**Câu 1: Giới thiệu về mã hóa đối xứng bất đối xứng**

* 1. **Khái Niệm:**

1. **Mã hóa đối xứng :**

Đây được cho là kỹ thuật mã hóa đơn giản và được sử dụng phổ biến nhất, với một số đặc điểm nổi bật như:

* Do thuật toán mã hóa đối xứng ít phức tạp hơn và có thể thực thi nhanh hơn, đây là kỹ thuật được đặc biệt ưa thích trong các hoạt động truyền tải dữ liệu hàng loạt.
* Văn bản gốc được mã hóa bằng một key trước khi gửi đi, và chính key này cũng sẽ được người nhận sử dụng để giải mã dữ liệu.
* Một số thuật toán mã hóa đối xứng được sử dụng phổ biến nhất bao gồm AES-128, AES-192 và AES-256.

1. **Mã hóa bất đối xứng:**

Đây là loại hình mã hóa ra đời sau mã hóa đối xứng và còn được gọi là công nghệ mã hóa public-key:

* Mã hóa bất đối xứng được cho là an toàn hơn mã hóa đối xứng vì nó sử dụng 2 key riêng biệt cho 2 quy trình mã hóa và giải mã.
* Public key được sử dụng để mã hóa sẽ được công khai, nhưng private key để giải mã là hoàn toàn bí mật.
* Phương pháp mã hóa này được sử dụng trong các giao tiếp hàng ngày qua internet.
* Khi một tin nhắn được mã hóa bằng public key, nó chỉ có thể được giải mã bằng private key. Tuy nhiên, khi một tin nhắn được mã hóa bằng private key, nó có thể được giải mã bằng public key.
* Chứng chỉ kỹ thuật số trong mô hình máy khách-máy chủ có thể được sử dụng để tìm thấy các public key.
* Điểm hạn chế của mã hóa bất đối xứng là mất nhiều thời gian thực hiện hơn so với mã hóa đối xứng.
* Các kỹ thuật mã hóa bất đối xứng phổ biến bao gồm RSA, DSA và PKCS.
  1. **Vẽ sơ đồ**
  2. **Sơ đồ đối xứng:**

A screenshot of a phone

Description automatically generated

* 1. **Sơ đồ bất đối xứng:**

A screenshot of a computer

Description automatically generated

* 1. **So Sánh:**

Mã hóa bất đối xứng và mã hóa đối xứng có những đặc điểm khác nhau và phù hợp với những mục đích khác nhau:

* Mã hóa bất đối xứng:
  + Sử dụng một cặp khóa: khóa chung và khóa riêng.
  + Khóa chung được sử dụng để mã hóa, khóa riêng để giải mã.
  + Thích hợp cho việc trao đổi chìa khóa, kỹ thuật số
  + Chậm và tốn kém về mặt tính toán so với mã hóa đối xứng.
* Mã hóa đối xứng:
  + Sử dụng một
  + Thích hợp để mã hóa dữ liệu số lượng lớn và nhanh hơn để mã hóa lượng lớn dữ liệu.
  + Yêu cầu cơ chế trao đổi khóa an toàn để chia sẻ khóa giữa người gửi và người nhận.
  + Hiệu quả hơn và nhanh hơn so với mã hóa bất đối xứng.

**1.4**

using System;

using System.Security.Cryptography;

using System.Text;

public class AsymmetricEncryptionExample

{

public static void Main()

{

// Create RSA encryption provider

using (RSA rsa = RSA.Create())

{

// Get the public and private keys

RSAParameters publicKey = rsa.ExportParameters(false); // false for public key

RSAParameters privateKey = rsa.ExportParameters(true); // true for private key

// Convert sample data to bytes

string originalData = "Hoang Dep Zai";

byte[] dataToEncrypt = Encoding.UTF8.GetBytes(originalData);

// Encrypt data using the public key

byte[] encryptedData = EncryptData(dataToEncrypt, publicKey);

// Decrypt data using the private key

byte[] decryptedData = DecryptData(encryptedData, privateKey);

// Convert decrypted bytes back to string

string decryptedString = Encoding.UTF8.GetString(decryptedData);

Console.WriteLine("Original: " + originalData);

Console.WriteLine("Encrypted: " + Convert.ToBase64String(encryptedData));

Console.WriteLine("Decrypted: " + decryptedString);

