Microsoft® Official Course

Module 13

Encrypting and Decrypting Data

Module Overview

- Implementing Symmetric Encryption
- Implementing Asymmetric Encryption

Lesson 1: Implementing Symmetric Encryption

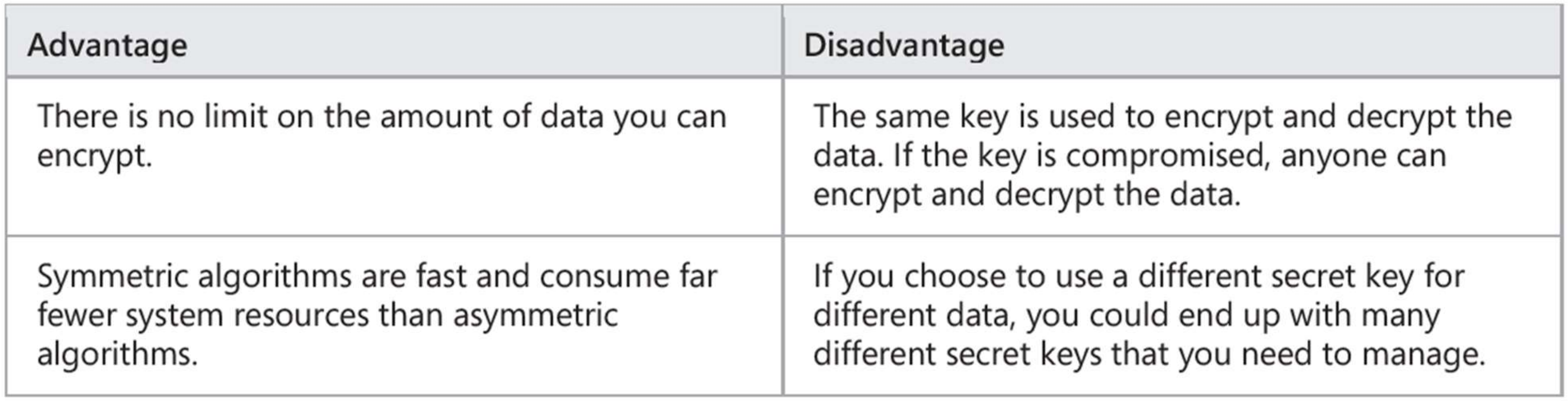
- What Is Symmetric Encryption?
- Encrypting Data by Using Symmetric Encryption
- Hashing Data
- Demonstration: Encrypting and Decrypting Data

What Is Symmetric Encryption?

- Symmetric encryption is the cryptographic transformation of data by using a mathematical algorithm in which the same key is used to encrypt and decrypt the data
  - Therefore, when you use symmetric encryption, you must keep the secret key secure

- To help improve the effectiveness of symmetric encryption, many symmetric encryption algorithms also use an initialization vector (IV) in addition to a secret key.
  - The IV is an arbitrary block of bytes that helps to randomize the first encrypted block of data.
  - The IV makes it much more difficult for a malicious user to decrypt your data.

- Advantages and Disadvantages of Symmetric Encryption:

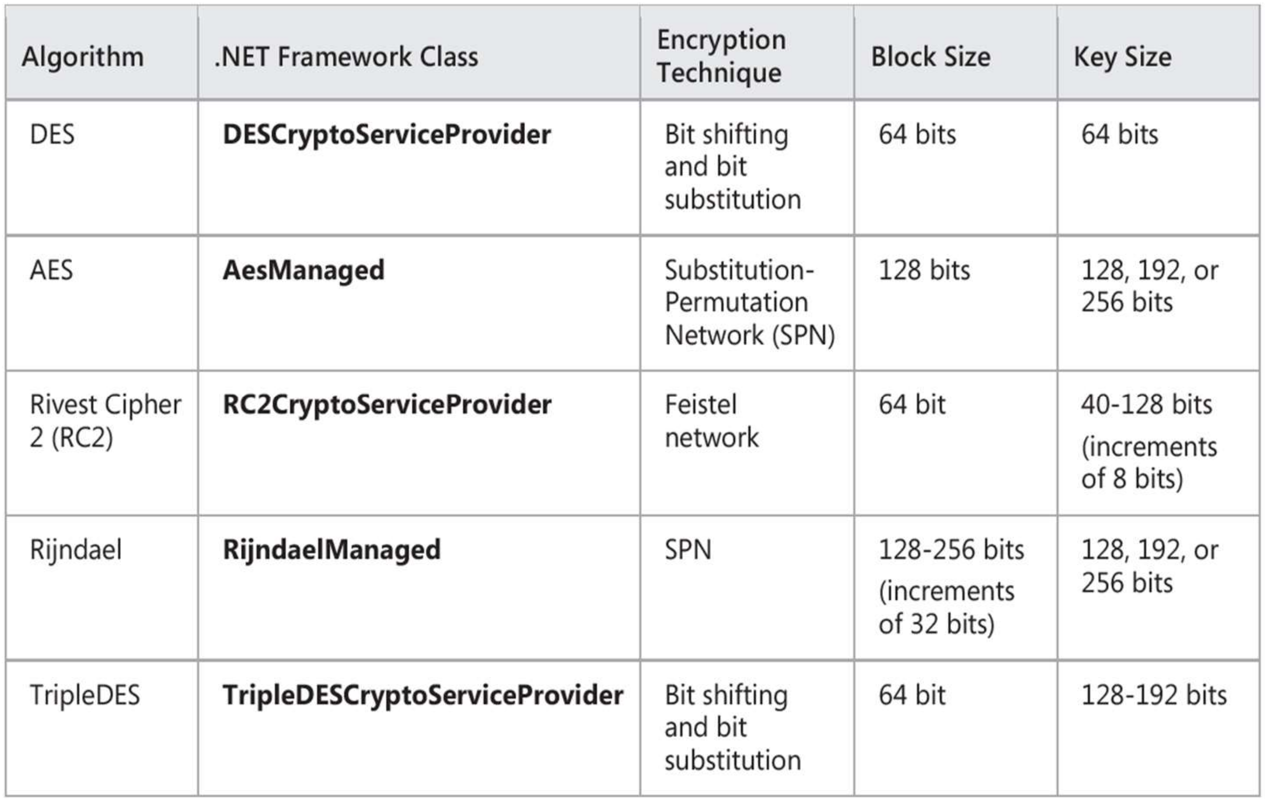


Advantage	Disadvantage
There is no limit on the amount of data you can encrypt.	The same key is used to encrypt and decrypt the data. If the key is compromised, anyone can encrypt and decrypt the data.
Symmetric algorithms are fast and consume far fewer system resources than asymmetric algorithms.	If you choose to use a different secret key for different data, you could end up with many different secret keys that you need to manage.

