

سپس از عضو خصوصی اشتراکی hasher (یک نمونه از SHA1Managed) برای تولید یک آرایه‌ی بایتی درهم‌برهمی استفاده می‌کند:

```
// generate hash from byte array of password
byte[] passwordHash = hasher.ComputeHash(passwordBytes);
```

در پایان، با استفاده از تابع کمکی Convert.ToBase64String، کد درهم‌برهم شده را به یک رشته تبدیل می‌کند و نتیجه را برمی‌گرداند:

```
        // convert hash to string
        return Convert.ToBase64String(passwordHash, 0, passwordHash.Length);
```

همه‌ی کلاس‌های موجود در .NET Framework. که کار درهم‌برهمی را انجام می‌دهند، از متد ComputeHash برای گرفتن یک شکل درهم‌برهم از یک آرایه‌ی بایتی ورودی استفاده می‌کنند. برای افزایش اندازه‌ی درهم برهم، می‌توانید hasher را با کدهای دیگری جایگزین کنید. برای نمونه، به کد زیر نگاه کنید:

```
public static class PasswordHasher
    {
      private static SHA512Managed hasher = new SHA512Managed();
      ...
    }
```

تغییرات بالا، کد درهم برهم را به 512 بیت تغییر می‌دهد.

صفحه‌ی SecurityLibTest.aspx، دو رمز عبور را درهم‌برهم و نتیجه‌ها را با هم مقایسه می‌کند.

رمزگذاری

رمزگذاری، در شکل و اندازه‌های گوناگونی ظاهر می‌شود، و به‌شکل یک موضوع مهیج ادامه دارد. راه‌حل از پیش تعیین‌شده‌ای برای رمزگذاری وجود ندارد. به‌طور کلی، دو شکل رمزگذاری وجود دارد:

- رمزگذاری متقارن[1]: از یک کلید واحد، هم برای رمزگذاری و هم رمزگشایی استفاده می‌شود.
- رمز نگاری نامتقارن[2]: در این حالت، کلیدهای متفاوتی برای رمزگذاری و رمزگشایی داده‌ها استفاده می‌گردد. عموما کلید رمزگذاری، با نام کلید عمومی[3] شناخته می‌شود و هرکسی می‌تواند از آن برای رمزگذاری اطلاعات استفاده کند. کلید رمزگشایی، با نام کلید خصوصی[4] شناخته می‌شود؛ زیرا تنها می‌تواند برای رمزگشایی داده‌هایی استفاده شود که با استفاده از کلید عمومی رمزگذاری شده‌اند. کلید رمزگذاری (کلید عمومی) و کلید رمزگشایی (کلید خصوصی)، به‌شکل ریاضی به‌هم وابسته‌اند و همیشه باهم تولید می‌شوند. کلید عمومی و خصوصی، هر کدام نمی‌توانند با استفاده از دیگری به‌دست آیند.

---

[1] Symmetric encryption
[2] Asymmetric encryption
[3] Public Key
[4] Private Key

در برخی مکان‌ها از قبیل امضای دیجیتالی، کلید خصوصی برای رمزگذاری، و کلید عمومی برای رمزگشایی استفاده می‌شود. به‌هرشکل، در این فصل از این تکنیک‌ها استفاده نمی‌شود.

اگرچه روش نامتقارن، امن‌تر است ولی در عوض به قدرت پردازش بیشتری نیاز دارد. رمزگذاری متقارن سریع‌تر است، اما امنیت کمتری دارد؛ زیرا رمزگذار و رمزگشا هر دو از یک کلید یکتا استفاده می‌کنند. با رمزگذاری متقارن، رمزگذار باید کلید را برای رمزگشا ارسال کند.

در پروژه‌ی Balloonshop، عملیات مربوط به رمزگذاری و رمزگشایی، خیلی آسان‌تر از ارتباطات اینترنت می‌باشد. تنها کافی‌است اطلاعات را برای ذخیره‌سازی در پایگاه‌داده‌ها، رمزگذاری کنید و دوباره هر زمانی که موردنیاز بودند، آنها را رمزگشایی نمایید (به‌همین دلیل می‌توانید از یک الگوریتم متقارن، استفاده کنید).

