Cifrado y Descifrado de información con AES (Advanced Encryption Standard)

Práctica 3

1. Objetivo

El objetivo de esta práctica es afianzar la comprensión de los conceptos teóricos sobre cifrado de información con algoritmos simétricos y saber utilizarlos en un entorno de desarrollo. Para ello, el alumno comenzará usando alguna herramienta que permita cifrar y descifrar ficheros utilizando el algoritmo AES. Seguidamente el alumno debe utilizar las clases disponibles en el entorno Visual Studio y .NET Framework en el espacio de nombres System.Security.Cryptography. Ahora, el objetivo será cifrar, y posteriormente descifrar, bloques de información con el algoritmo **simétrico** AES, tal como lo hace la herramienta utilizada previamente.

2. Herramientas de cifrado/descifrado de ficheros con el algoritmo AES

El alumno debe usar alguna herramienta de cifrado y descifrado de información, que actualmente usan AES y/u otros algoritmos. Hay muchas herramientas disponibles en Internet.

Una herramienta bien establecida es **AxCrypt**, que ha estado en continua evolución desde 2001. La versión AES-128 es gratuita y la versión AES-256 es de pago.

http://www.axcrypt.net/es/

Una herramienta muy sencilla es **AES Crypt**, que es de código completamente abierto y gratuita.

https://www.aescrypt.com/

Utilizar esta herramienta debido a su gran sencillez. Inicialmente descargar la versión para consola, que es un ejecutable que no necesita instalación. Descargar la guía del usuario en PDF. El uso mediante la línea de comandos es igual que en Linux y está en la última sección de la guía.

Este programa ilustra lo que se debería conseguir al realizar las secciones 5 y 6 de esta práctica.

Para analizar cómo funciona una herramienta de cifrado/descifrado basada en una Interfaz Gráfica (GUI) descargar la versión correspondiente. Como requiere instalación hay que utilizar la **máquina virtual de prácticas**. Usar la herramienta para realizar operaciones básicas de cifrado y descifrado de ficheros.

3. Cifrado y descifrado de arrays de bytes (uno o más bloques AES)

La tarea inicial a realizar en la práctica consistirá en un programa que primero cifra un texto plano (array de bytes) y almacena el texto cifrado (array de bytes) en un fichero. Después lee el array de bytes del fichero y lo descifra para obtener nuevamente el texto plano.

El **programa** deberá realizar las tareas siguientes:

1) Declarar en el método estático Main() la variable int TamClave = 16 ó 24 ó 32 y el array de bytes Clave de tamaño TamClave. Da valor a la clave con la siguiente línea de código:

```
for (int i = 0; i < Clave.Length; i++) Clave[i] = (byte)(i%256);
```

El contenido de la Clave serán los bytes 0x00, 0x01, y así sucesivamente hasta completar los bytes necesarios en función del tamaño de la clave.

2) Declarar un vector de inicialización VI que será un array de 16 bytes, con los bytes 0xA0 a 0xAF.

```
for (int i = 0; i < VI.Length; i++) VI[i] = (byte)((i+160)%256);
```

3) Declarar un array de 48 bytes, denominado TextoPlano usando el código siguiente:

```
byte[] TextoPlano =
{ 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
  0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
  0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
  0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
  0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
  0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F
};
```

El TextoPlano tiene exactamente el tamaño de tres bloques de información que cifra AES. En este caso los tres bloques son idénticos. Usando solo las dos primeras líneas de números, TextoPlano tiene el tamaño de 1 bloque AES. Usando solo las tres primeras líneas de números, TextoPlano tiene el tamaño de 1,5 bloques AES, etc.

4) Crear un objeto proveedor de servicios criptográficos AES.

Para cifrar con el algoritmo AES hay que crear un objeto a partir de la clase apropiada. En el entorno .NET hay dos posibilidades: La clase AesCryptoServiceProvider y la clase AesManaged.

Crear un objeto de la clase **AesCryptoServiceProvider** para cifrar y descifrar información.

5) Mostrar en la consola las características básicas del objeto proveedor: Blocksize, Keysize, Padding, Mode.

6) Asignar a las características Keysize, Padding y Mode nuevos valores. Comprobar que la asignación se ha realizado, recuperando las características y mostrándolas en la consola. Incluir en el programa, mediante comentarios, los valores que es posible asignar a estas características.

7) Generar una clave simétrica aleatoria para el proveedor utilizando el método **Generatekey()**. Mostrar la clave generada en la consola. Aunque se debería guardar y usar esta clave aleatoria, en esta práctica es más conveniente usar una de las claves deterministas generadas en el programa.

8) Asignar al proveedor la clave del tamaño elegido definida al principio del programa, leerla y mostrar la lectura en la consola para comprobar que la asignación se ha realizado correctamente.

9) Generar un vector de inicialización aleatorio para el proveedor utilizando el método **GenerateIV()**. Mostrar el vector generado en la consola. Aunque se debería guardar y usar este vector de inicialización, en esta práctica es más conveniente usar el vector determinista generado en el programa.

10) Asignar al proveedor el vector de inicialización definido al principio del programa, leerlo y mostrar la lectura en la consola para comprobar que la asignación se ha realizado correctamente.

Proceso de cifrado de un array de bytes

1) Definir un flujo de datos (Stream) para almacenar el texto cifrado. Si se desea almacenar en memoria hay que usar un MemoryStream y si se desea almacenar en un fichero hay que usar un FileStream. En esta práctica, el texto cifrado se almacenará en un fichero y por tanto debes crear un objeto de la clase **FileStream**, que permita escribir en un fichero y que se denominará "zz_TextoCifrado.bin". Usar el modo **Create** para el fichero, pues así siempre se crea un fichero nuevo. El modo de acceso será **Write** y el de compartición **None**.