Symmetric Encryption Classes in the .NET Framework

- Each of the .NET Framework encryption classes are known as block ciphers, because the algorithm will chunk data into fixed-length blocks and then perform a cryptographic transformation on each block.
  - The higher the number of bits, the larger the number of possible secret keys …
  - 2^64 = 18,446,744,073,709,551,616, 2^128 = ? How long would it take to brute force … ?

- The **System.Security.Cryptography** namespace includes:
  - See also: http://go.microsoft.com/fwlink/?LinkID=267877
  - Data Encryption Standard (DES)
  - Advanced Encryption Standard (AES)



Algorithm	.NET Framework Class	Encryption Technique	Block Size	Key Size
DES	**DESCryptoServiceProvider**	Bit shifting and bit substitution	64 bits	64 bits
AES	**AesManaged**	Substitution-Permutation Network (SPN)	128 bits	128, 192, or 256 bits
Rivest Cipher 2 (RC2)	**RC2CryptoServiceProvider**	Feistel network	64 bit	40-128 bits (increments of 8 bits)
Rijndael	**RijndaelManaged**	SPN	128-256 bits (increments of 32 bits)	128, 192, or 256 bits
TripleDES	**TripleDESCryptoServiceProvider**	Bit shifting and bit substitution	64 bit	128-192 bits

Encrypting Data by Using Symmetric Encryption

- Aside from encrypting and decrypting data by using an algorithm, the encryption process typically involves the following tasks:
  - Derive a secret key and an IV from a password or salt.
    - A salt is a random collection of bits used in combination with a password to generate a secret key and an IV.
    - A salt makes it much more difficult for a malicious user to randomly discover the secret key.
  - Read and write encrypted data to and from a stream

Encrypting Data by Using Symmetric Encryption

Steps to encrypt and decrypt data symmetrically:

1. Create an **Rfc2898DeriveBytes** object
   - provides an implementation of the password-based key derivation function (PBKDF2)
   - use this object to derive your secret keys and your IVs from a password and a salt

```
var password = "Pa$$w0rd";
var salt = "S@lt";
var rgb = new Rfc2898DeriveBytes(password, Encoding.Unicode.GetBytes(salt));
```

2. Create an **AesManaged** object
   - used to encrypt the data

```
var algorithm = new AesManaged();
```

3. Generate a secret key and an IV (byte[])
   - Divide by 8 to get # of bytes from # of bits
   - Get pseudo random keys for secret key and for the IV

```
var rgbKey = rgb.GetBytes(algorithm.KeySize / 8);
var rgbIV = rgb.GetBytes(algorithm.BlockSize / 8);
```

4. Create a stream to buffer the transformed (encrypted/unencrypted) data

```
var bufferStream = new System.IO.MemoryStream();
```

5. Create a symmetric encryptor/decryptor object

```
// Create an encryptor object.
var transformer = algorithm.CreateEncryptor(rgbKey, rgbIV);
//...
//// Create a decryptor object.
//var transformer = algorithm.CreateDecryptor(rgbKey, rgbIV);
```

6. Create a **CryptoStream** object
   - use to write the cryptographic bytes to the buffer stream.

```
var cryptoStream = new CryptoStream(
    bufferStream,
    transformer,
    CryptoStreamMode.Write);
```

7. Write the transformed data to the buffer stream
   - Invoke the Write and FlushFinalBlock methods to perform the cryptographic transform.

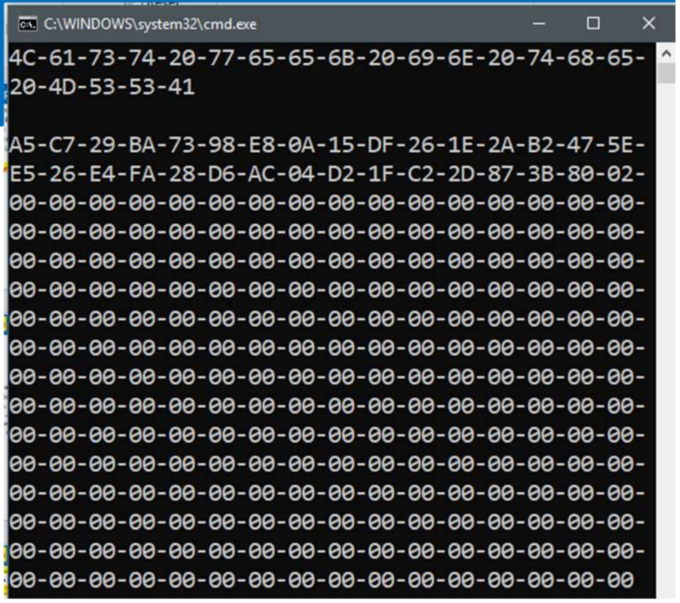
```
var bytesToTransform = Encoding.ASCII.GetBytes("Last week of MSSA")
cryptoStream.Write(bytesToTransform, 0, bytesToTransform.Length);
cryptoStream.FlushFinalBlock();
```

8. Close the streams
   - transformed data is flushed to the buffer stream

```
cryptoStream.Close();
bufferStream.Close();
```

