

Session: **16**

Encrypting and Decrypting Data

- ◆ Explain symmetric encryption.
- ◆ Explain asymmetric encryption.
- ◆ List the various types in the `System.Security.Cryptography` namespace that supports symmetric and asymmetric encryptions.

For Aptech Centre Use Only

Introducing Cryptography and Encryption 1-2

- ◆ All organizations need to handle sensitive data.
- ◆ This data can be either present in storage or may be exchanged between different entities within and outside the organization over a network.

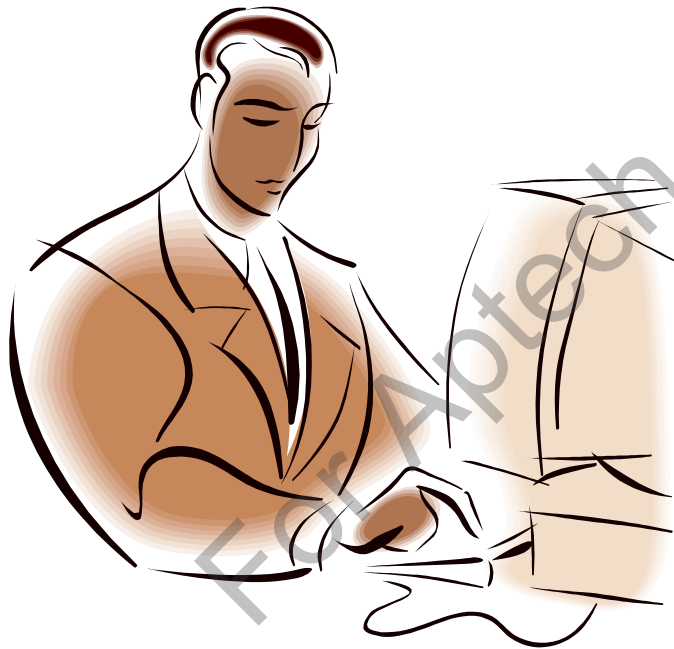
Example

- ◆ Steve is the CEO of a company.
- ◆ Steve while travelling needs to access performance appraisal reports of the top management of the company.
- ◆ These reports contain data that are confidential and are stored in the company's server.
- ◆ Such data is often prone to misuse either intentionally with malicious intent or unintentionally.
- ◆ To avoid such misuse, there should be some security mechanism that can ensure confidentiality and integrity of data.



Introducing Cryptography and Encryption 2-2

- ◆ Cryptography is a security mechanism to ensure data confidentiality and integrity.
- ◆ The commonly used ways to secure sensitive data is through encryption.



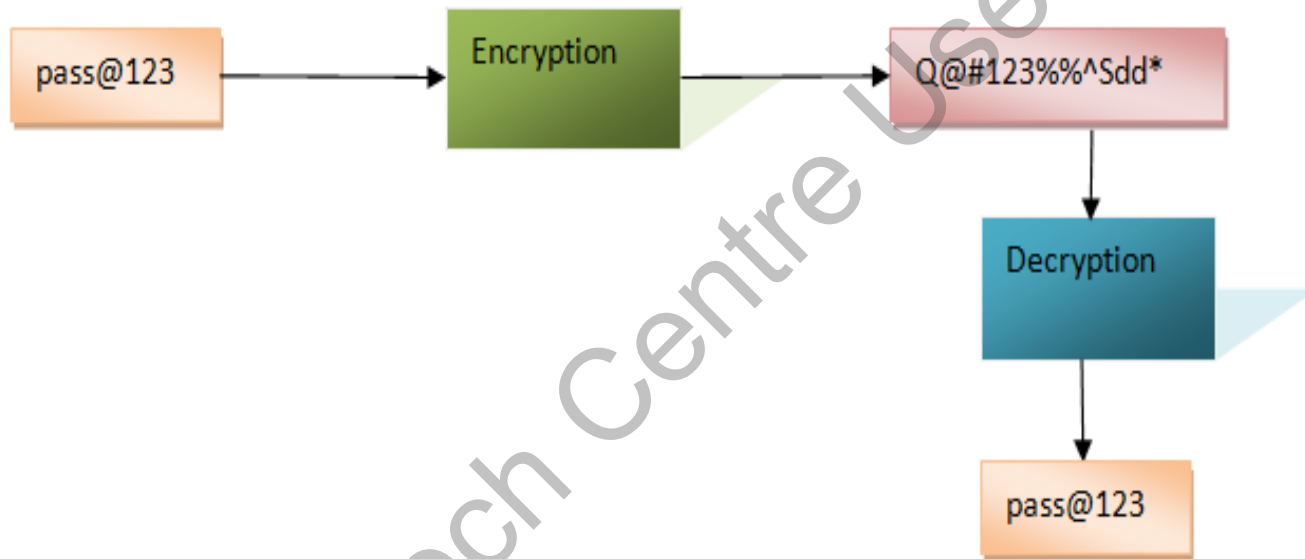
Security

- ◆ The primary objective of cryptography is to secure data exchanged between entities, each of which can be a person or an application.

Example

- ◆ Consider a scenario:
 - ◆ User A transmits sensitive data to User B.
 - ◆ The transmission needs to be confidential to ensure that even if a third party obtains the data, the data is incomprehensible.
 - ◆ User B before using the data must be sure about the integrity of the data, which means that User B must be sure that the data has not been modified in transit.
 - ◆ User B must be able to authenticate that the data is actually been sent by User A.
 - ◆ After sending the data, User A must not be able to deny sending the data to User B.
- ◆ **Encryption:**
 - ◆ Is a cryptographic technique that ensures data confidentiality.
 - ◆ Converts data in plain text to cipher (secretly coded) text.
- ◆ The opposite process of encryption is called decryption, which is a process that converts the encrypted cipher text back to the original plain text.