2) Para que el objeto proveedor AES pueda cifrar información es necesario crear un objeto cifrador definido por la interfaz **ICryptoTransform**. Para crear el cifrador usar el método **CreateEncryptor()** de la clase **AesCryptoServiceProvider**. Observar la técnica utilizada en .NET de definir una interfaz (≈clase abstracta) genérica para realizar transformaciones criptográficas (cifrar y descifrar) y crear el objeto que realiza la transformación a partir de la interfaz usando un método proporcionado por el proveedor de cada algoritmo, AES, Rindjael, etc.

3) El objeto que realiza la transformación criptográfica, el cifrado en este caso, no funciona solo, sino que realiza su trabajo para un objeto de una clase Stream, concretamente, para un objeto de la clase **CryptoStream**.

El diseño del entorno de criptografía de .NET está orientado a trabajar con Streams. La idea es encadenar varios objetos Stream de modo que la salida de uno constituye la entrada del siguiente en la cadena de Streams. No es necesario usar una variable para almacenar información entre la salida de un objeto y la entrada del siguiente. Para el cifrado, la salida de un objeto CryptoStream (que cifra la información) alimenta la entrada de un objeto FileStream (que almacena la información cifrada en un fichero).

Crear un objeto de la clase CryptoStream. El constructor utiliza tres parámetros. El primero es el Stream con el que opera el CryptoStream, en este caso el FileStream de escritura creado previamente. El segundo es el objeto que realiza la transformación criptográfica de cifrado y que se ha creado previamente. El tercero es el modo de trabajo del CryptoStream con el FileStream

definido en el primer parámetro y que puede ser Read o Write. En este caso, el modo de trabajo es Write para que el CryptoStream escriba la información cifrada en el FileStream.

4) Cifrar la información llamando al método **Write()** del objeto CryptoStream creado previamente y pasando como primer parámetro uno de los arrays denominados TextoPlano declarados al principio del programa.

5) Para que los bytes se vuelquen al Stream de salida hay que llamar al método **Flush()**. Finalmente hay que cerrar el CryptoStream llamando al método **Close()**. Al llamar a Close() se hace previamente el volcado del contenido del CryptoStream.

6) Después hay que liberar los recursos asignados al objeto cifrador invocando a su método **Dispose()**. No obstante, si se va a usar el mismo objeto como descifrador en la siguiente sección de la práctica, no debe invocarse el método Dispose().

7) Finalmente hay que cerrar el FileStream llamando a su método **Close()**.

Comprobación de que el texto cifrado ha sido creado

Verificar que aparece el fichero "zz_TextoCifrado.bin" en el directorio ..\bin\Debug\ del proyecto correspondiente. Visualizar el contenido del fichero con un programa visualizador de ficheros binarios como HxD.

Proceso de descifrado de un array de bytes

1) Declarar un array de bytes denominado TextoDescifrado de la longitud suficiente para almacenar el texto que hay que descifrar. Como tamaño del array se puede utilizar el tamaño del fichero que contiene la información cifrada. Si ese tamaño es mayor que el del texto plano original, sobrarán bytes al final del array.

2) Crear un objeto de la clase **FileStream**, que permita leer el fichero "zz_TextoCifrado.bin". Usar el modo **Open** para el fichero, pues así siempre debe existir el fichero. El modo de acceso será **Read** y el de compartición **None**.

3) Para que el objeto proveedor AES pueda descifrar información es necesario crear un objeto descifrador definido por la interfaz **ICryptoTransform**. Para crear el descifrador usar el método **CreateDecryptor()** de la clase **AesCryptoServiceProvider**.

4) El objeto que realiza la transformación criptográfica, el descifrado en este caso, no funciona solo, sino que realiza su trabajo para un objeto de una clase Stream, concretamente, para un objeto de la clase **CryptoStream**. Para el descifrado, la salida de un objeto FileStream (que lee la información cifrada en un fichero) alimenta la entrada de un objeto CryptoStream (que descifra la información).

Crear un objeto de la clase CryptoStream para realizar la transformación criptográfica de descifrado. El constructor utiliza tres parámetros. El primero es el Stream con el que opera el CryptoStream, en este caso el FileStream de lectura creado previamente. El segundo es el objeto que realiza la transformación criptográfica de descifrado y que se ha creado previamente. El tercero es el modo de trabajo del CryptoStream con el FileStream definido en el primer parámetro y que puede ser Read o Write. En este caso, el modo de trabajo es Read para que el CyptoStream lea la información cifrada del FileStream.

5) Descifrar la información llamando al método **Read()** del objeto CryptoStream creado previamente y pasando como primer parámetro el array denominado TextoDescifrado, declarado al principio de esta sección del programa.

6) Para asegurar el vaciado y el total procesamiento de los bytes recibidos por el CryptoStream hay que llamar a su método **Flush()**. Finalmente hay que cerrar el CryptoStream llamando al método **Close()**. Al llamar a Close() se hace previamente el vaciado del contenido del CryptoStream.

7) Después hay que liberar los recursos asignados al objeto descifrador invocando a su método **Dispose()**.

8) Finalmente hay que cerrar el FileStream llamando a su método **Close()**.

9) Mostrar los bytes descifrados en la consola. Para representar solo los bytes que corresponden a la información descifrada se debe usar el número de bytes devueltos por el método Read del CryptoStream.

4. Pruebas con el programa

Encadenamiento de cifradores

1) Cifrar y descifrar una información que debe dividirse en varios bloques elementales de cifrado, probando el funcionamiento de los diversos modos de encadenamiento de bloques.

Por ejemplo, si se cifra un texto plano de 48 bytes compuesto por tres bloques de 16 bytes idénticos, usando el modo ECB debemos obtener un texto cifrado de 48 bytes con tres bloques de 16 bytes que también serán idénticos. Pero si utilizamos otro modo, como CBC, esto no ocurrirá. Para hacer esta prueba, no utilices relleno para el último bloque, esto es, cifra una información cuya longitud en bytes sea un múltiplo exacto de 16.