Example - skip

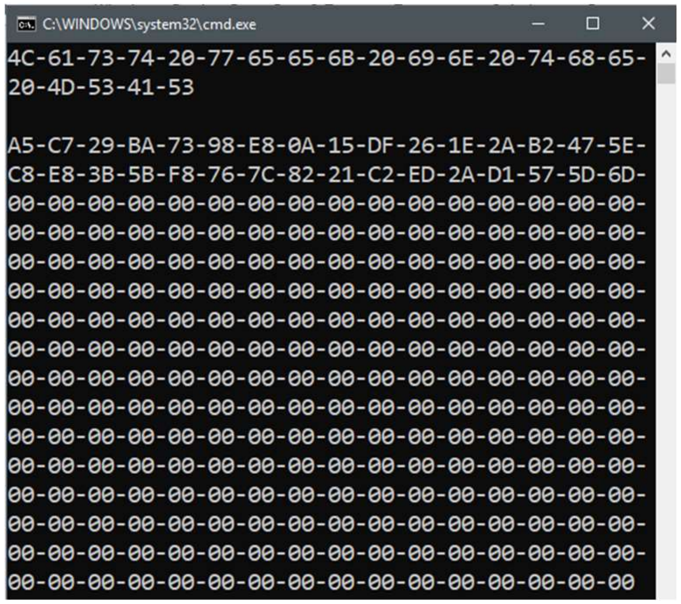
```
var password = "Pa$$w0rd";
var salt = "S@lt";
var rgb = new Rfc2898DeriveBytes(password, Encoding.Unicode.GetBytes(salt));

var algorithm = new AesManaged();

var rgbKey = rgb.GetBytes(algorithm.KeySize / 8);//KeySize is in bits ...
var rgbIV = rgb.GetBytes(algorithm.BlockSize / 8); //BlockSize is in bits

var bufferStream = new System.IO.MemoryStream();

// Create an encryptor object.
var transformer = algorithm.CreateEncryptor(rgbKey, rgbIV);
//// Create a decryptor object.
//var transformer = algorithm.CreateDecryptor(rgbKey, rgbIV);

var cryptoStream = new CryptoStream(
    bufferStream,
    transformer,
    CryptoStreamMode.Write);

String message = "Last week in the MSSA";
var bytesToTransform = Encoding.ASCII.GetBytes(message);
Console.WriteLine(BitConverter.ToString(Encoding.ASCII.GetBytes(message)));

Console.WriteLine(BitConverter.ToString(bufferStream.GetBuffer()));


cryptoStream.Write(bytesToTransform, 0, bytesToTransform.Length);
cryptoStream.FlushFinalBlock();

Console.WriteLine(BitConverter.ToString( bufferStream.GetBuffer()));
cryptoStream.Close();
bufferStream.Close();
```



C:\WINDOWS\system32\cmd.exe

```
4C-61-73-74-20-77-65-65-6B-20-69-6E-20-74-68-65-
20-4D-53-53-41

A5-C7-29-BA-73-98-E8-0A-15-DF-26-1E-2A-B2-47-5E-
E5-26-E4-FA-28-D6-AC-04-D2-1F-C2-2D-87-3B-80-02-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
```

```
String message = "Last week in the MSAS";
```



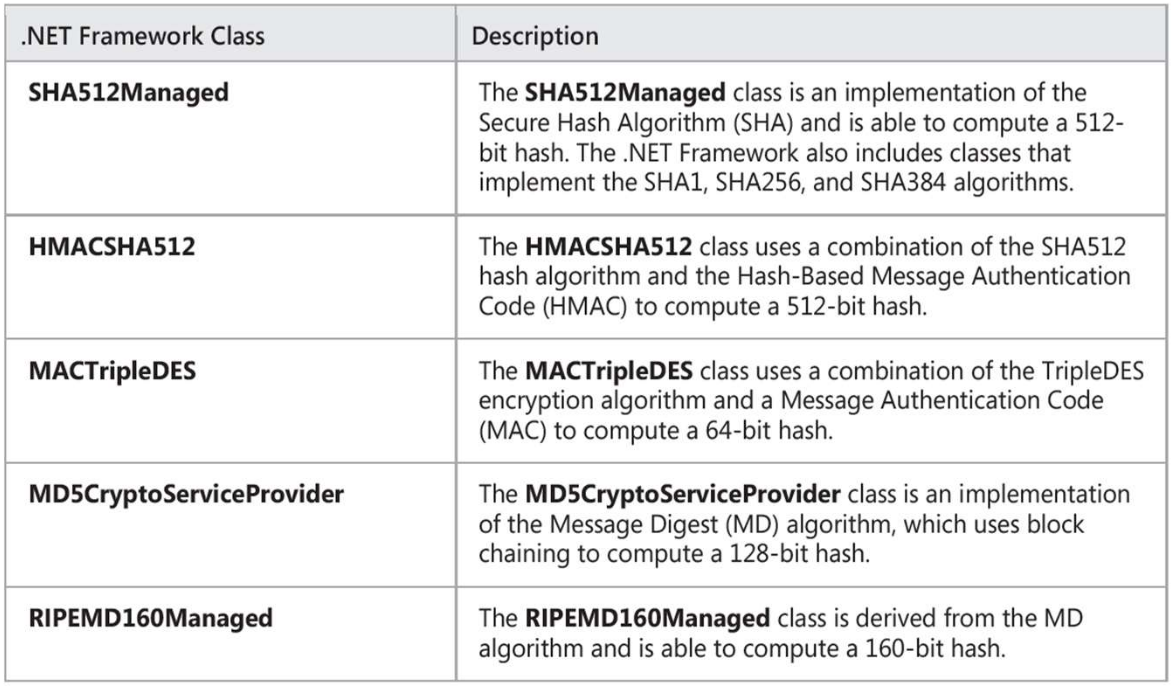
C:\WINDOWS\system32\cmd.exe

```
4C-61-73-74-20-77-65-65-6B-20-69-6E-20-74-68-65-
20-4D-53-41-53

A5-C7-29-BA-73-98-E8-0A-15-DF-26-1E-2A-B2-47-5E-
C8-E8-3B-5B-F8-76-7C-82-21-C2-ED-2A-D1-57-5D-6D-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
```