}

}

// Method to encrypt data using the public key

public static byte[] EncryptData(byte[] data, RSAParameters publicKey)

{

using (RSA rsa = RSA.Create())

{

rsa.ImportParameters(publicKey);

return rsa.Encrypt(data, RSAEncryptionPadding.Pkcs1); // Changed padding mode to Pkcs1

}

}

// Method to decrypt data using the private key

public static byte[] DecryptData(byte[] data, RSAParameters privateKey)

{

using (RSA rsa = RSA.Create())

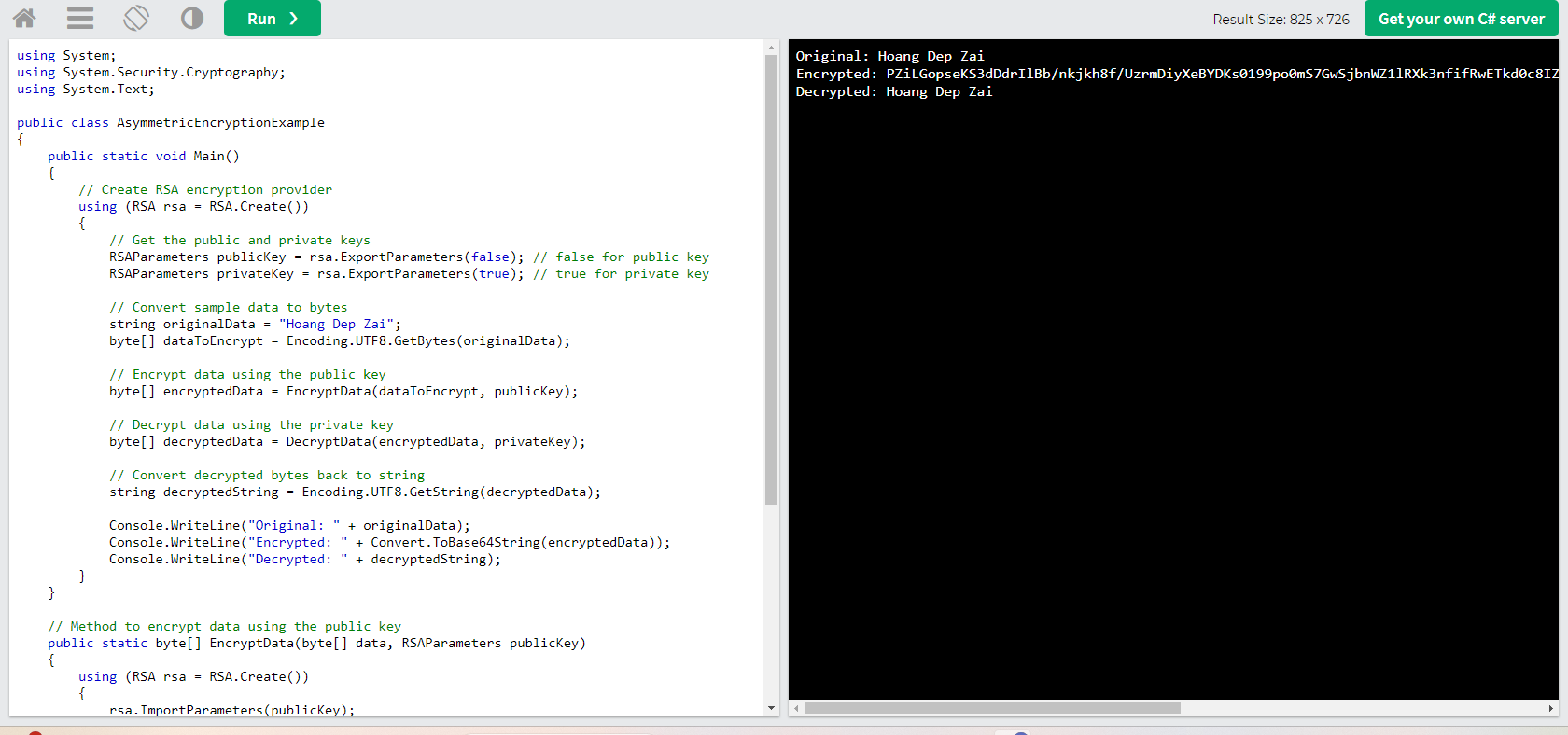
{

rsa.ImportParameters(privateKey);

return rsa.Decrypt(data, RSAEncryptionPadding.Pkcs1); // Changed padding mode to Pkcs1