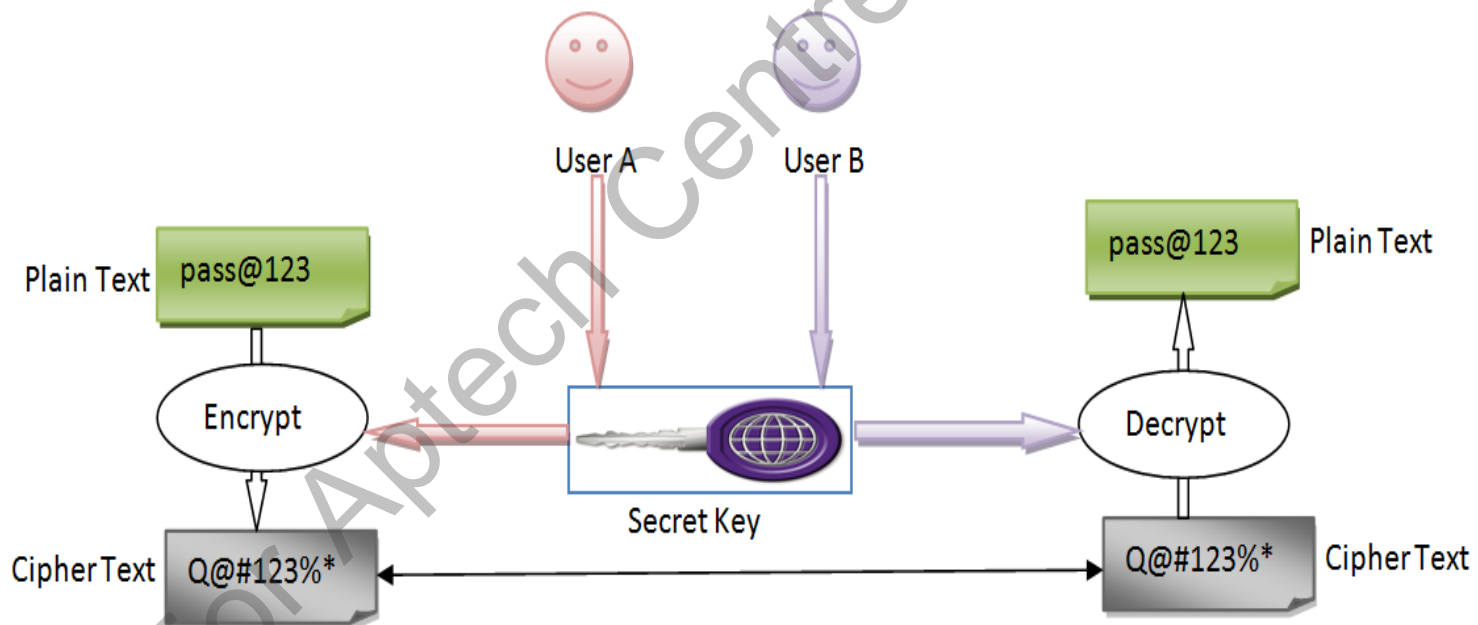
- ◆ The following figure shows the process of encryption and decryption of a password as an example.



- ◆ In the figure:
 - ◆ The plain text, pass@123 is encrypted to a cipher text.
 - ◆ The cipher text is decrypted back to the original plain text.

Symmetric Encryption 1-2

- ◆ Symmetric encryption, or secret key encryption, uses a single key, known as the secret key both to encrypt and decrypt data.
- ◆ The following figure shows the symmetric encryption and decryption process.

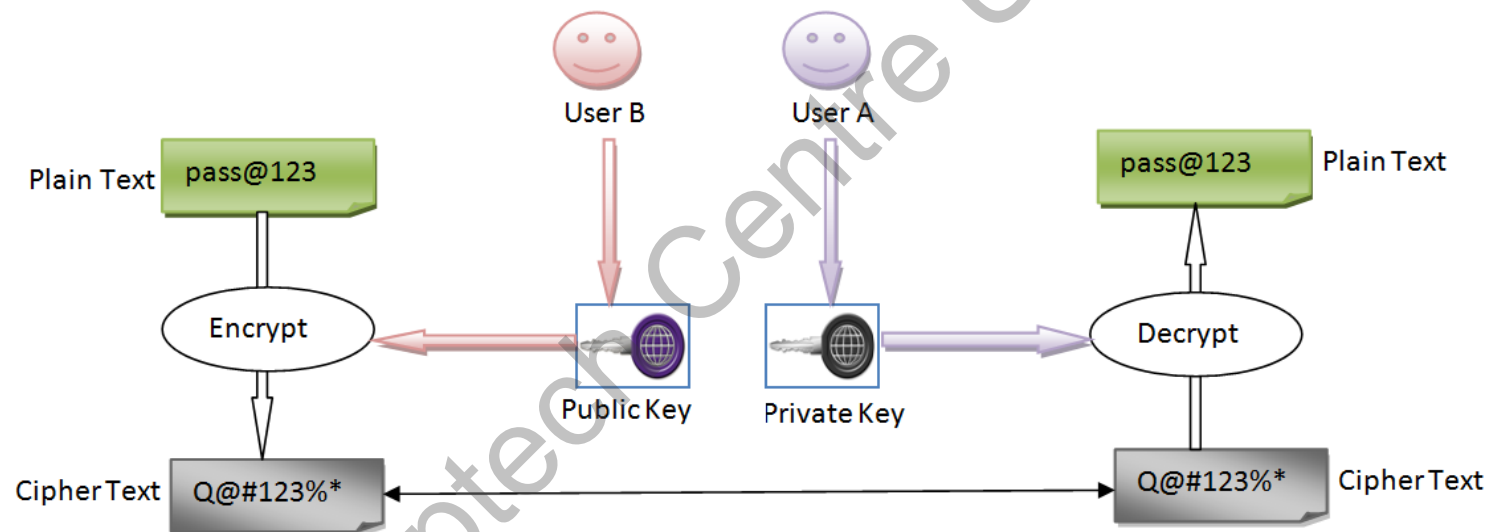


- ◆ The following steps outline an example usage of symmetric encryption:
 - ◆ User A uses a secret key to encrypt a plain text to cipher text.
 - ◆ User A shares the cipher text and the secret key with User B.
 - ◆ User B uses the secret key to decrypt the cipher text back to the original plain text.

For Aptech Centre Use Only

Asymmetric Encryption 1-2

- ◆ Asymmetric encryption uses a pair of public and private key to encrypt and decrypt data.
- ◆ The following figure shows the symmetric encryption and decryption process.

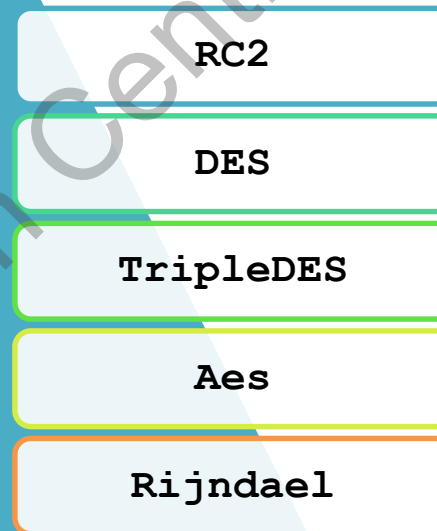


- ◆ The following steps outline an example usage of asymmetric encryption:
 - ◆ User A generates a public and private key pair.
 - ◆ User A shares the public key with User B.
 - ◆ User B uses the public key to encrypt a plain text to cipher text.
 - ◆ User A uses the private key to decrypt the cipher text back to the original plain text.