Hashing Data

- A hash is a numerical representation of a piece of data
  - If you use a good hash algorithm, it is unlikely to get the same hash for two different pieced of data: therefore a hash can be thought of as a digital fingerprint
  - applications often use hashes to store sensitive data if the value never has to be displayed.
    - Use the example of a password. It is very rare that a system will display a password onscreen in plain text, which is why they are often hashed. The meaning of the password still exists, but it is now represented by a numerical piece of data that is totally meaningless if you do not have the original data to hash and compare.
      - C:\Windows\System32\config\SAM file stores hashes
  - another application: to check/assure data integrity

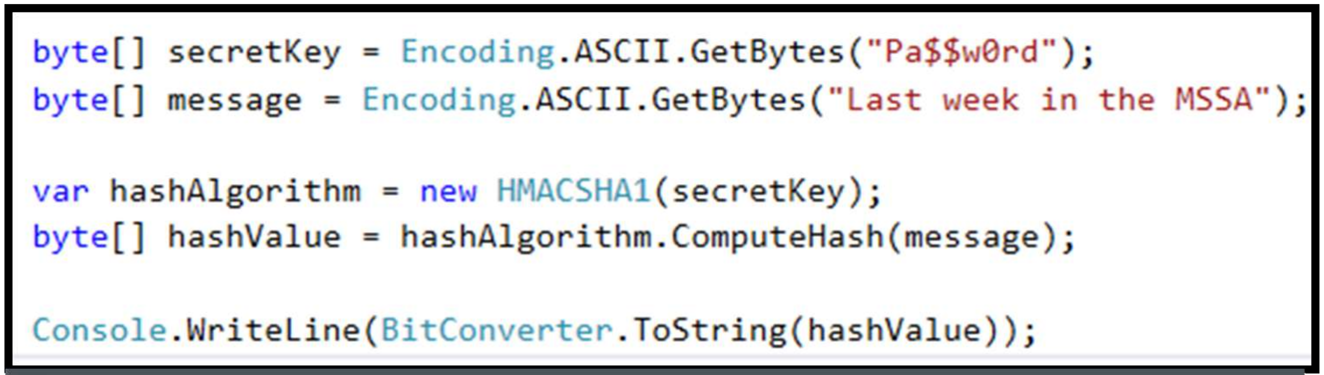
- Hash Algorithms in the .NET Framework
  - **System.Security.Cryptography**



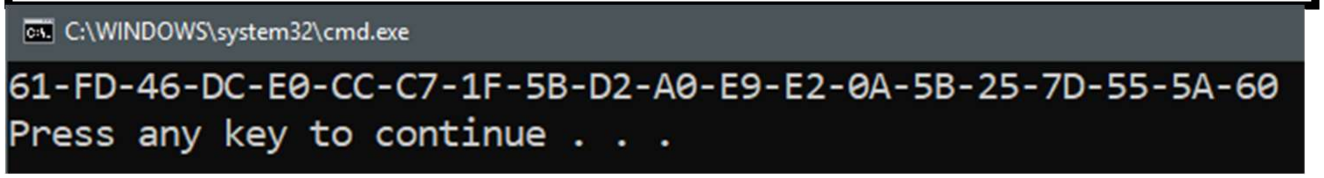
.NET Framework Class	Description
SHA512Managed	The **SHA512Managed** class is an implementation of the Secure Hash Algorithm (SHA) and is able to compute a 512-bit hash. The .NET Framework also includes classes that implement the SHA1, SHA256, and SHA384 algorithms.
HMACSHA512	The **HMACSHA512** class uses a combination of the SHA512 hash algorithm and the Hash-Based Message Authentication Code (HMAC) to compute a 512-bit hash.
MACTripleDES	The **MACTripleDES** class uses a combination of the TripleDES encryption algorithm and a Message Authentication Code (MAC) to compute a 64-bit hash.
MD5CryptoServiceProvider	The **MD5CryptoServiceProvider** class is an implementation of the Message Digest (MD) algorithm, which uses block chaining to compute a 128-bit hash.
RIPEMD160Managed	The **RIPEMD160Managed** class is derived from the MD algorithm and is able to compute a 160-bit hash.

Hashing Data

- A hash can be computed by using the following code

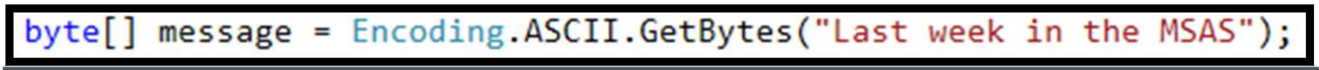


```
byte[] secretKey = Encoding.ASCII.GetBytes("Pa$$w0rd");
byte[] message = Encoding.ASCII.GetBytes("Last week in the MSSA");

var hashAlgorithm = new HMACSHA1(secretKey);
byte[] hashValue = hashAlgorithm.ComputeHash(message);

Console.WriteLine(BitConverter.ToString(hashValue));
```

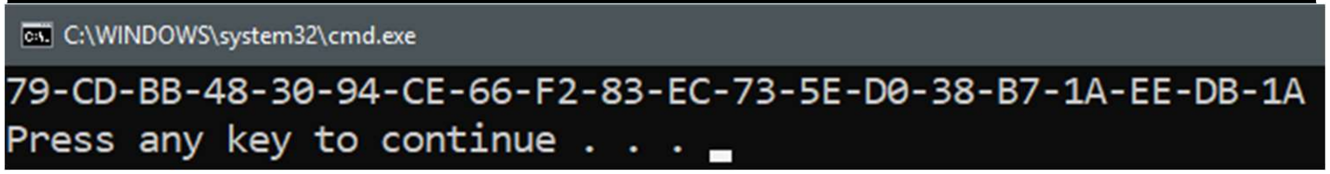


C:\WINDOWS\system32\cmd.exe

```
61-FD-46-DC-E0-CC-C7-1F-5B-D2-A0-E9-E2-0A-5B-25-7D-55-5A-60
Press any key to continue . . .
```



```
byte[] message = Encoding.ASCII.GetBytes("Last week in the MSAS");
```