}

}

}

**Câu 2: Lỗ hổng bảo mật**

* 1. **Khái Niệm:**

Lỗ hổng bảo mật là điểm yếu hoặc lỗ hổng trong hệ thống hoặc phần mềm mà kẻ tấn công có thể khai thác để xâm phạm tính bảo mật, tính toàn vẹn hoặc tính khả dụng của hệ thống hoặc dữ liệu của hệ thống. Lỗ hổng có thể tồn tại trong nhiều thành phần khác nhau của hệ thống, bao gồm phần mềm, phần cứng, cấu hình mạng và hoạt động của người dùng. Việc khai thác lỗ hổng có thể dẫn đến truy cập trái phép, vi phạm dữ liệu, tấn công từ chối dịch vụ và các sự cố bảo mật khác.

* 1. **SQL insert và cách phòng tránh:**
     1. **SQL insert:**

SQL insert là một lỗ hổng phổ biến và nghiêm trọng xảy ra khi kẻ tấn công có thể thao túng thông tin đầu vào của người dùng được sử dụng để xây dựng các truy vấn SQL. Kẻ tấn công có thể tiêm mã SQL độc hại vào truy vấn, điều này có thể dẫn đến truy cập hoặc thao túng cơ sở dữ liệu trái phép.

* + 1. **Cách phòng ngừa:**
* Sử dụng truy vấn được tham số hóa hoặc câu lệnh được chuẩn bị sẵn: Thay vì xây dựng động các truy vấn SQL bằng cách nối dữ liệu đầu vào của người dùng, hãy sử dụng truy vấn được tham số hóa hoặc câu lệnh được chuẩn bị sẵn. Các cơ chế này tách mã SQL khỏi đầu vào của người dùng, ngăn chặn hiệu quả việc tiêm mã độc.
* Xác thực và khử trùng đầu vào: Xác thực và vệ sinh đầu vào của người dùng để đảm bảo nó tuân thủ các định dạng mong đợi và không chứa bất kỳ ký tự hoặc mã SQL độc hại nào. Điều này có thể liên quan đến các kỹ thuật như xác thực độ dài đầu vào, xác thực loại dữ liệu và sử dụng danh sách trắng hoặc biểu thức chính quy để lọc ra các ký tự có khả năng gây hại.
* Nguyên tắc đặc quyền tối thiểu: Đảm bảo rằng các tài khoản cơ sở dữ liệu được ứng dụng sử dụng có các đặc quyền và quyền truy cập hạn chế. Sử dụng các tài khoản riêng biệt có quyền hạn chế cho các hoạt động khác nhau (ví dụ: một tài khoản dành cho quyền truy cập chỉ đọc và tài khoản khác có quyền truy cập ghi).
* Triển khai các biện pháp kiểm soát truy cập mạnh mẽ: Thực thi các cơ chế xác thực và ủy quyền phù hợp để đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập và sửa đổi cơ sở dữ liệu. Sử dụng mật khẩu mạnh và duy nhất cho các tài khoản cơ sở dữ liệu và thường xuyên cập nhật chúng.
* Thường xuyên cập nhật và vá lỗi phần mềm: Luôn cập nhật hệ thống quản lý cơ sở dữ liệu của bạn và phần mềm liên quan với các bản vá và cập nhật bảo mật mới nhất. Các lỗ hổng bảo mật thường được các nhà cung cấp phần mềm phát hiện và vá lỗi, do đó, việc cập nhật sẽ giúp bảo vệ khỏi các lỗ hổng đã biết.
* Kiểm tra bảo mật và xem xét mã: Thực hiện kiểm tra bảo mật thường xuyên, bao gồm đánh giá lỗ hổng bảo mật và kiểm tra thâm nhập, để xác định và khắc phục mọi lỗ hổng SQL Insert. Tiến hành đánh giá mã để đảm bảo tuân thủ các phương pháp mã hóa an toàn.

Bằng cách triển khai các biện pháp phòng ngừa này, nguy cơ xảy ra lỗ hổng SQL Insert có thể giảm đáng kể, từ đó nâng cao tính bảo mật và tính toàn vẹn của các ứng dụng dựa trên cơ sở dữ liệu.