For Apteck Centre Use Only

Symmetric Encryption Algorithms 1-6

- ◆ The `System.Security.Cryptography` namespace provides the `SymmetricAlgorithm` base class for all symmetric algorithm classes.
- ◆ The derived classes of the `SymmetricAlgorithm` base class are as follows:



RC2

- ◆ Is an abstract base class for all classes that implement the RC2 algorithm.
- ◆ Is a proprietary algorithm developed by RSA Data Security, Inc in 1987.
- ◆ Supports key sizes ranging from 40 bits to 128 bits in 8-bit increments for encryption.
- ◆ Was designed for the old generation processors and currently have been replaced by more faster and secure algorithms.
- ◆ Derives the `RC2CryptoServiceProvider` class to provide an implementation of the RC2 algorithm.

DES

- ◆ Is an abstract base class for all classes that implement the Data Encryption Standard (DES) algorithm.
- ◆ Developed by IBM but as of today available as a U.S. Government Federal Information Processing Standard (FIPS 46-3).
- ◆ Works on the data to encrypt as blocks where each block is of 64 bits.
- ◆ Uses a key of 64 bits to perform the encryption. On account of its small key size DES encrypts data faster as compared to other asymmetric algorithms.
- ◆ Is prone to brute force security attacks because of its smaller key size.
- ◆ Derives the `DESCryptoServiceProvider` class to provide an implementation of the DES algorithm.

TripleDES

- ◆ Is an abstract base class for all the classes that implement the TripleDES algorithm.
- ◆ Is an enhancement to the DES algorithm for the purpose of making the DES algorithm more secured against security threats.
- ◆ Works on 64 bit blocks that is similar to the DES algorithm.
- ◆ Supports key sizes of 128 bits to 192 bits.
- ◆ Derives the `TripleDESCryptoServiceProvider` class to provide an implementation of the TripleDES algorithm.

Aes

- ◆ Is an abstract base class for all classes that implement the Advanced Encryption Standard (AES) algorithm.
- ◆ Is a successor of DES and is currently considered as one of the most secure algorithm.
- ◆ Is more efficient in encrypting large volume of data in the size of several gigabytes.
- ◆ Works on 128-bits blocks of data and key sizes of 128, 192, or 256 bits for encryption.
- ◆ Derives the `AesCryptoServiceProvider` and `AesManaged` classes to provide an implementation of the AES algorithm.

Rijndael

- ◆ Is an abstract base class for all the classes that implement the `Rijndael` algorithm.
- ◆ Is a superset of the `Aes` algorithm.
- ◆ Supports key sizes of 128, 192, or 256 bits similar to the `Aes` algorithm.
- ◆ Can have block sizes of 128, 192, or 256 bits, unlike `Aes`, which has a fixed block size of 128 bits.
- ◆ Provides the flexibility to select an appropriate block size based on the volume of data to encrypt by supporting different block sizes.
- ◆ Derives the `RijndaelManaged` class to provide an implementation of the `Rijndael` algorithm.

Asymmetric Encryption Algorithm 1-2

- ◆ The `System.Security.Cryptography` namespace provides the `AsymmetricAlgorithm` base class for all asymmetric algorithm classes.
- ◆ RSA is an abstract class that derives from the `AsymmetricAlgorithm` class.
- ◆ The `RSA` class is the base class for all the classes that implement the RSA algorithm.
- ◆ The RSA algorithm was designed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman and till now is the most widely adopted algorithm to perform asymmetric encryption and decryption.
- ◆ This algorithm functions in three steps: key generation, encryption, and decryption.

Asymmetric Encryption Algorithm 2-2

- ◆ The RSA algorithm generates a public key as a product of two large prime numbers, along with a public (or encryption) value.
- ◆ The algorithm generates a private key as a product of the same two large prime numbers, along with a private (or decryption) value.
- ◆ The public key is used to perform encryption while the private key is used to perform decryption.
- ◆ In the .NET Framework, the `RSACryptoServiceProvider` class derives from the `RSA` class to provide an implementation of the RSA algorithm.

- ◆ You need to use one of the symmetric encryption implementation classes of the .NET Framework to perform symmetric encryption.
- ◆ The first step to perform symmetric encryption is to create the symmetric key.



Generating Symmetric Keys 1-4

- ◆ When you use the default constructor of the symmetric encryption classes, such as `RijndaelManaged` and `AesManaged`, a key and IV are automatically generated.
- ◆ The generated key and the IV can be accessed as byte arrays using the `Key` and `IV` properties of the encryption class.
- ◆ The following code creates a symmetric key and IV using the `RijndaelManaged` class.

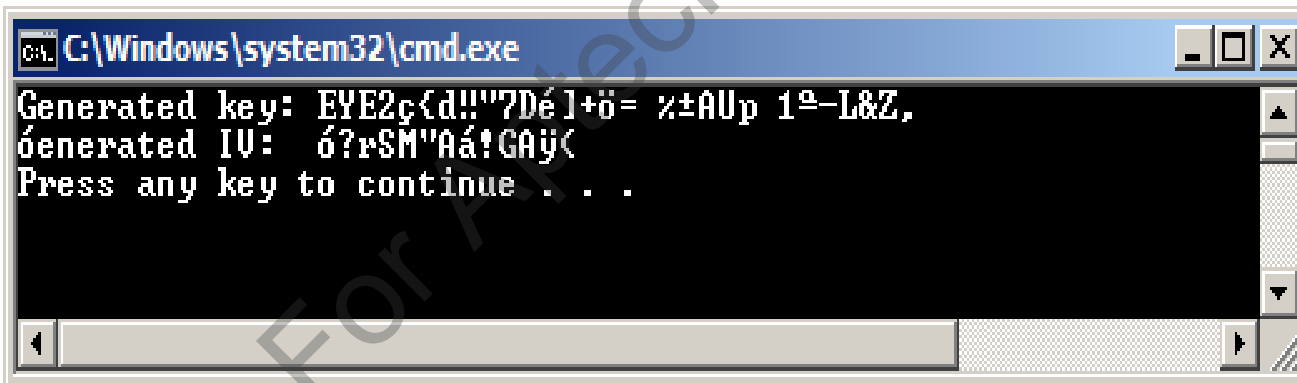
Snippet

```
using System;
using System.Security.Cryptography;
using System.Text;
. . .
. . .
RijndaelManagementsymAlgo = new RijndaelManaged();
Console.WriteLine("Generated key: {0}, \nGenerated IV:  {1}",
Encoding.Default.GetString(symAlgo.Key), Encoding.Default.GetString(symAlgo.IV));
```

Generating Symmetric Keys 2-4

- ◆ The code snippet uses the default constructor of the `RijndaelManaged` class to generate a symmetric key and IV.
- ◆ The `Key` and `IV` properties are accessed and printed as strings using the default encoding to the console.
- ◆ The following figure shows the output of the code.