الگوریتم‌های گوناگونی می‌توانند برای رمزگذاری متفارن و نامتقارن استفاده شوند. .NET Framework پیاده‌سازی‌های بی‌شماری از روش‌های بالا در فضای‌نامSystem. Security.Cryptography را داراست.

دو الگوریتم نامتقارن در دسترس، عبارت‌اند از: DSA[1] و RSA[2]. الگوریتم‌های متقارن موجود در .NET Framework عبارت‌اند از: DES[3]، 3DES[4]، RC2[5] و Rijndael[6].

عملیاتی که برای رمزگذاری و رمزگشایی باید انجام داد، کمی پیچیده‌تر از عملیاتی است که برای درهم‌برهم کردن، انجام می‌دهید. کلاس‌های موجود در .NET Framework. برای کار با جریان داده‌ها، بهینه‌سازی شده‌اند، بنابراین باید کارهای بیشتری، با تبدیل داده‌ها انجام دهید. همچنین باید یک کلید (Key) و یک بردار مقداردهی (IV[7])، برای رمزگذاری و رمزگشایی تعریف کنید. IV، وظیفه دارد مقدارهای رمزگذاری شده برای یک ترتیب بیتی را محاسبه کند. IV به‌این دلیل استفاده می‌شود که این

---

[1] Digital Signature Algorithm

[2] Rivest-Shamir-Adleman– Ronald Rivest ،Adi Shamir & Leonard Adleman(اسامی این مخترعان)

[3] Data Encryption Stan-dard

[4] Triple DES

[5] Ron's Code یا Rivest's Cipher

[6] John Daemen و Vincent Rijman- اسامی مخترعان آن

[7] initialization vector

قبیل مقادیر، در ابتدا وجود ندارند. در عمل، IV و Key می‌توانند به‌نام یک آرایه‌ی بایتی[1]، نمایش داده شوند که در رمزگذاری DES، 64 بیت (8 بایت) طول دارند.

برای کسب اطلاعات بیشتر در مورد روش‌های متفاوت رمزگذاری، به آدرس زیر مراجعه نمایید:

http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

گام‌های لازم برای رمزگذاری یک رشته به یک رشته‌ی رمزگذاری شده، به‌شرح زیر می‌باشد:

- تبدیل رشته‌ی منبع، به یک آرایه‌ی بایتی،
- مقداردهی یک کلاس الگوریتم رمزگذاری،
- استفاده از کلاس الگوریتم رمزگذاری، برای تولید یک شیء رمزگذاری‌کننده (پشتیبانی از رابط ICryptoTransform). این کار، مقادیر key و IV را نیاز دارد،
- استفاده از شیء رمزگذاری‌کننده، برای مقداردهی یک جریان پنهانی (شیء CryptoStream). همچنین این جریان، نیاز دارد بداند که شما در حال رمزگذاری داده‌ها هستید و برای نوشتن داده‌های رمزگذاری شده درون آن، به یک جریان هدف نیاز دارید،
- استفاده از جریان پنهانی برای نوشتن داده‌های رمزگذاری شده در یک جریان حافظه‌ی هدف، با استفاده از آرایه‌ی بایتی منبع که پیش از این ایجاد شد،
- استخراج داده‌های بایتی ذخیره‌شده در جریان،
- تبدیل داده‌های بایتی در یک رشته.

برای رمزگشایی کردن، عملیات زیر انجام می‌گیرد:

- تبدیل رشته‌ی منبع به یک آرایه‌ی بایتی،
- پرکردن جریان حافظه با محتوای آرایه‌ی بایتی،
- مقداردهی یک کلاس الگوریتم رمزگذار،
- استفاده از شیء رمزگذارکننده، برای مقداردهی جریان پنهانی (شیء CryptoStream). همچنین این جریان نیاز دارد بداند که شما در حال رمزگذاری داده‌ها هستید، و برای خواندن داده‌های رمزگذاری شده از آن، به یک جریان هدف نیاز دارید،

[1] byte array

- استفاده از جریان پنهانی، برای خواندن داده‌های رمزگشایی شده (می‌توانید از متد StreamReader.ReadToEnd برای گرفتن نتیجه‌ها به عنوان یک رشته استفاده کنید).