**Câu 3: Mã hóa mật khẩu (SHA-256) trong C#**

using System;

using System.Security.Cryptography;

using System.Text;

public class PasswordEncryptionExample

{

public static void Main()

{

string password = "HoangDaiKa"; // Replace with the user's password

string hashedPassword = ComputeSHA256Hash(password);

Console.WriteLine("Original Password: " + password);

Console.WriteLine("Hashed Password: " + hashedPassword);

}

public static string ComputeSHA256Hash(string input)

{

using (SHA256 sha256 = SHA256.Create())

{

byte[] inputBytes = Encoding.UTF8.GetBytes(input);

byte[] hashBytes = sha256.ComputeHash(inputBytes);

StringBuilder builder = new StringBuilder();

for (int i = 0; i < hashBytes.Length; i++)

{

builder.Append(hashBytes[i].ToString("x2"));

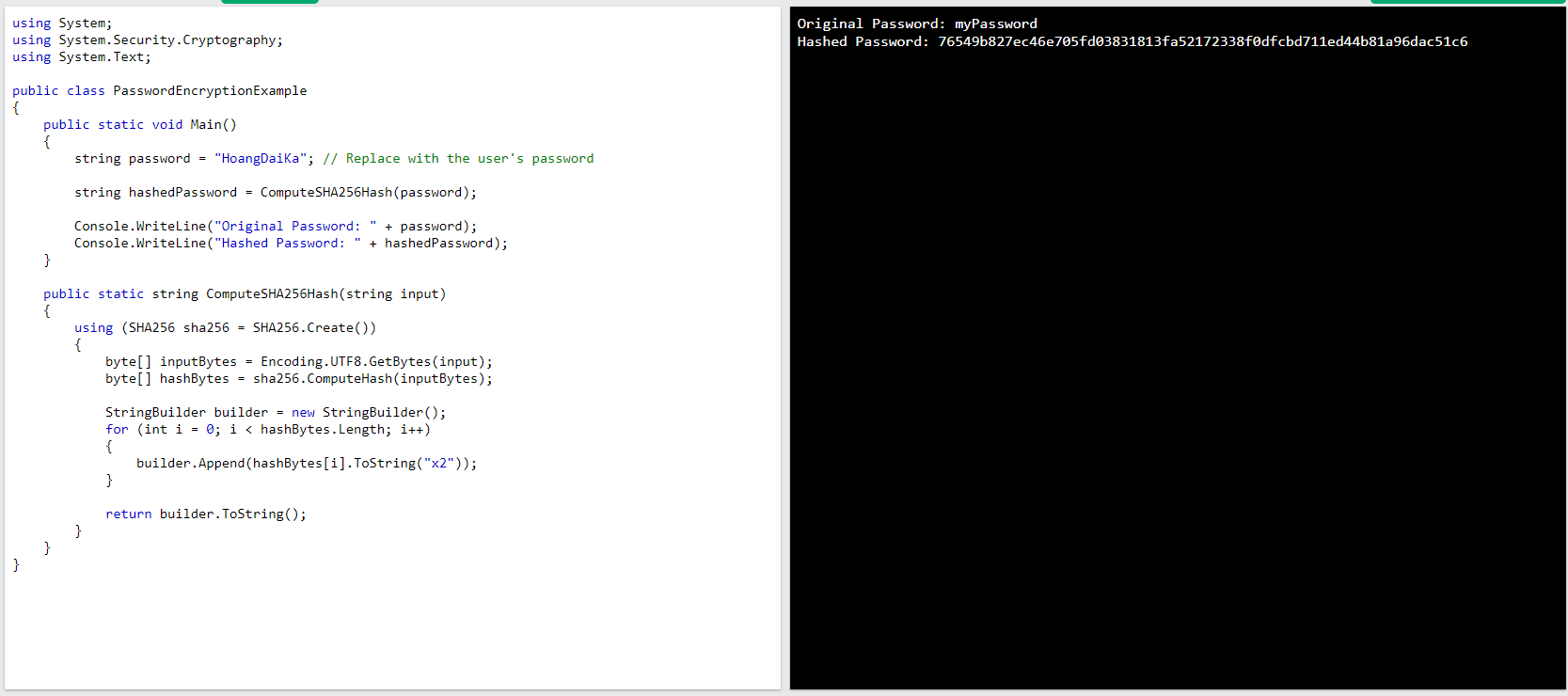
}

return builder.ToString();

}

}

}

****