Output



```
C:\Windows\system32\cmd.exe
Generated key: EYE2ç{d!!"7Dél+ö= %±AU p 1^~L&Z,
Generated IV: ó?rSM"Aá!GAÿ<
Press any key to continue . . .
```

Generating Symmetric Keys 3-4

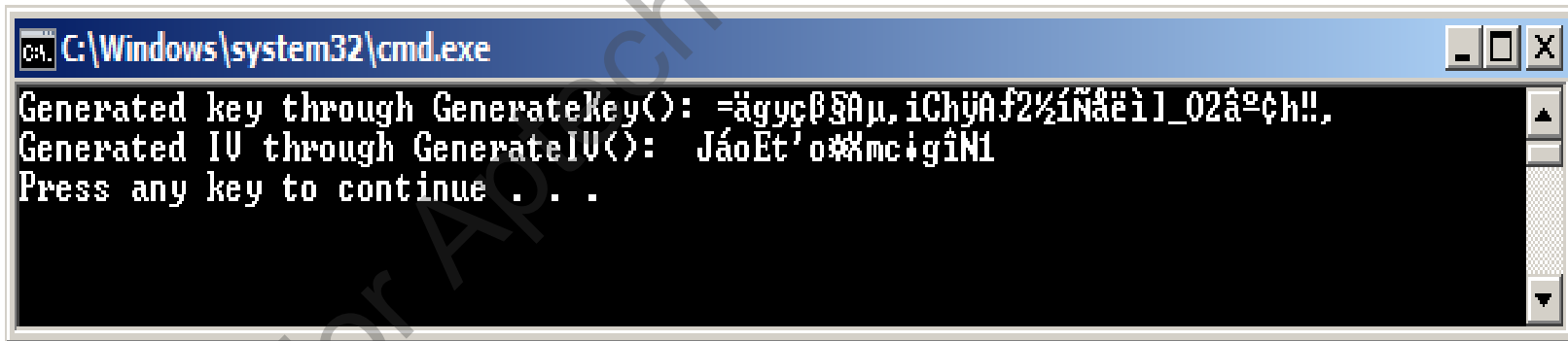
- ◆ The symmetric encryption classes, such as `RijndaelManaged` also provide the `GenerateKey()` and `GenerateIV()` methods that you can use to generate keys and IVs, as shown in the following code:

Snippet

```
using System;
using System.Security.Cryptography;
using System.Text;
. . .
. . .
RijndaelManagementsymAlgo = new RijndaelManaged();
symAlgo.GenerateKey();
symAlgo.GenerateIV();
byte[] generatedKey = symAlgo.Key;
byte[] generatedIV = symAlgo.IV;
Console.WriteLine("Generated key through GenerateKey(): {0}, \nGenerated IV through
    GenerateIV(): {1}", Encoding.Default.GetString(generatedKey),
    Encoding.Default.GetString(generatedIV));
```


- ◆ In the code:
 - ◆ An `RijndaelManaged` object is created and the `GenerateKey()` and `GenerateIV()` methods are called to generate a key and an IV.
 - ◆ The Key and the IV properties are then accessed and printed as strings using the default encoding to the console.
- ◆ The following figure shows the output of the code.

Output



```
C:\Windows\system32\cmd.exe
Generated key through GenerateKey(): =ägycß$Aµ,iChyAf2¼iÑÄëìl_02â°çh!!,
Generated IV through GenerateIV(): JáoEt'o%%mcigîNl
Press any key to continue . . .
```

- ◆ The symmetric encryption classes of the .NET Framework provides the `CreateEncryptor()` method that returns an object of the `ICryptoTransform` interface.
- ◆ The `ICryptoTransform` object is responsible for transforming the data based on the algorithm of the encryption class.
- ◆ Once you have obtained an `ICryptoTransform` object, you can use the `CryptoStream` class to perform encryption.
- ◆ The `CryptoStream` class acts as a wrapper of a stream-derived class, such as `FileStream`, `MemoryStream`, and `NetworkStream`.

- ◆ The `CryptoStream` class acts as a wrapper of a stream-derived class, such as `FileStream`, `MemoryStream`, and `NetworkStream`.
- ◆ A `CryptoStream` object operates in one of the following two modes defined by the `CryptoStreamMode` enumeration:
 - ◆ **Write:**
 - This mode allows write operation on the underlying stream.
 - Use this mode to perform encryption.
 - ◆ **Read:**
 - This mode allows read operation on the underlying stream.
 - Use this mode to perform decryption.

- ◆ You can create a `CryptoStream` object by calling the constructor that accepts the following three parameters:
 - ◆ The underlying stream object
 - ◆ The `ICryptoTransform` object
 - ◆ The mode defined by the `CryptoStreamMode` enumeration
- ◆ After creating the `CryptoStream` object you can call the `Write()` method to write the encrypted data to the underlying stream.
- ◆ The following code encrypts data using the `RijndaelManaged` class and writes the encrypted data to a file.