Modos de relleno

2) Cambiar el modo de relleno (padding) utilizado para completar la información a cifrar hasta un número de bytes que sea un múltiplo exacto del tamaño del bloque utilizado por el algoritmo AES que es de 128 bits (16 bytes). En esta prueba, utiliza un texto plano de 40 bytes, que equivalen a 2,5 bloques de 16 bytes.

Comprueba lo que ocurre al utilizar un PaddingMode.None.

Después utiliza PaddingMode.Zeros como relleno. Comprueba que el número de bytes que devuelve el método Read() del CryptoStream es 48, interpretando el relleno final de ceros como si fuesen auténticos datos del texto plano.

Seguidamente utiliza los rellenos .ANSIX923, .ISO10126, y .PKCS7. Comprueba que el método Read() devuelve 40 bytes, que es el tamaño del texto plano original, sin el relleno.

Tamaños de clave

3) Cambiar el tamaño de la clave de cifrado, probando no solo el tamaño estándar de 128 bits, sino también los de 192 y 256 bits. ¿Hay que hacer algún cambio adicional en el programa?

Cambiar el tipo de proveedor de servicios criptográficos

4) Utilizar en el programa la clase AesManaged en vez de la clase AesCryptoServiceProvider. Comprueba que tan solo hay que cambiar el tipo de objeto proveedor de servicios AES, pues todo el resto del programa funciona sin necesidad de modificaciones.

5. Cifrado y descifrado de cadenas de caracteres

Haz una variación del programa anterior que cifre una cadena de caracteres y la descifre.

En C# las cadenas de caracteres (string) se almacenan en formato Unicode. Pero no se suele almacenar en archivos los caracteres Unicode (16 bits) directamente, sino que se transforman a una codificación más eficiente como ASCII, UTF-8, etc. Del mismo modo, para almacenar la información cifrada, lo habitual es cifrar la codificación ASCII, UTF-8, etc., y no la Unicode.

Para cifrar una cadena de caracteres hay que declarar un StreamWriter que use (escriba en) el CryptoStream de cifrado. Observar que tras esta declaración hemos encadenado tres Streams:

StreamWriter > CryptoStream > FileStream

Luego, basta con escribir la cadena a cifrar en el StreamWriter usando el método WriteLine() del propio StreamWriter. Finamente cerrar el StreamWriter usando su método Close().

Para descifrar hay que declarar un StreamReader que use (lea de) el CryptoStream de descifrado. Observar que tras esta declaración hemos encadenado tres Streams:

FileStream > CryptoStream > StreamReader

Luego, basta con leer del StreamReader usando el método ReadToEnd() del propio StreamReader y cargar el string devuelto en la cadena en la que se desea almacenar el texto descifrado. Finamente cerrar el StreamReader usando su método Close().

6. Cifrado y descifrado de ficheros

Ahora debes realizar un par de programas: uno para cifrar un fichero cualquiera y el otro para descifrar el fichero cifrado.

Si el fichero es pequeño, puedes cargarlo completo en un array en memoria y cifrar el contenido del array en otro fichero. Haz lo mismo para descifrar.

Si el fichero es grande, tendrás que mantener abiertos dos ficheros: el fichero de entrada (plano) para lectura y el fichero de salida (cifrado) para escritura. Tendrás que realizar un bucle en que leas un bloque del fichero de entrada en un búfer de memoria y lo escribas seguidamente en el fichero de salida, mientras que la longitud del búfer leído sea mayor que cero. Realiza un proceso similar para descifrar.

7. Generar claves a partir de contraseñas

Cuando un usuario cifra información debe utilizar una clave que nunca debe estar incluida en el programa cifrador para evitar que un atacante la descubra usando técnicas de desensamblado de código. El usuario que recibe la información cifrada debe utilizar la misma clave que también deberá introducir en el programa descifrador.

Las claves NO deben incluirse en los programas NI almacenarse en ficheros, sino que deben residir solamente en la memoria de los usuarios. Pero es prácticamente imposible que un usuario memorice claves AES de 16, 24 o 32 bytes que tengan una apariencia aleatoria.

Por ello, lo normal es derivar los bytes de las claves AES de una contraseña que el usuario pueda memorizar. El entorno .NET proporciona la clase **Rfc2898DeriveBytes** para realizar esta tarea.

Aunque los conceptos de creación y comprobación de contraseñas se verán en una práctica posterior sobre autenticación de usuarios, se explica a continuación el uso de esta clase.

Crea una nueva solución de Visual Studio.

Declara el array Contra y asígnale una cadena de caracteres (la contraseña) leída del teclado con el método **ReadLine()** de la clase Console. Declara el array de bytes Sal y asígnale un valor, por ejemplo de esta forma:

```
byte[] Sal = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 };
```

Este vector Sal tiene una longitud mínima. En una utilización real hay que usar un vector más largo, por ejemplo de 32 o 64 bytes, y sin un patrón determinado en sus bytes.

Crea el objeto Generador de la clase Rfc2898DeriveBytes pasándole tres parámetros en el constructor: Contra, Sal y el entero 1000. Este entero especifica el número de iteraciones internas a realizar por el algoritmo y se puede cambiar.

Para obtener los N bytes que se utilizarán como clave, declara el array de bytes Clave y asígnale lo que devuelve la llamada al método **GetBytes(N)** del objeto Generador. Para AES se usará un valor apropiado para N de 16, 24 o 32 bytes. Muestra la clave generada en la consola.

También es posible derivar los 16 bytes del vector de inicialización de la contraseña. Para ello, declara el array de bytes VI y asígnale lo que devuelve una nueva llamada a GetBytes(16). Muestra el vector de inicialización generado en la consola.