C:\WINDOWS\system32\cmd.exe

```
79-CD-BB-48-30-94-CE-66-F2-83-EC-73-5E-D0-38-B7-1A-EE-DB-1A
Press any key to continue . . .
```

```
public byte[] ComputeHash(byte[] dataToHash, byte[] secretKey)
{
  using (var hashAlgorithm = new HMACSHA1(secretKey))
  {
    using (var bufferStream = new MemoryStream(dataToHash))
    {
      return hashAlgorithm.ComputeHash(bufferStream);
    }
  }
}
```

Lesson 2: Implementing Asymmetric Encryption

- What Is Asymmetric Encryption?
- Encrypting Data by Using Asymmetric Encryption
- Creating and Managing X509 Certificates
- Managing Encryption Keys
- Demonstration: Encrypting and Decrypting Grade Reports Lab

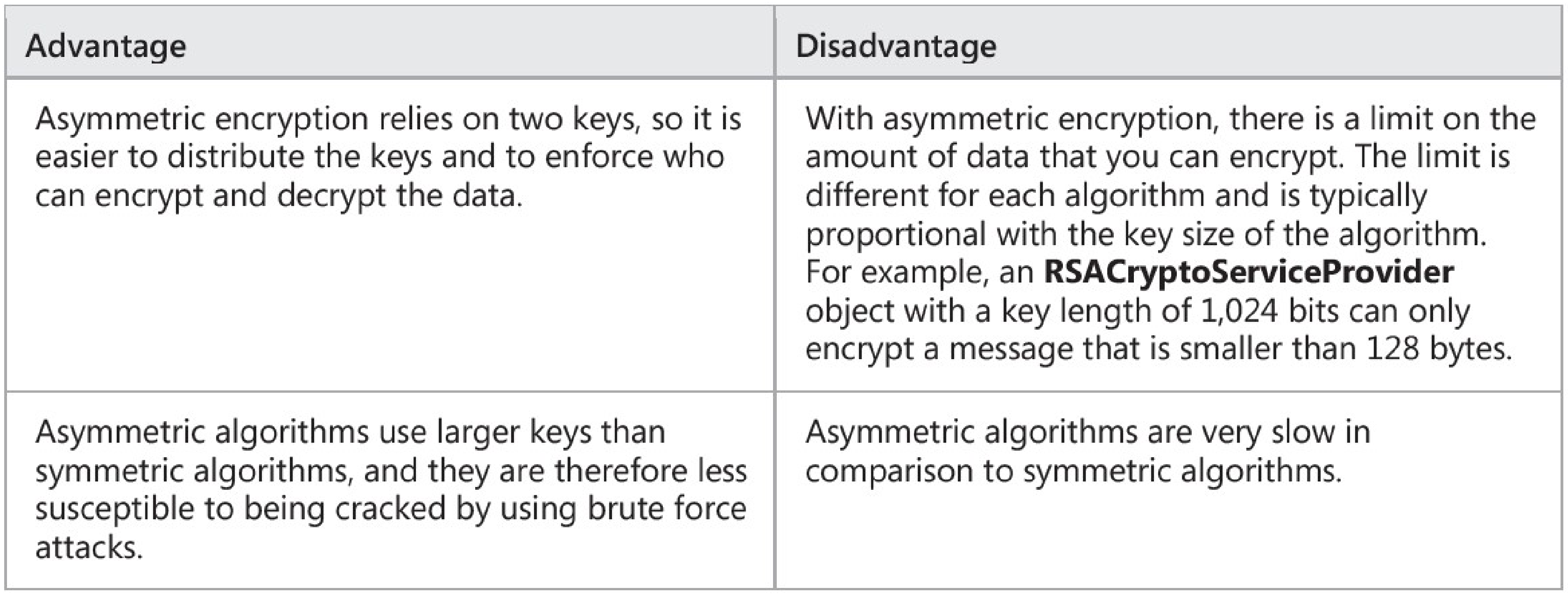
What Is Asymmetric Encryption?

- Unlike symmetric encryption, where one secret key is used to perform both the encryption and the decryption, asymmetric encryption uses:
  a public key to perform the encryption and
  a private key to perform the decryption.
  - The public and private keys are mathematically linked:
    - The private key is used to derive the public key.
    - However, you cannot derive a private key from a public key.
- In a system that uses asymmetric encryption, the public key is made available to any application that requires the ability to encrypt data.
  - the private key is kept safe and is only distributed to applications that require the ability to decrypt the data
- You can also use asymmetric algorithms to sign data.
  - Signing is the process of generating a digital signature so that you can ensure the integrity of the data. When signing data, you
    use the private key to perform the signing and then
    use the public key to verify the data.

What Is Asymmetric Encryption?

- Asymmetric encryption is a powerful encryption technique, but it is not designed for encrypting large amounts of data.
  - If you want to encrypt large amounts of data with asymmetric encryption, you should consider using a combination of asymmetric and symmetric encryption.
    - Example: use asymmetric encryption to encrypt the key, and use symmetric encryption to encrypt the message

- Advantages and Disadvantages of Asymmetric Encryption



Advantage	Disadvantage
Asymmetric encryption relies on two keys, so it is easier to distribute the keys and to enforce who can encrypt and decrypt the data.	With asymmetric encryption, there is a limit on the amount of data that you can encrypt. The limit is different for each algorithm and is typically proportional with the key size of the algorithm. For example, an **RSACryptoServiceProvider** object with a key length of 1,024 bits can only encrypt a message that is smaller than 128 bytes.
Asymmetric algorithms use larger keys than symmetric algorithms, and they are therefore less susceptible to being cracked by using brute force attacks.	Asymmetric algorithms are very slow in comparison to symmetric algorithms.

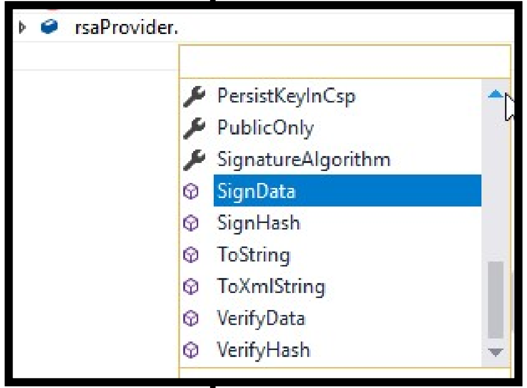
What Is Asymmetric Encryption?