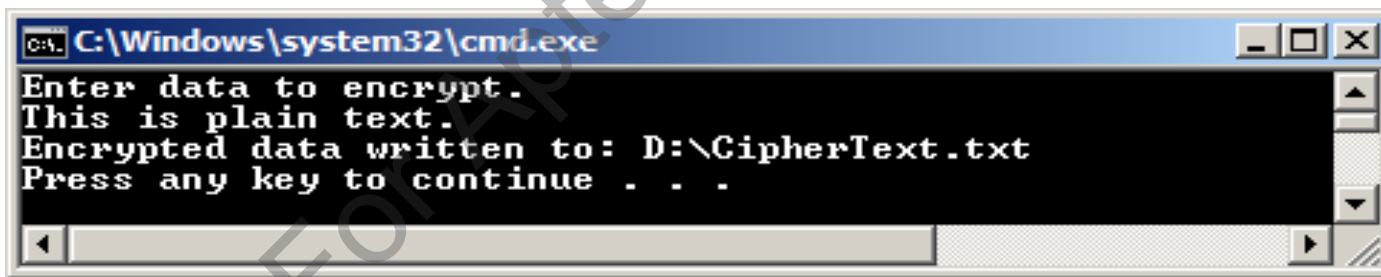
Snippet

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
class SymmetricEncryptionDemo
{
    static void EncryptData(String plainText, RijndaelManaged algo)
    {
        byte[] plainDataArray = ASCIIEncoding.ASCII.GetBytes(plainText);
```

```
ICryptoTransform transform=algo.CreateEncryptor();
using (var fileStream = new FileStream("D:\\CipherText.txt",
    FileMode.OpenOrCreate, FileAccess.Write))
{
    using (var cryptoStream = new CryptoStream(fileStream, transform,
        CryptoStreamMode.Write))
    {
        cryptoStream.Write(plainDataArray, 0,
            plainDataArray.GetLength(0));
        Console.WriteLine("Encrypted data written to:
            D:\\CipherText.txt");
    }
}
}
static void Main()
{
    RijndaelManaged symAlgo = new RijndaelManaged();
    Console.WriteLine("Enter data to encrypt.");
    string dataToEncrypt = Console.ReadLine();
    EncryptData(dataToEncrypt, symAlgo);
}
}
```

- ◆ In the code:
 - ◆ The `Main()` method creates a `RijndaelManaged` object and passes it along with the data to encrypt to the `EncryptData()` method.
 - ◆ The call to the `CreateEncryptor()` method creates the `ICryptoTransform` object in the `EncryptData()` method.
 - ◆ Then, a `FileStream` object is created to write the encrypted text to the `CipherText.txt` file.
 - ◆ The `CryptoStream` object is created and its `Write()` method is called.
- ◆ The following figure shows the output of the code.

Output



```
C:\Windows\system32\cmd.exe
Enter data to encrypt.
This is plain text.
Encrypted data written to: D:\CipherText.txt
Press any key to continue . . .
```

- ◆ To decrypt data, you need to:

Step 1

- Use the same symmetric encryption class, key, and IV used for encrypting the data.

Step 2

- Call the `CreateDecryptor()` method to obtain a `ICryptoTransform` object that will perform the transformation.

Step 3

- Create the `CryptoStream` object in Read mode and initialize a `StreamReader` object with the `CryptoStream` object.

Step 4

- Call the `ReadToEnd()` method of the `StreamReader` that returns the decrypted text as a string.

- ◆ The code creates a program that performs both encryption and decryption.

Snippet

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;
class SymmetricEncryptionDemo
{
    static void EncryptData(String plainText, RijndaelManaged algo)
    {
        byte[] plainDataArray = ASCIIEncoding.ASCII.GetBytes(plainText);

        ICryptoTransform transform = algo.CreateEncryptor();
        using (var fileStream = new FileStream("D:\\CipherText.txt",
        FileMode.OpenOrCreate, FileAccess.Write))
        {
            using (var cryptoStream = new CryptoStream(fileStream, transform,
        CryptoStreamMode.Write))
            {
                cryptoStream.Write(plainDataArray, 0, plainDataArray.GetLength(0));
                Console.WriteLine("Encrypted data written to: D:\\CipherText.txt");
            }
        }
    }
    static void DecryptData(RijndaelManaged algo)
    {

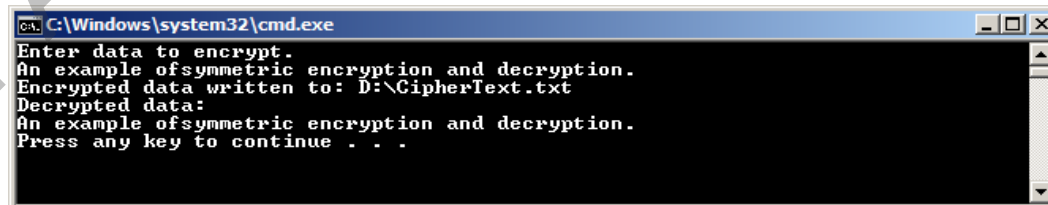
```

```
ICryptoTransform transform = algo.CreateDecryptor();
    using (var fileStream = new FileStream("D:\\CipherText.txt", FileMode.Open,
FileAccess.Read))
    {
        using (CryptoStream cryptoStream = new CryptoStream(fileStream,
transform, CryptoStreamMode.Read))
        {
            using (var streamReader = new StreamReader(cryptoStream))
            {
                string decryptedData = streamReader.ReadToEnd();
                Console.WriteLine("Decrypted data: \n{0}", decryptedData);
            }
        }
    }
}
static void Main()
{
    RijndaelManaged symAlgo = new RijndaelManaged();
    Console.WriteLine("Enter data to encrypt.");
    string dataToEncrypt = Console.ReadLine();
    EncryptData(dataToEncrypt, symAlgo);
    DecryptData(symAlgo);
}
}
```

◆ In the code:

- ◆ The `Main()` method creates a `RijndaelManaged` object and passes it along with the data to encrypt the `EncryptData()` method.
 - ◆ The encrypted data is saved to the `CipherText.txt` file.
 - ◆ The `Main()` method calls the `DecryptData()` method passing the same `RijndaelManaged` object created for encryption.
 - ◆ The `DecryptData()` method creates the `ICryptoTransform` object and uses a `FileStream` object to read the encrypted data from the file.
 - ◆ The `CryptoStream` object is created in the Read mode initialized with the `FileStream` and `ICryptoTransform` objects.
 - ◆ A `StreamReader` object is created by passing the `CryptoStream` object to the constructor.
 - ◆ The `ReadToEnd()` method of the `StreamReader` object is called.
- ◆ The decrypted text returned by the `ReadToEnd()` method is printed to the console, as shown in the following figure:

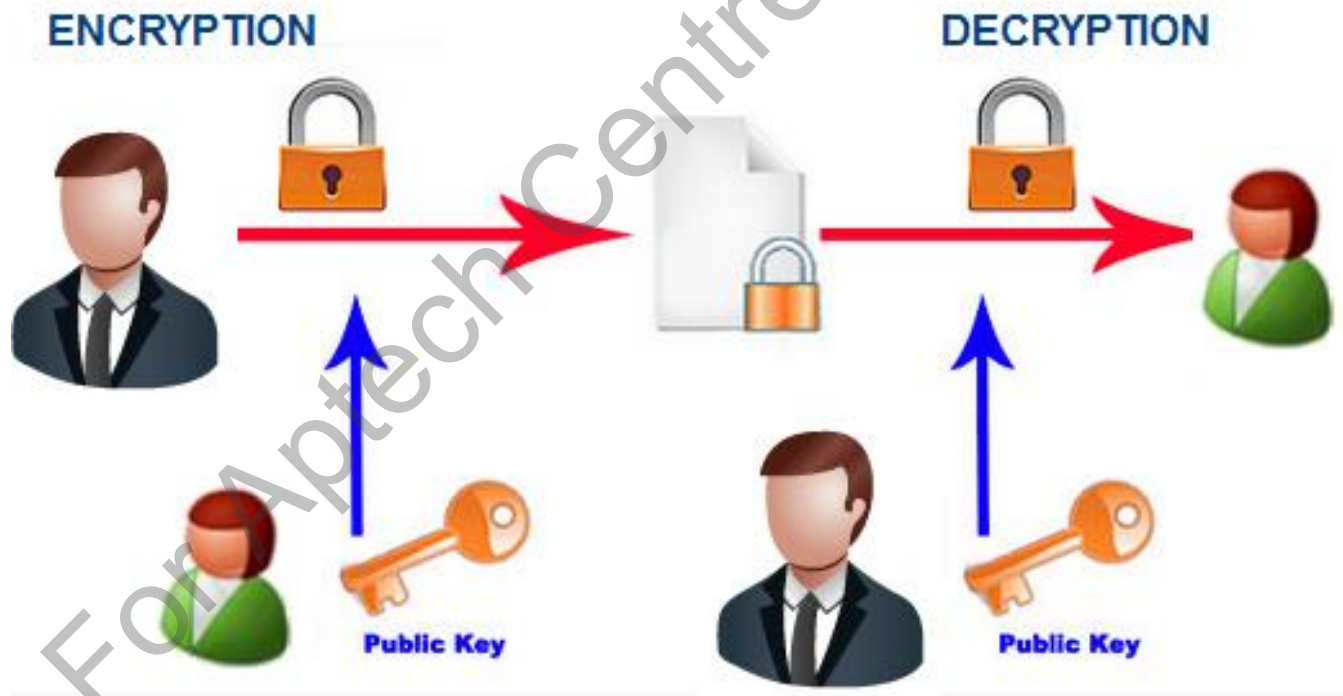
Output



```
C:\Windows\system32\cmd.exe
Enter data to encrypt.
An example of symmetric encryption and decryption.
Encrypted data written to: D:\CipherText.txt
Decrypted data:
An example of symmetric encryption and decryption.
Press any key to continue . . .
```

Performing Asymmetric Encryption

- ◆ You can use the `RSACryptoServiceProvider` class of the `System.Security.Cryptography` namespace to perform asymmetric encryption.



- ◆ When you call the default constructor of the `RSACryptoServiceProvider` class, a new public/private key pair is automatically generated.
- ◆ After you create a new instance of the class, you can export the key information by using one of the following methods:
 - ◆ `ToXMLString()`
 - Returns an XML representation of the key information.
 - ◆ `ExportParameters()`
 - Returns an `RSAPParameters` structure that holds the key information.
 - Accept a Boolean value by both the `ToXMLString()` and `ExportParameters()` methods.
 - A `false` value indicates that the method should return only the public key information while a `true` value indicates that the method should return information of both the public and private keys.

Generating Asymmetric Keys 2-2

- ◆ The following code shows how to create and initialize an `RSACryptoServiceProvider` object and then export the public key in XML format:

Snippet

```
using System;
using System.Security.Cryptography;
using System.Text;
. . .
RSACryptoServiceProviderrSAKeyGenerator = new RSACryptoServiceProvider();
string publicKey = rSAKeyGenerator.ToXmlString(false);
}
```

- ◆ The following code shows how to create and initialize an `RSACryptoServiceProvider` object and then export both the public and private keys as an `RSAParameters` structure:

Snippet

```
using System;
using System.Security.Cryptography;
using System.Text;
. . .
RSACryptoServiceProviderrSAKeyGenerator = new RSACryptoServiceProvider();
RSAParametersrSAKeyInfo = rSAKeyGenerator.ExportParameters(true);
```


- ◆ Private keys used to decrypt data in asymmetric encryption should be stored using a secured mechanism.
- ◆ To achieve this, you can use a key container that is a logical structure to securely store asymmetric keys.
- ◆ The .NET Framework provides the `CspParameters` class to create a key container and to add and remove keys to and from the container.
- ◆ To create a key container for an `RSACryptoServiceProvider` object:
 - ◆ Use the default constructor of the `CspParameters` class to create a key container instance.
 - ◆ Set the container name using the `KeyContainerName` property of the `CspParameters` class.
 - ◆ Pass the `CspParameters` object to the constructor while creating the `RSACryptoServiceProvider` object to store the key pair in the key container.

- The following code uses a key container to store a key pair:

Snippet

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

...

CspParameters cspParams = new CspParameters(); cspParams.KeyContainerName =
"RSA_CONTAINER"; RSACryptoServiceProvider rsaKeyGenerator =
new RSACryptoServiceProvider(cspParams); Console.WriteLine("RSA key added to the
container, \n\n{0}",    rsaKeyGenerator.ToXmlString(true));
```

- The following figure shows the output of the code:

Output