Ahora, entre la derivación de la clave y la derivación del vector de inicialización, llama al método **Reset()** del objeto Generador. Comprueba que la secuencia de bytes del vector de inicialización se reinicia y coincide con la secuencia de bytes de la clave.

Declara un nuevo objeto Generador2, inicializándolo con los mismos parámetros que el objeto Generador anterior. Carga en un nuevo array de bytes, invocando a su método GetBytes(64).

Comprueba que cualquier objeto de la clase **Rfc2898DeriveBytes** genera la misma secuencia de bytes para la misma contraseña, la misma Sal y el mismo número de iteraciones.

Ahora todo este conocimiento adquirido se puede utilizar al principio de cualquiera de los programas de cifrado y descifrado desarrollados previamente en esta práctica.

Se puede optar por derivar solamente la clave de la contraseña. También se puede optar por derivar ambos, la clave y el VI, de la contraseña. En este caso se puede usar un solo objeto **Rfc2898DeriveBytes** o bien se puede usar un objeto para la clave y otro para el VI, cada objeto con una inicialización diferente.

8. Cifrado de discos

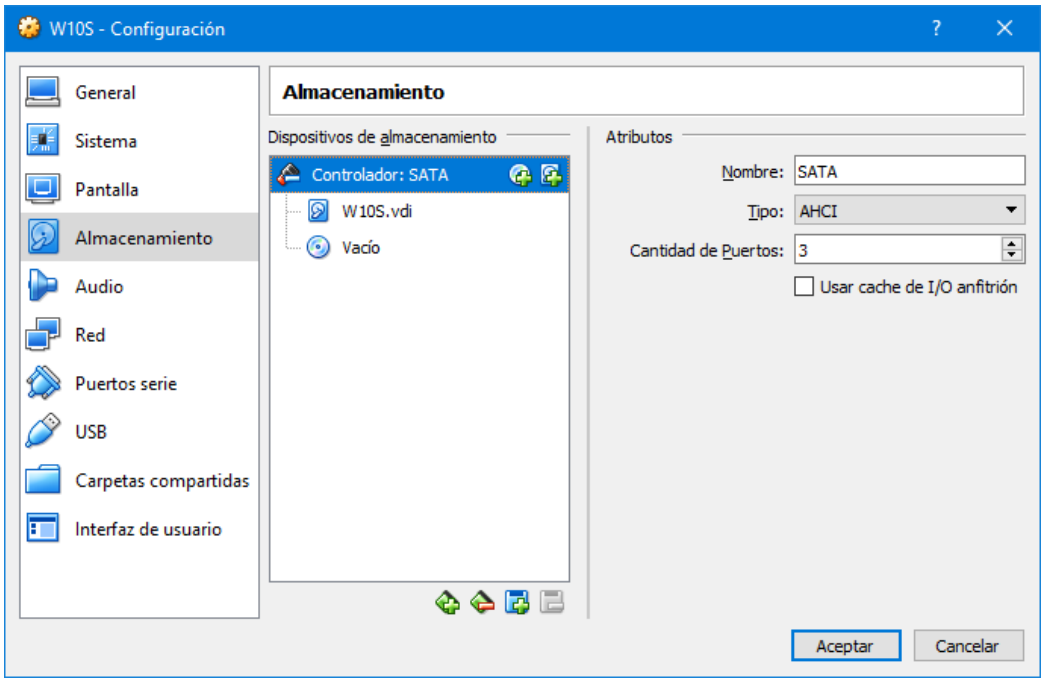
Para almacenar información muy confidencial es conveniente utilizar un disco cifrado. De este modo si un atacante roba el dispositivo con el disco no podrá acceder a la información.

Una de las tecnologías disponibles para cifrar una unidad de disco es BitLocker, ya que está integrada con el sistema operativo Windows.

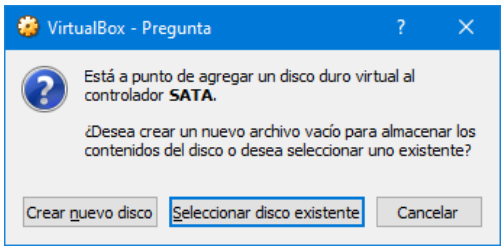
Crea un disco adicional de poca capacidad (16 GB) para la MV de prácticas. Puedes denominarlo DCnnnnnn (Disco Cifrado), siendo nnnnnn el número de matrícula del alumno. También se puede utilizar DCnombrealumno. Lo único importante es que el disco quede perfectamente identificado.

Con la MV apagada, inicia el gestor de MV de VirtualBox y selecciona la MV de prácticas.

Selecciona el controlador SATA de discos de la MV, tal como se muestra en la ventana siguiente:



Pulsa el botón del disco que tiene dibujado un símbolo + encima y aparece la ventana:



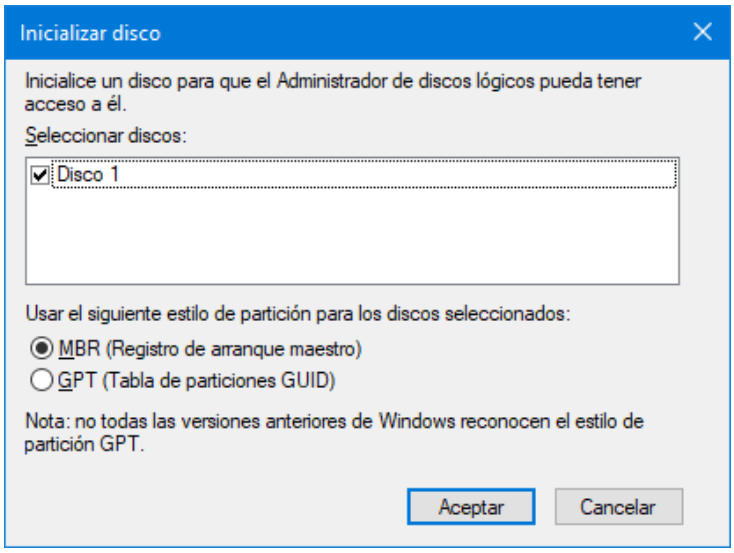
Pulsa el botón “Crear nuevo disco”.