- The **System.Security.Cryptography** namespace includes:
  - The **RSACryptoServiceProvider** class
    - supports both encryption and signing
    - provides an implementation of the RSA algorithm (named after its creators, Ron Rivest, Adi Shamir, and Leonard Adleman)
    - supports key lengths ranging from 384 to 512 bits in 8-bit increments
(up to 16,384 bits in length if you have Microsoft Enhanced Cryptographic Provider installed)
  - The **DSACryptoServiceProvider** class
    - only supports signing
    - provides an implementation of the Digital Signature Algorithm (DSA) algorithm
    - supports keys ranging from 512 to 1,024 bits in 64-bit increments
- See also: http://go.microsoft.com/fwlink/?LinkID=267881

Encrypting Data by Using Asymmetric Encryption

To encrypt and decrypt data asymmetrically

```
var rawBytes = Encoding.Default.GetBytes("hello world..");
var decryptedText = string.Empty;

//default constructor RSACryptoServiceProvider() will generate a set of public and private keys
//      the RSACryptoServiceProvider class exposes members that enable you to
//      export and import the public and private keys.
using (var rsaProvider = new RSACryptoServiceProvider())
{
    var useOaepPadding = true; //use Optimal Asymmetric Encryption Padding (OAEP).

    var encryptedBytes =
       rsaProvider.Encrypt(rawBytes, useOaepPadding);

    var decryptedBytes =
       rsaProvider.Decrypt(encryptedBytes, useOaepPadding);

    decryptedText = Encoding.Default.GetString(decryptedBytes);
    Console.WriteLine(decryptedText); // decryptedText == hello world..
}
```



rsaProvider.

- PersistKeyInCsp
- PublicOnly
- SignatureAlgorithm
- SignData
- SignHash
- ToString
- ToXmlString
- VerifyData
- VerifyHash

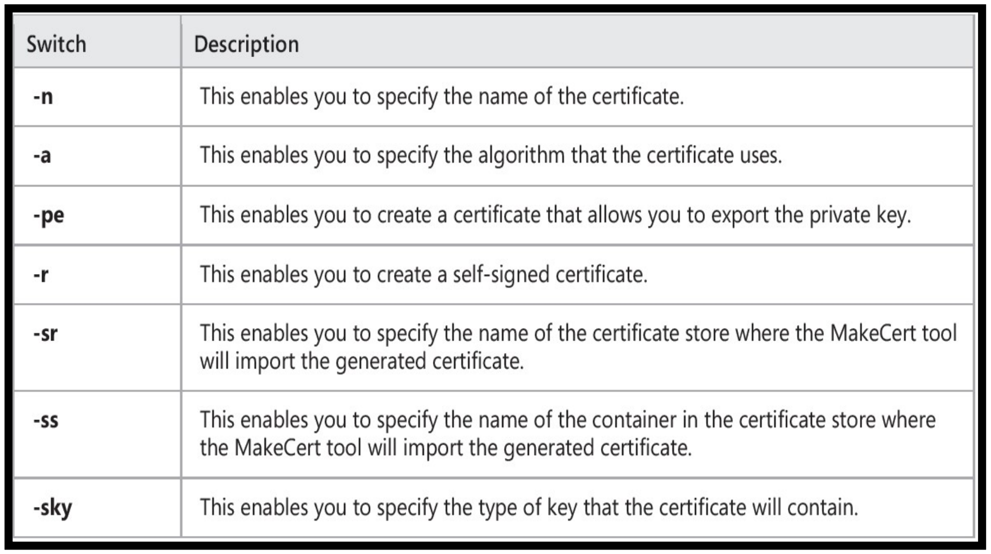
To extract the key information for later use …

```
//use the ExportCspBlob and ImportCspBlob methods to share the public and private keys
var keys = default(byte[]);
var exportPrivateKey = true;
using (var rsaProvider = new RSACryptoServiceProvider())
{
    keys = rsaProvider.ExportCspBlob(exportPrivateKey);
    // Code to perform encryption.
}
var decryptedText = string.Empty;
using (var rsaProvider = new RSACryptoServiceProvider())
{
    rsaProvider.ImportCspBlob(keys);
    // Code to perform decryption.
}
```

Creating X509 Certificates

- An X509 certificate is a digital document that contains information, such as the name of the organization that is supplying the data.
  - X509 certificates can also contain public and private keys, which you can use in the asymmetric encryption process.

- Use MakeCert, a command-line tool, to create your own X509 certificates for development and testing.



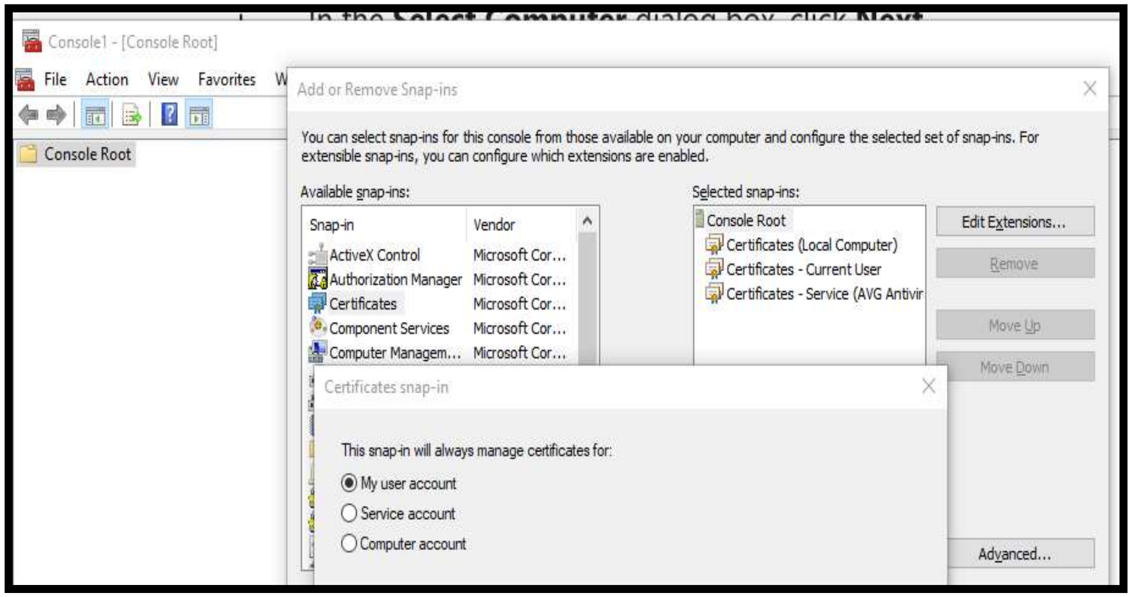
Switch	Description
-n	This enables you to specify the name of the certificate.
-a	This enables you to specify the algorithm that the certificate uses.
-pe	This enables you to create a certificate that allows you to export the private key.
-r	This enables you to create a self-signed certificate.
-sr	This enables you to specify the name of the certificate store where the MakeCert tool will import the generated certificate.
-ss	This enables you to specify the name of the container in the certificate store where the MakeCert tool will import the generated certificate.
-sky	This enables you to specify the type of key that the certificate will contain.