در پروژه‌ی Balloonshop، از DES استفاده می‌شود ولی کد موجود در StringEncryptor می‌توانست با کدی جایگزین شود تا برای هرکدام از الگوریتم‌هایی که پیش از این مشخص شدند، قابل استفاده باشند.

تمرین : پیاده‌سازی کلاس StringEncryptor

1. کلاس جدیدی با نام StringEncryptorException، به پوشه‌ی SecurityLib افزوده و کد زیر را به آن اضافه نمایید:

```
using System;
 using System.Collections.Generic;
 using System.Text;
namespace SecurityLib
 {
  public class StringEncryptorException : Exception
  {
   public StringEncryptorException(string message)
    : base(message)
   {
   }
  }
 }
```

2. کلاس جدید دیگری با نام StringEncryptor به پوشه‌ی SecurityLib اضافه کنید و کدهای زیر را به آن بیافزایید:

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

 namespace SecurityLib
 {
  public static class StringEncryptor
  {
   public static string Encrypt(string sourceData)
   {
    // set key and initialization vector values
    byte[] key = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
    byte[] iv = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
    try
    {
     // convert data to byte array
     byte[] sourceDataBytes =
      System.Text.ASCIIEncoding.ASCII.GetBytes(sourceData);

     // get target memory stream
```

```
    MemoryStream tempStream = new MemoryStream();

    // get encryptor and encryption stream
    DESCryptoServiceProvider encryptor =
      new DESCryptoServiceProvider();
    CryptoStream encryptionStream =
      new CryptoStream(tempStream,
        encryptor.CreateEncryptor(key, iv),
        CryptoStreamMode.Write);
            // encrypt data
            encryptionStream.Write(sourceDataBytes, 0,
              sourceDataBytes.Length);
            encryptionStream.FlushFinalBlock();
            // put data into byte array
            byte[] encryptedDataBytes = tempStream.GetBuffer();

            // convert encrypted data into string
            return Convert.ToBase64String(encryptedDataBytes, 0,
             (int)tempStream.Length);
          }
          catch
          {
            throw new StringEncryptorException(
              "Unable to encrypt data.");
          }
        }

        public static string Decrypt(string sourceData)
        {
          // set key and initialization vector values
          byte[] key = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
          byte[] iv = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
          try
          {
            // convert data to byte array
            byte[] encryptedDataBytes =
              Convert.FromBase64String(sourceData);

            // get source memory stream and fill it
            MemoryStream tempStream =
              new MemoryStream(encryptedDataBytes, 0,
                encryptedDataBytes.Length);

            // get decryptor and decryption stream
            DESCryptoServiceProvider decryptor =
              new DESCryptoServiceProvider();
            CryptoStream decryptionStream =
              new CryptoStream(tempStream,
                decryptor.CreateDecryptor(key, iv),
                CryptoStreamMode.Read);
    // decrypt data
    StreamReader allDataReader =
      new StreamReader(decryptionStream);
    return allDataReader.ReadToEnd();
  }
  catch
  {
    throw new StringEncryptorException(
      "Unable to decrypt data.");
```

```
    }
  }
 }
}
```

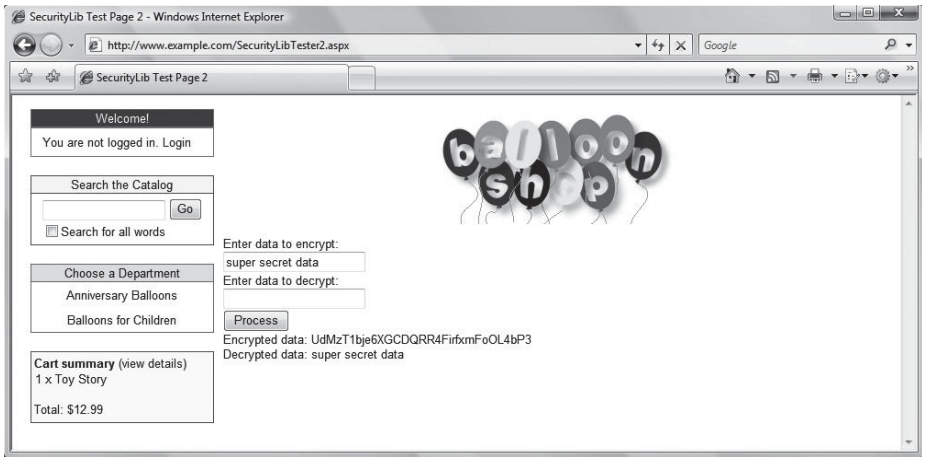
3. یک صفحه‌ی جدید با نام SecurityLibTester2.aspx به ریشه‌ی پروژه‌ی BalloonShop اضافه کنید و کد زیر را به آن بیافزایید:

```
<%@ Page Title="SecurityLib Test Page 2" Language="C#" MasterPageFile=
 "~/BalloonShop.master" AutoEventWireup="true"
  CodeFile="SecurityLibTester2.aspx.cs" Inherits="SecurityLibTester2" %>

 <asp:Content ID="Content1" ContentPlaceHolderID="head" runat="Server">
 </asp:Content>
 <asp:Content ID="Content2" ContentPlaceHolderID="ContentPlace
 Holder1" runat="Server">
  Enter data to encrypt:
  <br />
  <asp:TextBox ID="encryptBox" runat="server" />
  <br />
  Enter data to decrypt:
  <br />
  <asp:TextBox ID="decryptBox" runat="server" />
  <br />
  <asp:Button ID="processButton" runat="server" Text="Process"
 OnClick="processButton_Click" />
  <br />
  <asp:Label ID="result" runat="server" />
 </asp:Content>
```

4. کد موجود در SecurityLibTester2.aspx.cs را مانند زیر تغییر دهید:

```
using System;
using System.Text;
using SecurityLib;
        public partial class SecurityLibTester2 : System.Web.UI.Page
        {
          protected void processButton_Click(object sender, EventArgs e)
          {
            string stringToEncrypt = encryptBox.Text;
            string stringToDecrypt = decryptBox.Text;
            string encryptedString =
            StringEncryptor.Encrypt(stringToEncrypt);
            if (stringToDecrypt == "")
            {
              stringToDecrypt = encryptedString;
            }
            string decryptedString =
            StringEncryptor.Decrypt(stringToDecrypt);

            StringBuilder sb = new StringBuilder();
            sb.Append("Encrypted data: ");
            sb.Append(encryptedString);
            sb.Append("<br />Decrypted data: ");
            sb.Append(decryptedString);
            result.Text = sb.ToString();
          }
        }
```

5. SecurityLibTester2.aspx را بارگذاری کنید. برای رمزگذاری، یک رشته در نخستین تکست باکس[1] وارد (تکست باکس دومی را فعلا خالی رها کنید) و سپس بر روی دکمه‌ی Process کلیک کنید. نتیجه در شکل 2-16 نمایش داده شده است.



شکل 2-16. نتیجه‌ی رمزگذاری رشته

روش کار: پیاده‌سازی کلاس StringEncryptor

کلاس StringEncryptor، دارای دو متد اشتراک‌شده می‌باشد (Encrypt و Decrypt) که برای رمزگذاری و رمزگشایی استفاده می‌گردند (هرکدام از این دو متد را در ادامه بررسی می‌کنیم).

متد Encrypt، با تعریف دو آرایه‌ی بایتی برای key و IV استفاده شده در رمزگذاری، کار را آغاز می‌کند:

```
public static string Encrypt(string sourceData)
{
  // set key and initialization vector values
  byte[] key = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
  byte[] iv = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
```

در اینجا، هر دو آرایه با مقدارهای موقتی تنظیم شدند. این آرایه‌ها، بسته به کلیدی که می‌خواهید داشته باشید به‌راحتی می‌توانستند هر مقدار دیگری داشته باشند. در نظر داشته باشید که این مقدارها را در زمانی‌که متد فراخوانی می‌شود، مقداردهی می‌کنید (به‌جای استفاده از مقدارهای ثابت).

[1] text box

این کار به این دلیل انجام می‌شود که آرایه‌ی IV، به‌عنوان بخشی از فرایند رمزگذاری تغییر پیدا می‌کند به‌همین دلیل اگر آن را دوباره مقداردهی نکنید، آن مقدارها، متفاوت خواهد بود.

این در عمل به این مفهوم است که تعداد کمی از نخستین بایت‌های داده‌های رمزگشایی‌شده، دچار دست‌کاری شوند.

به‌همین دلیل باید مقدارهای خودتان را استفاده کنید، نه مقدارهای موقتی که پیش از این در کد استفاده گردید. برای ایجاد این قبیل مقدارها، می‌توانید از کلاس‌ها و متدهای موجود در فضای‌نام System.Security.Cryptography استفاده کنید (یا می‌توانید مقدارهای تصادفی، درج کنید).

کد رمزگذاری، در یک بلوک try...catch احاطه شده است (برای حالتی‌که خطا رخ دهد). کدهای موجود در این بلوک، گام‌های تشریح شده در مرحله‌ی پیش را دنبال می‌کند (با تبدیل رشته‌ی منبع به یک آرایه‌ی بایتی، آغاز می‌شود):

```
try
{
  // convert data to byte array
  byte[] sourceDataBytes =
    System.Text.ASCIIEncoding.ASCII.GetBytes(sourceData);
```

سپس یک شیء MemoryStream مقداردهی می‌گردد که برای ذخیره‌کردن داده‌ی رمزگذاری شده استفاده می‌شود:

```
// get target memory stream
    MemoryStream tempStream = new MemoryStream();
```

اکنون شیء رمزگذاری کننده را دارید (در این مورد، یک نمونه از کلاس DESCryptoServiceProvider) و از آن (به همراه key و IV که پیش از این تولید شد) برای تولید یک شیء CryptoStream استفاده می‌کنید:

```
// get target memory stream
MemoryStream tempStream = new MemoryStream();

// get encryptor and encryption stream
DESCryptoServiceProvider encryptor =
  new DESCryptoServiceProvider();
CryptoStream encryptionStream =
  new CryptoStream(tempStream,
    encryptor.CreateEncryptor(key, iv),
    CryptoStreamMode.Write);
```

بخش بعدی کد، رمزگذاری واقعی را انجام می‌دهد: نتیجه‌ها آرایه‌ی بایتی را در MemoryStream (که پیش از این ایجاد شده بود) می‌نویسد:

```
// encrypt data
encryptionStream.Write(sourceDataBytes, 0,
  sourceDataBytes.Length);
encryptionStream.FlushFinalBlock();
```

در این بخش، فراخوانی FlushFinalBlock ضروری است. بدون این فراخوانی، داده‌هایی که نوشته نشدند ممکن است از بافر CryptoStream حذف شود.

سپس داده‌ها را از MemoryStream دریافت کرده و آنها را در یک آرایه‌ی بایتی قرار می‌دهد:

```
// put data into byte array
        byte[] encryptedDataBytes = tempStream.GetBuffer();
```

در پایان، نتایج آرایه‌ی بایتی را به یک رشته تبدیل کرده و آن را برمی‌گرداند:

```
// convert encrypted data into string
         return Convert.ToBase64String(encryptedDataBytes, 0,
          (int)tempStream.Length);
```

اگر در طی این فرایند، خطایی رخ دهد یک استثنا StringEncryptorException رخ می‌دهد:

```
       catch
       {
         throw new StringEncryptorException(
           "Unable to encrypt data.");
       }
     }
```

متد Decrypt، خیلی همانند Encrypt است. با مقداردهی ابتدایی key و IV (پیش از انتقال به یک بلوک Try...Catch) شروع و رشته‌ی منبع را به آرایه‌ی بایتی تبدیل می‌کند:

```
     public static string Decrypt(string sourceData)
     {
       // set key and initialization vector values
       byte[] key = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
       byte[] iv = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
       try
       {
         // convert data to byte array
         byte[] encryptedDataBytes =
           Convert.FromBase64String(sourceData);
```

چون CryptoStream از یک جریان می‌خواند (به جای آنکه بر روی آن بنویسد)، در این لحظه به یک جریان نیاز دارید که با این آرایه‌ی بایتی منبع، پر شود:

```
  // get source memory stream and fill it
   MemoryStream tempStream =   new MemoryStream(encryptedDataBytes, 0,
encryptedDataBytes.Length);
```

کد بعدی نیز بسیار ساده است. از متد CreateDecryptor و مد CryptoStreamMode.Read برای مشخص‌کردن رمزگشایی استفاده می‌کند:

```
  // get decryptor and decryption stream
DESCryptoServiceProvider decryptor =
new DESCryptoServiceProvider();
CryptoStream decryptionStream =
new CryptoStream(tempStream,
decryptor.CreateDecryptor(key, iv),
CryptoStreamMode.Read);
```

سرانجام با استفاده از یک شیء StreamReader، داده‌های رمزگذاری‌شده را بیرون از CryptoStream دریافت می‌کنید (که به شکل دستی، به شما اجازه می‌دهد داده را دریافت کرده و به‌شکل مستقیم برای برگشت دادن، آن را در رشته قرار دهید). آخرین کاری که انجام می‌شود این است که وقتی اشتباهی رخ دهد، یک استثنا StringEncryptorException رخ می‌دهد:

```
// decrypt data
StreamReader allDataReader =
new StreamReader(decryptionStream);
return allDataReader.ReadToEnd();
}
Catch
{
  throw new StringEncryptorException(
"Unable to decrypt data.");
}
 }
```

کد مشتری برای این کلاس، به آسانی داده‌ها را رمزگذاری و رمزگشایی می‌کند. از آنجایی‌که کد این کلاس بسیار ساده است، بنابراین نیازی به توضیح جزئیات آن نیست. اکنون که کد کلاس StringEncryptor را دارید، آخرین گام در ایجاد کتابخانه‌ی SecureLib، اضافه‌کردن کلاس SecureCard می‌باشد.

تمرین : پیاده‌سازی کلاس SecureCard

1. یک کلاس جدید با کدهای زیر به‌نام SecureCardException.cs را به پوشه‌ی SecurityLib اضافه کنید:

```
using System;