Selecciona una disco DVI, Reservado dinámicamente, y de tamaño máximo 16 GB.

Ahora hay que crear el disco en el sistema operativo Windows. Arranca la MV y en el SO haz:

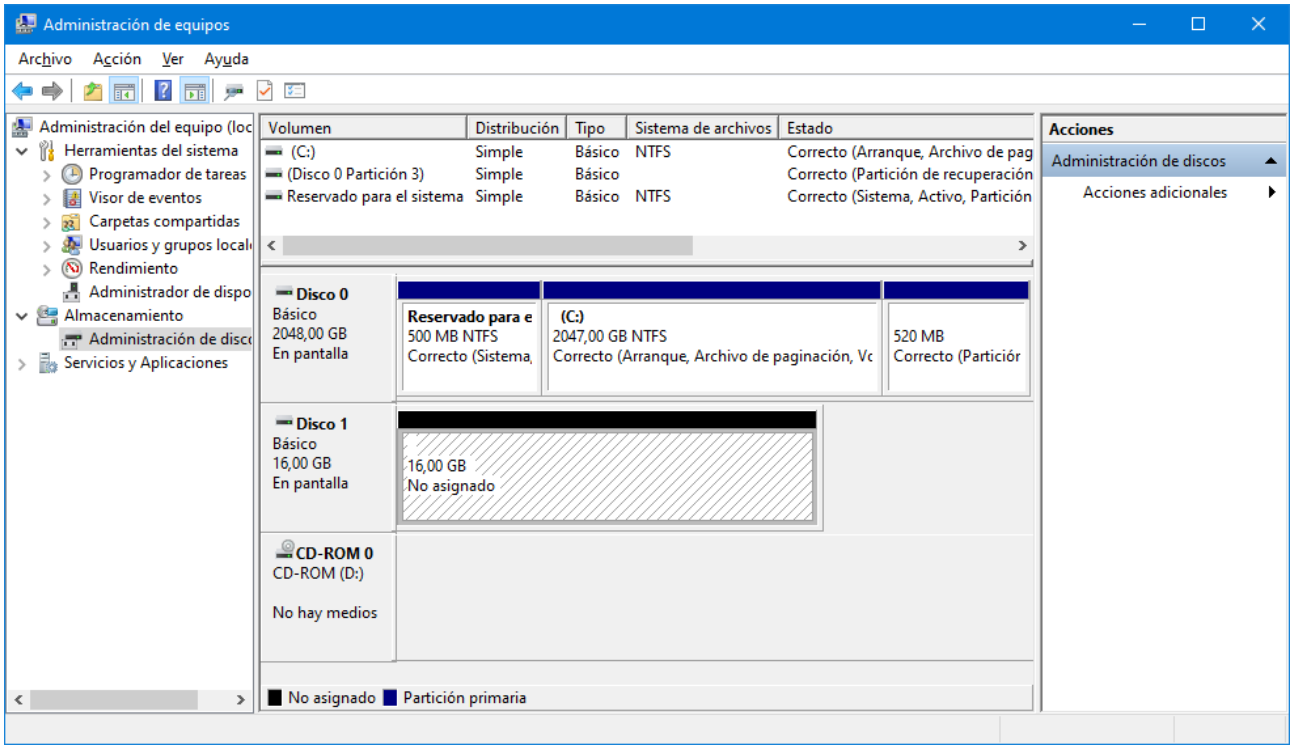
Panel de Control > Herramientas administrativas > Administración de equipos > Almacenamiento > Administración de discos

Al arrancar la herramienta aparece esta ventana emergente:



Acepta la inicialización del disco que se propone en la ventana.

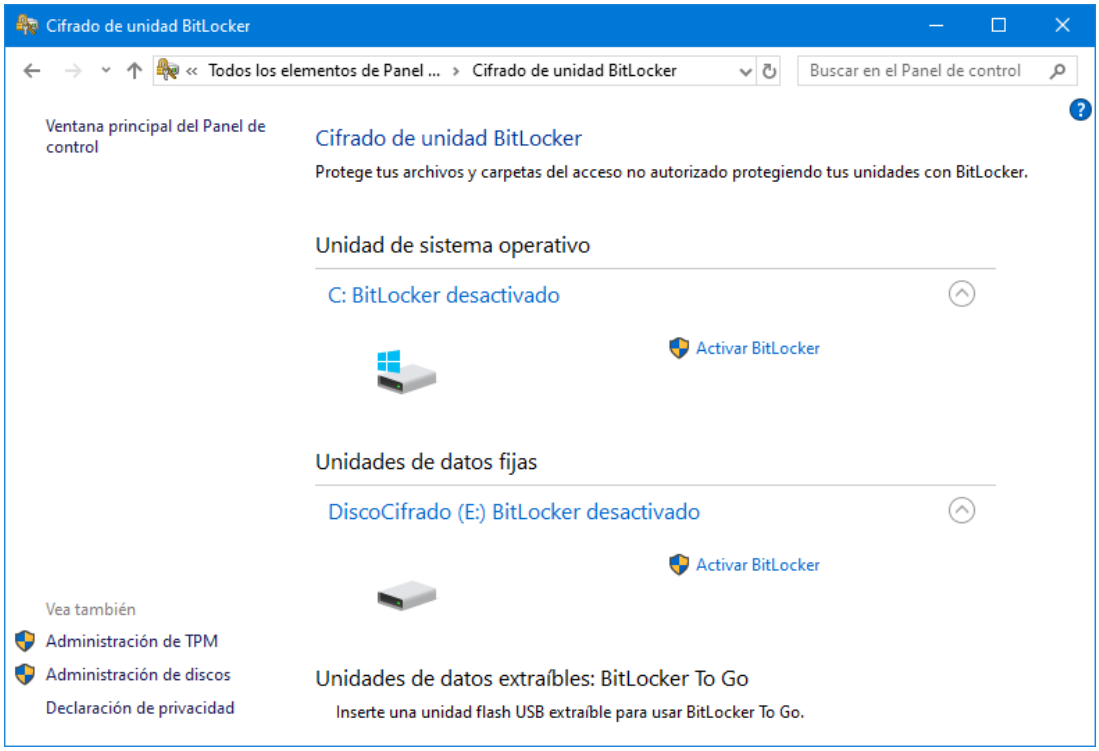
La Consola de Administración muestra el aspecto siguiente:



Haz clic-izda sobre el espacio de 16GB no asignado para seleccionar al disco y luego haz clic-dcha. En el menú emergente selecciona “Nuevo volumen simple” y aparece el asistente para generar un volumen. Puedes denominarlo DiscoCifrado.

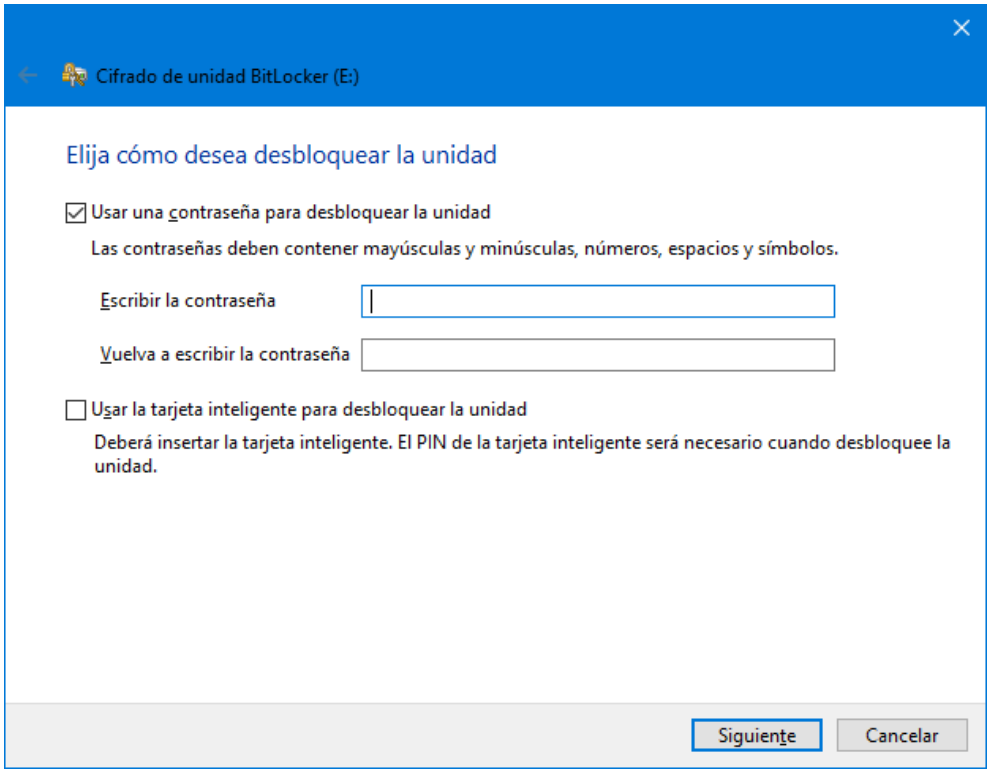
Sal de Administrador y reinicia la MV para que aparezca el nuevo disco en el explorador de archivos.

Para activar BitLocker se puede hacer: Panel de control > Cifrado de unidad BitLocker y aparece:



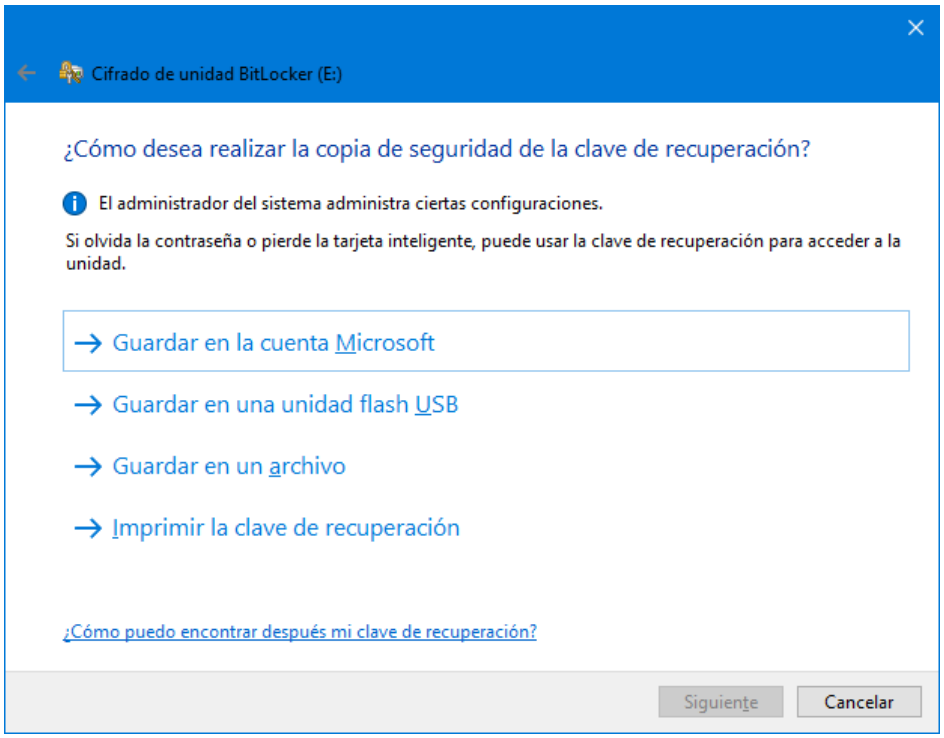
También en el propio explorador de ficheros se puede seleccionar una unidad de disco y hacer clic-derecho. En el menú emergente que aparece esta la opción "Activar BitLocker".