- Example:

```
makecert -n "CN=FourthCoffee" -a sha1 -pe -r -sr LocalMachine -ss my -sky exchange
```

Managing X509 Certificates

- Use the MMC Certificates snap-in to manage the X509 certificates in your certificate stores.
  - You must be logged on as an administrator to see the service account and computer account options when adding the Certificates snap-in to MMC.
  - If you do not see these options, you are logged on as a standard user.



Console1 - [Console Root]

Add or Remove Snap-ins

You can select snap-ins for this console from those available on your computer and configure the selected set of snap-ins. For extensible snap-ins, you can configure which extensions are enabled.

Available snap-ins:

Snap-in	Vendor
ActiveX Control	Microsoft Cor...
Authorization Manager	Microsoft Cor...
Certificates	Microsoft Cor...
Component Services	Microsoft Cor...
Computer Managem...	Microsoft Cor...

Selected snap-ins:

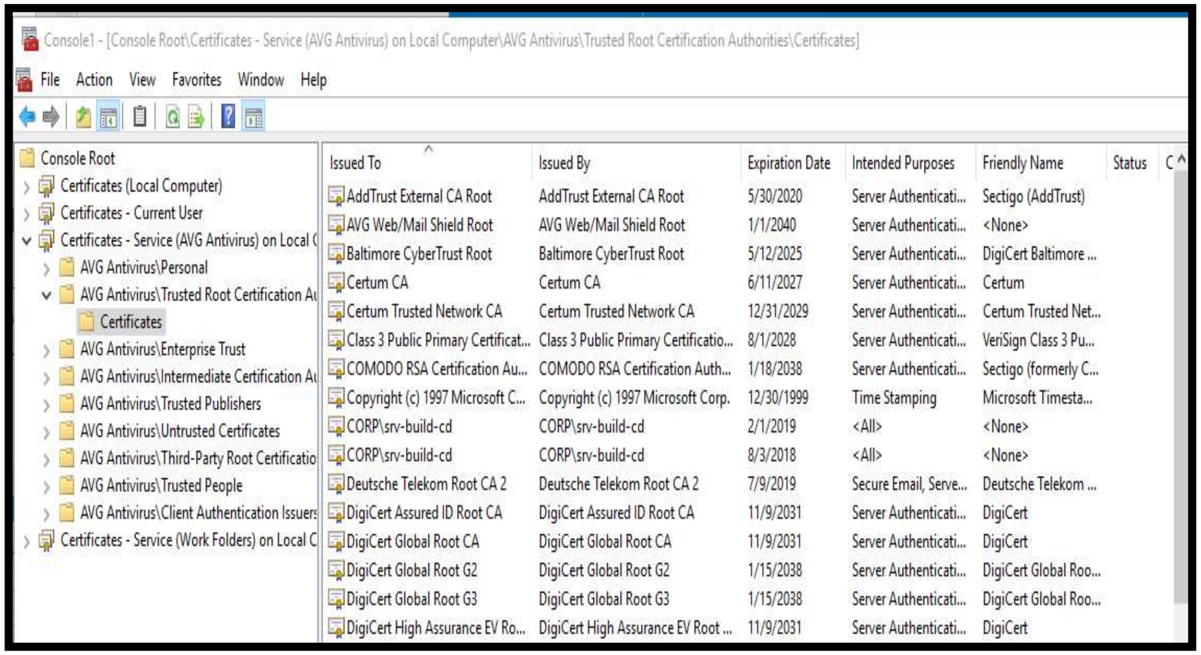
- Console Root
  - Certificates (Local Computer)
  - Certificates - Current User
  - Certificates - Service (AVG Antivir

Certificates snap-in

This snap-in will always manage certificates for:

[✓ My user account] Service account Computer account

- In the **Windows Start** window, search for **mmc.exe**
- on **File** menu, click **Add/Remove Snap-in**.
- click **Certificates**, and then **Add**.
- click either **My user account**, **Service account**, or **Computer account**, ..., then **Finish**, multiple times to select multiple certificates. Then, press **OK**, so you can:
  - View the properties that are associated with any X509 certificate in any of the certificate stores, such as Personal or Trusted Root Certificate Authorities stores.
  - Export an X509 certificate from a certificate store to the file system.
  - Manage the private keys that are associated with an X509 certificate.
  - Issue a request to renew an existing X509 certificate.