```
C:\Windows\system32\cmd.exe
RSA key added to the container.
<RSAKeyValue><Modulus>tnoov3WnR/ZfVgS9UxE3xUKhBDqA+c5B195s05IEgXWgZoUJSq+3mkHHE
WZF0JHm9vMsIzOLL6dB0Qm0+gSf1Ri1oPpAU12+Ue5Bn6c58jKopoY17cS5PEWw81MIJozUgAQOfTnIF
IK9ObExsQsw27UJDNeWELWce9+hanZmm8=</Modulus><Exponent>AQAB</Exponent><P>64HMUdro
r-hosErRiFQI7mYSNrJZmcv4CgI9+3W6U9SGjLC809d27Vupxf8yk0Ntr0d6czoloB060nWh1rfiaRw==
</P><Q>x1s00B06KewLNB7ee9LW53kG-NFSvt4n4QnABRyim50MC5ddw+34Ld8shSL6DNPEiE0XLeve
vUJifz3leuQmQ==</Q><DP>IZW3D41KUUuCI0B2mm2Ue8Uddx1Trt5ufp91Ekfo9vY/UNPC170xP2/1V
6ZJcsd3zssSTJj6gvMRFEmlTq37JQ==</DP><DQ>s/y0eEOi0ITwNZ9GFfWXY/sBweMPPnq/nQgh4ZuW
QbZD09CygPtSk9/oUPbgBuux/jleYTOW6j0EHCiH39etIQ==</DQ><InverseQ>5bHIIcrr8Q13W25xh
XPHiVoS/oxzjUDM7MTemsPiIIgSrql9wgmKFO9IUDjYkPUDzCD458igLFPRDqhy/UMYA==</Inverse
Q><D>UleHeUN6a1a90C/gigmorOusIB0RZ21BcUAH9CoejMUhXm6NMQOEhUQWc3yZhCih/agNgifjkk
6jklPtPRLDXmiWd3epKoZU5qRz7146a3jFdw9UGcHYLis15RAX+ttJUFRB/rj8vsio7s6tNt9Uc4xpUR
+E3MnRZDphGnkPHE=</D></RSAKeyValue>
Press any key to continue . . .
```

- ◆ To retrieve a key pair from the container:
 - ◆ Create a new object of the `CspParameters` class and initialize it with the name of the key container that contains the key pair.
 - ◆ Create a new object of the `RSACryptoServiceProvider` initialized with the `CspParameters` object.
- ◆ The `RSACryptoServiceProvider` object will now contain the keys stored in the key container.
- ◆ The following code retrieves keys from the key container, named `RSA_CONTAINER`:

Snippet

```
using System;
using System.Security.Cryptography;
using System.Text;
. . .
CspParameters cp = new CspParameters();
cp.KeyContainerName = "RSA_CONTAINER";
RSACryptoServiceProvider rsaEncryptor = new RSACryptoServiceProvider(cp);
```

- ◆ To encrypt data, you need to:
 - ◆ Create a new instance of the `RSACryptoServiceProvider` class
 - ◆ Call the `ImportParameters()` method to initialize the instance with the public key information exported to an `RSAParameters` structure.
- ◆ The following code shows how to initialize an `RSACryptoServiceProvider` object with the public key exported to an `RSAParameters` structure:



Snippet

```
using System;
using System.Security.Cryptography;
using System.Text;
. . .
. . .
RSACryptoServiceProvider rSAKeyGenerator = new
RSACryptoServiceProvider();
RSAParameters rSAKeyInfo = rSAKeyGenerator.ExportParameters(false);
RSACryptoServiceProvider rsaEncryptor = new
RSACryptoServiceProvider();
rsaEncryptor.ImportParameters(rSAKeyInfo);
```

- ◆ If the public key information is exported to XML format, you need to call the `FromXmlString()` method to initialize the `RSACryptoServiceProvider` object with the public key, as shown in the following code:

Snippet

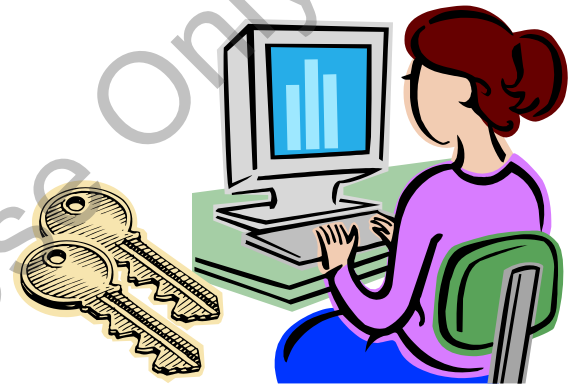
```
using System;
using System.Security.Cryptography;
using System.Text;
. . .
RSACryptoServiceProvider rsaKeyGenerator = new
RSACryptoServiceProvider();
string publicKey = rsaKeyGenerator.ToXmlString(false);
RSACryptoServiceProvider rsaEncryptor = new RSACryptoServiceProvider();
rsaEncryptor.FromXmlString(publicKey);
```

- ◆ After the `RSACryptoServiceProvider` object is initialized with the public key, you can encrypt data by calling the `Encrypt()` method of the `RSACryptoServiceProvider` class.
- ◆ The `Encrypt()` method accepts the following two parameters:
 - ◆ byte array of the data to encrypt
 - ◆ A `Boolean` value (It indicates whether or not to perform encryption using Optimal Asymmetric Encryption Padding (OAEP) padding. A true value uses OAEP padding while a false value uses PKCS#1 v1.5 padding.)
- ◆ The `Encrypt()` method after performing encryption returns a byte array of the encrypted text, as shown in the code:

Snippet

```
byte[] plainbytes = new UnicodeEncoding().GetBytes("Plain text to encrypt.");  
byte[] cipherbytes = rsaEncryptor.Encrypt(plainbytes, true);
```

- ◆ To decrypt data, you need to initialize an `RSACryptoServiceProvider` object using the private key of the key pair whose public key was used for encryption.



- ◆ The following code shows how to initialize an `RSACryptoServiceProvider` object with the private key exported to an `RSAParameters` structure:

Snippet

```
RSACryptoServiceProviderrSAKeyGenerator = new  
RSACryptoServiceProvider();  
RSAParametersrSAKeyInfo = rSAKeyGenerator.ExportParameters(true);  
RSACryptoServiceProviderrsaDecryptor= new RSACryptoServiceProvider();  
rsaDecryptor.ImportParameters(rSAKeyInfo);
```

- ◆ The following code shows how to initialize an `RSACryptoServiceProvider` object with the private key exported to XML format:

Snippet

```
RSACryptoServiceProviderrSAKeyGenerator = new  
RSACryptoServiceProvider();  
string keyPair = rSAKeyGenerator.ToXmlString(true);  
RSACryptoServiceProviderrsaDecryptor = new  
RSACryptoServiceProvider();  
rsaDecryptor.FromXmlString(keyPair);
```


- ◆ When the `RSACryptoServiceProvider` object is initialized with the private key, you can decrypt data by calling the `Decrypt()` method of the `RSACryptoServiceProvider` class.
- ◆ The `Decrypt()` method accepts the following two parameters:
 - ◆ `byte` array of the encrypted data
 - ◆ A `Boolean` value (It indicates whether or not to perform encryption using OAEP padding. A true value uses OAEP padding while a false value uses PKCS#1 v1.5 padding.)

- ◆ The `Decrypt()` method returns a byte array of the original data, as shown in the following code:

Snippet

```
byte[] plainbytes = rsaDecryptor.Decrypt(cipherbytes, false);
```

- ◆ The following code shows a program that performs asymmetric encryption and decryption:

Snippet