Al activarlo aparece esta ventana:



Utiliza una contraseña para desbloquear la unidad. Se recomienda usar **BitLocker2019** para que si se olvida se pueda volver a consultar en este guion de prácticas.

Ahora hay que indicar como se desea realizar la copia de seguridad de la clave de recuperación, para el caso de que se olvide la contraseña.

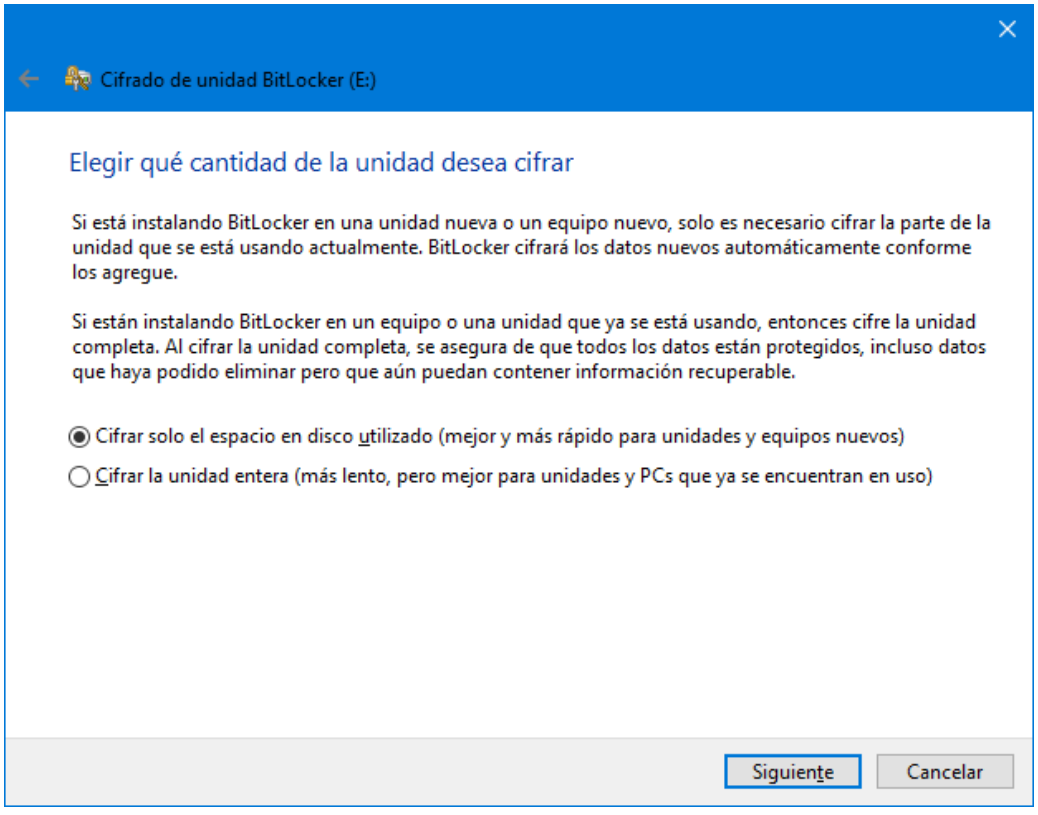


Selecciona la opción “Guardar en un archivo”. En este ejemplo se crea el archivo siguiente:

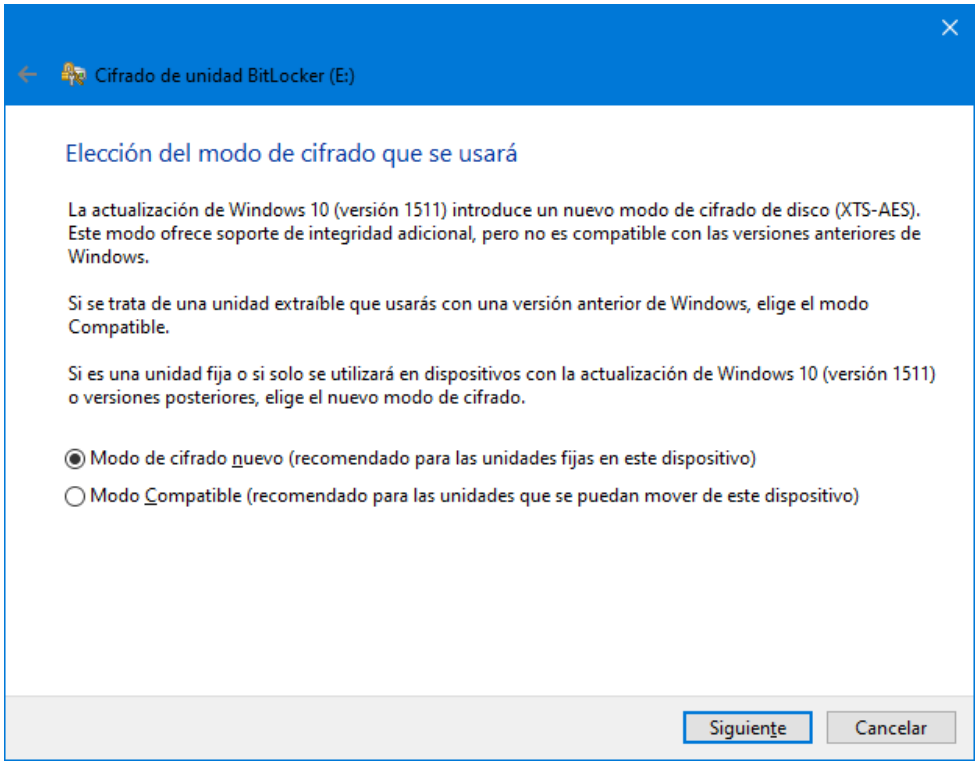
“Clave de recuperación de BitLocker 17E05E75-C180-49A0-931E-2E5AA28CCA99.TXT”

La secuencia alfanumérica que aparece es el identificador del disco cifrado.