Console1 - [Console Root\Certificates - Service (AVG Antivirus) on Local Computer\AVG Antivirus\Trusted Root Certification Authorities\Certificates]

File
Action
View
Favorites
Window
Help
Console Root
Certificates (Local Computer)
Certificates - Current User
Certificates - Service (AVG Antivirus) on Local (
AVG Antivirus\Personal
AVG Antivirus\Trusted Root Certification Au
Certificates
AVG Antivirus\Enterprise Trust
AVG Antivirus\Intermediate Certification Au
AVG Antivirus\Trusted Publishers
AVG Antivirus\Untrusted Certificates
AVG Antivirus\Third-Party Root Certificatio
AVG Antivirus\Trusted People
AVG Antivirus\Client Authentication Issuers
Certificates - Service (Work Folders) on Local C

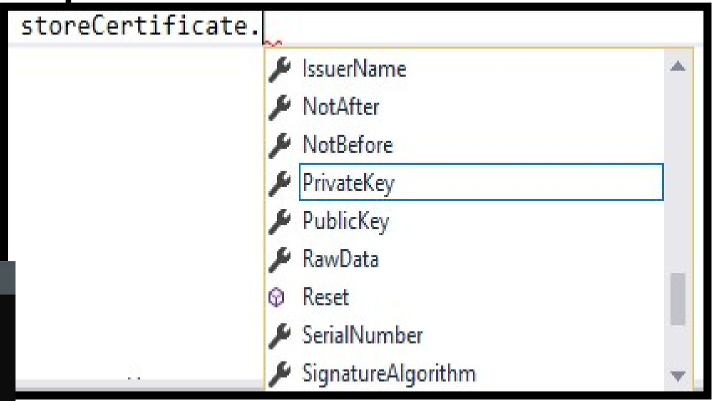
Issued To	Issued By	Expiration Date	Intended Purposes	Friendly Name	Status	C
AddTrust External CA Root	AddTrust External CA Root	5/30/2020	Server Authenticati...	Sectigo (AddTrust)		
AVG Web/Mail Shield Root	AVG Web/Mail Shield Root	1/1/2040	Server Authenticati...	<None>		
Baltimore CyberTrust Root	Baltimore CyberTrust Root	5/12/2025	Server Authenticati...	DigiCert Baltimore ...		
Certum CA	Certum CA	6/11/2027	Server Authenticati...	Certum		
Certum Trusted Network CA	Certum Trusted Network CA	12/31/2029	Server Authenticati...	Certum Trusted Net...		
Class 3 Public Primary Certificat...	Class 3 Public Primary Certificatio...	8/1/2028	Server Authenticati...	VeriSign Class 3 Pu...		
COMODO RSA Certification Au...	COMODO RSA Certification Auth...	1/18/2038	Server Authenticati...	Sectigo (formerly C...		
Copyright (c) 1997 Microsoft C...	Copyright (c) 1997 Microsoft Corp.	12/30/1999	Time Stamping	Microsoft Timesta...		
CORP\srv-build-cd	CORP\srv-build-cd	2/1/2019	<All>	<None>		
CORP\srv-build-cd	CORP\srv-build-cd	8/3/2018	<All>	<None>		
Deutsche Telekom Root CA 2	Deutsche Telekom Root CA 2	7/9/2019	Secure Email, Serve...	Deutsche Telekom ...		
DigiCert Assured ID Root CA	DigiCert Assured ID Root CA	11/9/2031	Server Authenticati...	DigiCert		
DigiCert Global Root CA	DigiCert Global Root CA	11/9/2031	Server Authenticati...	DigiCert		
DigiCert Global Root G2	DigiCert Global Root G2	1/15/2038	Server Authenticati...	DigiCert Global Roo...		
DigiCert Global Root G3	DigiCert Global Root G3	1/15/2038	Server Authenticati...	DigiCert Global Roo...		
DigiCert High Assurance EV Ro...	DigiCert High Assurance EV Root ...	11/9/2031	Server Authenticati...	DigiCert		

Managing Encryption Keys

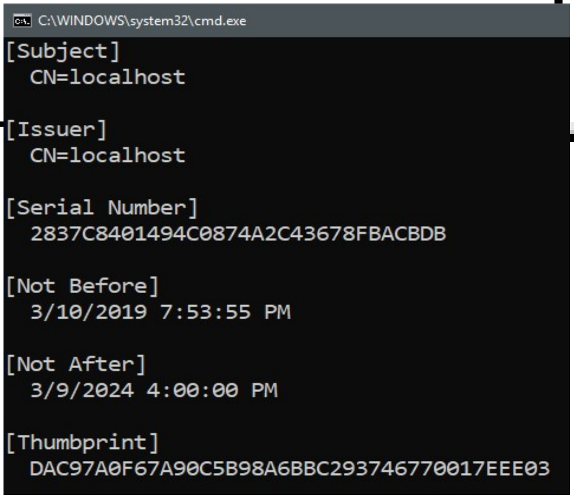
The **System.Security.Cryptography.X509Certificates** namespace contains classes that enable access to the certificate store and certificate metadata

- **X509Store** class enables you to access a certificate store and perform operations, such as finding an X509 certificate with a particular name.
- **X509Certificate2** class enables you create an in-memory representation of an X509 certificate that currently exists in a certificate store.
- **PublicKey** class enables you to manipulate the various pieces of metadata that are associated with an X509 certificate's public key, such as the public key's value.

```
var store = new X509Store(StoreName.My, StoreLocation.LocalMachine);

store.Open(OpenFlags.ReadOnly);

foreach (var storeCertificate in store.Certificates)
{
    // Code to process each certificate.
    Console.WriteLine(storeCertificate); //for example
}

store.Close();
```



```
storeCertificate.
```



```
[Subject]
  CN=localhost

[Issuer]
  CN=localhost

[Serial Number]
  2837C8401494C0874A2C43678FBACBDB

[Not Before]
  3/10/2019 7:53:55 PM

[Not After]
  3/9/2024 4:00:00 PM

[Thumbprint]
  DAC97A0F67A90C5B98A6BBC293746770017EEE03
```

Module Review and Takeaways

- **Question:** Fourth Coffee wants you to implement an encryption utility that can encrypt and decrypt large image files. Each image will be more than 200 megabytes (MB) in size. Fourth Coffee envisages that only a small internal team will use this tool, so controlling who can encrypt and decrypt the data is not a concern. Which of the following techniques will you choose?
  - ( )Option 1: Symmetric encryption
  - ( )Option 2: Asymmetric encryption
  - ( )Option 3: Hashing

- **Question:** Is the following statement true or false? Asymmetric encryption uses a public key to encrypt data.
  - ( )False
  - ( )True

Course Evaluation