```
using System;  
using System.IO;  
using System.Security.Cryptography;  
using System.Text;  
class AsymmetricEncryptionDemo  
{
```

```
static byte[] EncryptData(string plainText, RSAParameters
rsaParameters)
{
byte[] plainTextArray = new
UnicodeEncoding().GetBytes(plainText);
RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();
RSA.ImportParameters(rsaParameters);
byte[] encryptedData = RSA.Encrypt(plainTextArray,true);
    return encryptedData;

}

static byte[] DecryptData(byte[] encryptedData, RSAParameters
rsaParameters)
{
RSACryptoServiceProvider RSA = new
RSACryptoServiceProvider();
RSA.ImportParameters(rsaParameters);
byte[] decryptedData = RSA.Decrypt(encryptedData,true);
return decryptedData;
```

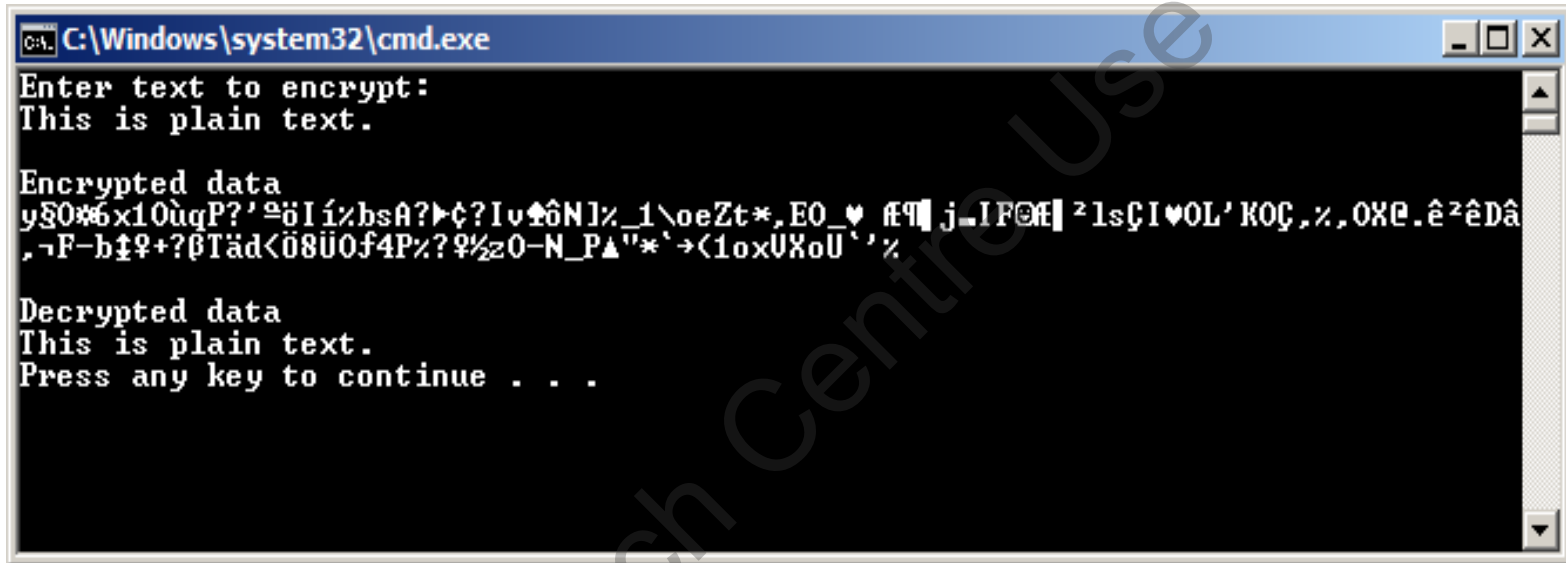
```
}  
  
static void Main(string[] args)  
{  
    Console.WriteLine("Enter text to encrypt:");  
    String inputText = Console.ReadLine();  
    RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();  
    RSAParameters RSAParam = RSA.ExportParameters(false);  
    byte[] encryptedData = EncryptData(inputText, RSAParam);  
    string encryptedString =  
        Encoding.Default.GetString(encryptedData);  
    Console.WriteLine("\nEncrypted data \n{0}", encryptedString);  
    byte[] decryptedData = DecryptData(encryptedData,  
                                       RSA.ExportParameters(true));  
    String decryptedString = new  
        UnicodeEncoding().GetString(decryptedData);  
    Console.WriteLine("\nDecrypted data \n{0}", decryptedString);  
}  
}
```

◆ In the code:

- ◆ The `Main()` method creates a `RSACryptoServiceProvider` object and exports the public key as a `RSAParameters` structure.
- ◆ The `EncryptData()` method is then called passing the user entered plain text and the `RSAParameters` object.
- ◆ The `EncryptData()` method uses the exported public key to encrypt the data and returns the encrypted data as a byte array.
- ◆ The `Main()` method then exports both the public and private key of the `RSACryptoServiceProvider` object into a second `RSAParameters` object.
- ◆ The `DecryptData()` method is called passing the encrypted byte array and the `RSAParameters` object.
- ◆ The `DecryptData()` method performs the decryption and returns the original plain text as a string.

- The following figure shows the output of the code.

Output



```
C:\Windows\system32\cmd.exe
Enter text to encrypt:
This is plain text.

Encrypted data
yS0%6x10uqP?'°öIí%bsA?>ç?Iv±ôNl%_1\oeZt*,EO_♥ æ¶ j-IF@æ zlsçI♥OL'KOç,%,OX@.ê²êDâ
,-F-b±q+?βTäd<08Ü0f4P%?¶½z0-N_P▲"*'→<1oxUxoU`',%

Decrypted data
This is plain text.
Press any key to continue . . .
```

- ◆ Encryption is a security mechanism that converts data in plain text to cipher text.
- ◆ An encryption key is a piece of information or parameter that determines how a particular encryption mechanism works.
- ◆ The .NET Framework provides various types in the `System.Security.Cryptography` namespace to support symmetric and asymmetric encryptions.
- ◆ When you use the default constructor to create an object of the symmetric encryption classes, a key and IV are automatically generated.
- ◆ The `ICryptoTransform` object is responsible for transforming the data based on the algorithm of the encryption class.
- ◆ The `CryptoStream` class acts as a wrapper of a stream-derived class, such as `FileStream`, `MemoryStream`, and `NetworkStream`.
- ◆ When you call the default constructor of the `RSACryptoServiceProvider` and `DSACryptoServiceProvider` classes, a new public/private key pair is automatically generated.