namespace SecurityLib
{
  public class SecureCardException : Exception
  {
    public SecureCardException(string message)
      : base(message)
    {
    }
  }
}
```

2. فایل جدید دیگری به‌نام SecureCard.cs را با کدهای زیر به پوشه‌ی SecurityLib اضافه کنید:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;

namespace SecurityLib
{
  public class SecureCard
```

```
  {
    private bool isDecrypted = false;
    private bool isEncrypted = false;
    private string cardHolder;
    private string cardNumber;
    private string issueDate;
    private string expiryDate;
    private string issueNumber;
    private string cardType;
    private string encryptedData;
    private XmlDocument xmlCardData;
public SecureCard(string newEncryptedData)
  {
    // constructor for use with encrypted data
    encryptedData = newEncryptedData;
    DecryptData();
  }

  public SecureCard(string newCardHolder,
    string newCardNumber, string newIssueDate,
    string newExpiryDate, string newIssueNumber,
    string newCardType)
  {
    // constructor for use with decrypted data
    cardHolder = newCardHolder;
    cardNumber = newCardNumber;
    issueDate = newIssueDate;
    expiryDate = newExpiryDate;
    issueNumber = newIssueNumber;
    cardType = newCardType;
    EncryptData();
  }

  private void CreateXml()
  {
    // encode card details as XML document
    xmlCardData = new XmlDocument();
    XmlElement documentRoot =
      xmlCardData.CreateElement("CardDetails");
    XmlElement child;

    child = xmlCardData.CreateElement("CardHolder");
    child.InnerXml = cardHolder;
    documentRoot.AppendChild(child);

    child = xmlCardData.CreateElement("CardNumber");
    child.InnerXml = cardNumber;
    documentRoot.AppendChild(child);

    child = xmlCardData.CreateElement("IssueDate");
    child.InnerXml = issueDate;
    documentRoot.AppendChild(child);

    child = xmlCardData.CreateElement("ExpiryDate");
    child.InnerXml = expiryDate;
    documentRoot.AppendChild(child);
child = xmlCardData.CreateElement("IssueNumber");
            child.InnerXml = issueNumber;
            documentRoot.AppendChild(child);
```

```
            child = xmlCardData.CreateElement("CardType");
            child.InnerXml = cardType;
            documentRoot.AppendChild(child);
            xmlCardData.AppendChild(documentRoot);
          }

          private void ExtractXml()
          {
            // get card details out of XML document
            cardHolder =
              xmlCardData.GetElementsByTagName(
                "CardHolder").Item(0).InnerXml;
            cardNumber =
              xmlCardData.GetElementsByTagName(
                "CardNumber").Item(0).InnerXml;
            issueDate =
              xmlCardData.GetElementsByTagName(
                "IssueDate").Item(0).InnerXml;
            expiryDate =
              xmlCardData.GetElementsByTagName(
                "ExpiryDate").Item(0).InnerXml;
            issueNumber =
              xmlCardData.GetElementsByTagName(
                "IssueNumber").Item(0).InnerXml;
            cardType =
              xmlCardData.GetElementsByTagName(
                "CardType").Item(0).InnerXml;
          }

          private void EncryptData()
          {
            try
            {
              // put data into XML doc
              CreateXml();
              // encrypt data
              encryptedData =
                StringEncryptor.Encrypt(xmlCardData.OuterXml);
              // set encrypted flag
              isEncrypted = true;
            }
catch
    {
     throw new SecureCardException("Unable to encrypt data.");
    }
  }

  private void DecryptData()
  {
    try
    {
      // decrypt data
      xmlCardData = new XmlDocument();
      xmlCardData.InnerXml =
        StringEncryptor.Decrypt(encryptedData);
      // extract data from XML
      ExtractXml();
      // set decrypted flag
```

```
      isDecrypted = true;
    }
    catch
    {
     throw new SecureCardException("Unable to decrypt data.");
    }
   }

   public string CardHolder
   {
    get
    {
     if (isDecrypted)
     {
      return cardHolder;
     }
     else
     {
      throw new SecureCardException("Data not decrypted.");
     }
    }
   }

   public string CardNumber
   {
    get
    {
     if (isDecrypted)
     {
      return cardNumber;
     }
else
                {
                 throw new SecureCardException("Data not decrypted.");
                }
               }
              }

              public string CardNumberX
              {
               get
               {
                if (isDecrypted)
                {
                 return "XXXX-XXXX-XXXX-"
                    + cardNumber.Substring(cardNumber.Length - 4, 4);
                }
                else
                {
                 throw new SecureCardException("Data not decrypted.");
                }
               }
              }

              public string IssueDate
              {
               get
               {
                if (isDecrypted)
```

```
            {
              return issueDate;
            }
            else
            {
              throw new SecureCardException("Data not decrypted.");
            }
          }
        }

        public string ExpiryDate
        {
          get
          {
            if (isDecrypted)
            {
              return expiryDate;
            }
    else
    {
      throw new SecureCardException("Data not decrypted.");
    }
  }
}

public string IssueNumber
{
  get
  {
    if (isDecrypted)
    {
      return issueNumber;
    }
    else
    {
      throw new SecureCardException("Data not decrypted.");
    }
  }
}

public string CardType
{
  get
  {
    if (isDecrypted)
    {
      return cardType;
    }
    else
    {
      throw new SecureCardException("Data not decrypted.");
    }
  }
}

public string EncryptedData
{
  get
  {
```

```
    if (isEncrypted)
    {
      return encryptedData;
    }
            else
            {
              throw new SecureCardException("Data not decrypted.");
            }
          }
        }
      }
    }
```

3. یک صفحه‌ی وب جدید به نام SecurityLibTester3.aspx را با تنظیمات و کدهای زیر در ریشه‌ی BalloonShop ایجاد کنید:

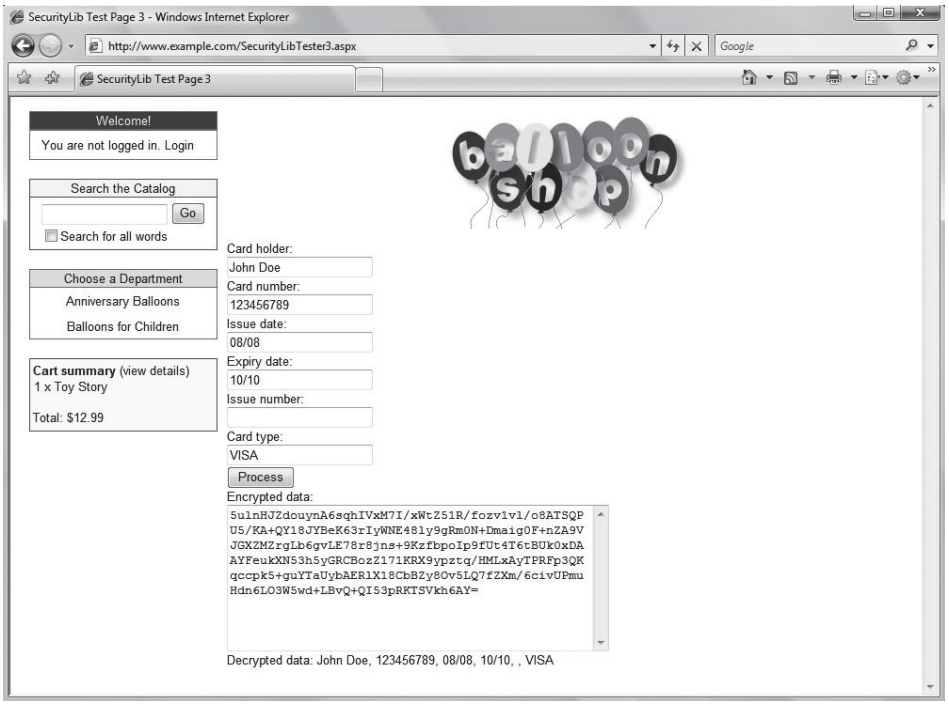
```
<%@ Page Title="SecurityLib Test Page 3" Language="C#" MasterPageFile=
"~/BalloonShop.master" AutoEventWireup="true" CodeFile="SecurityLib
Tester3.aspx.cs" Inherits="SecurityLibTester3" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
runat="Server">
  Card holder:<br />
  <asp:TextBox ID="cardHolderBox" runat="server" />
  <br />
  Card number:<br />
  <asp:TextBox ID="cardNumberBox" runat="server" />
  <br />
  Issue date:<br />
  <asp:TextBox ID="issueDateBox" runat="server" />
  <br />
  Expiry date:<br />
  <asp:TextBox ID="expiryDateBox" runat="server" />
  <br />
  Issue number:<br />
  <asp:TextBox ID="issueNumberBox" runat="server" />
  <br />
Card type:<br />
  <asp:TextBox ID="cardTypeBox" runat="server" />
  <br />
  <asp:Button ID="processButton" runat="server" Text="Process"
 OnClick="processButton_Click" />
  <br />
  <asp:Label ID="result" runat="server" />
 </asp:Content>
```

4. کد موجود در SecurityLibTester3.aspx.cs را همانند زیر تغییر دهید:

```
using System;
 using System.Text;
 using SecurityLib;

 public partial class SecurityLibTester3 : System.Web.UI.Page
 {
  protected void processButton_Click(object sender, EventArgs e)
  {
   SecureCard encryptedCard =
```

```
    new SecureCard(cardHolderBox.Text, cardNumberBox.Text,
      issueDateBox.Text, expiryDateBox.Text,issueNumberBox.Text,
      cardTypeBox.Text);
    string encryptedData = encryptedCard.EncryptedData;
    SecureCard decryptedCard = new SecureCard(encryptedData);
    string decryptedData = string.Format(
      "{0}, {1}, {2}, {3}, {4}, {5}",
      decryptedCard.CardHolder, decryptedCard.CardNumber,
      decryptedCard.IssueDate, decryptedCard.ExpiryDate,
      decryptedCard.IssueNumber, decryptedCard.CardType);
    StringBuilder sb = new StringBuilder();
    sb.Append("Encrypted data:<br />");
    sb.Append("<textarea style=\"width:400px;height:150px;\">");
    sb.Append(encryptedData);
    sb.Append("</textarea><br />Decrypted data:");
    sb.Append(decryptedData);
    result.Text = sb.ToString();
  }
}
```

5. SecurityLibTester3.aspx را بارگذاری، جزئیات کارت را برای رمزگذاری وارد، و بر روی دکمه‌ی **Process** کلیک کنید. نتیجه، در شکل 3 – 16 نشان داده شده است.



شکل 3 – 16. نتیجه‌ی رمز گذاری کارت اعتباری