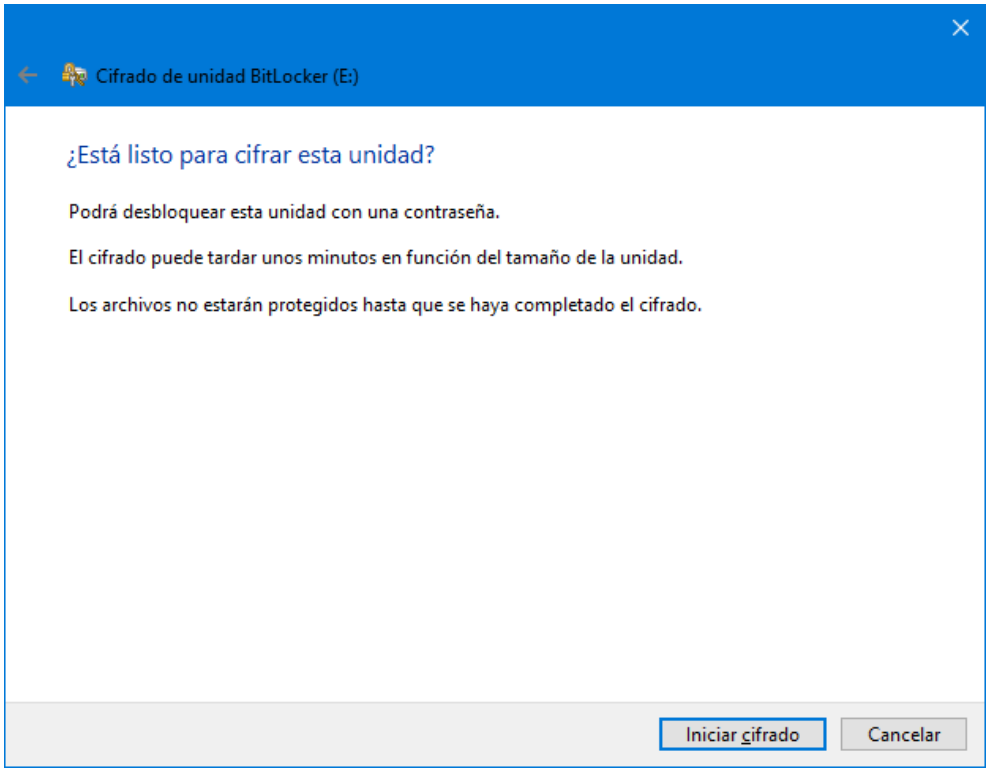
Elegir la cantidad de la unidad que se desea cifrar:



Seleccionar el modo de cifrado a usar:

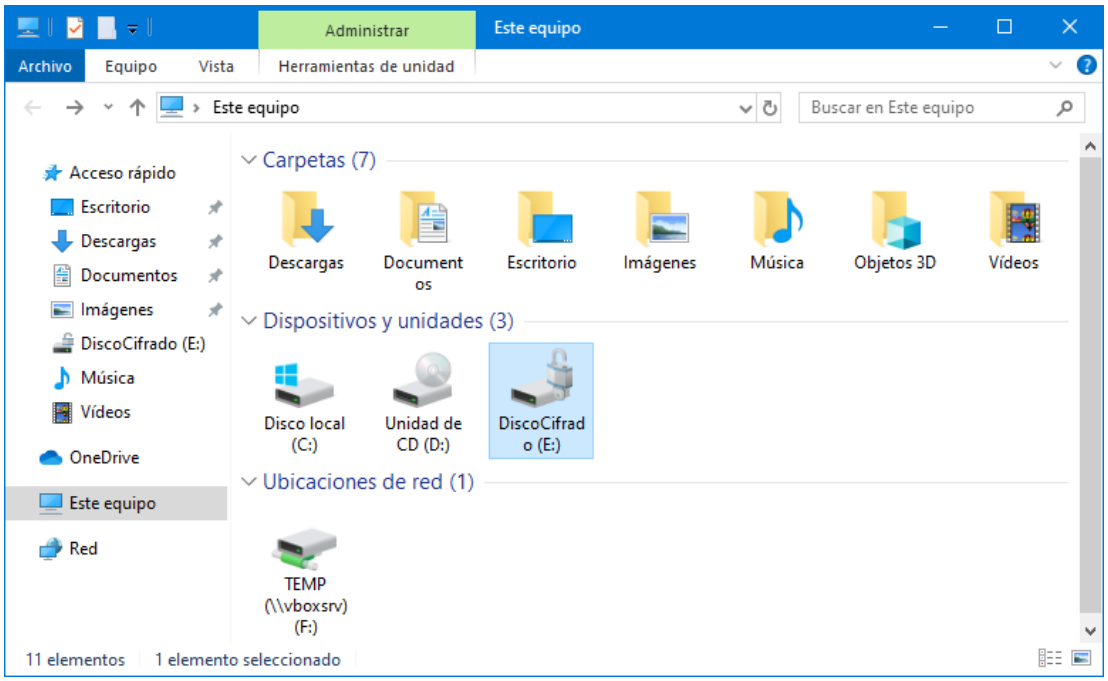


E iniciar el cifrado:



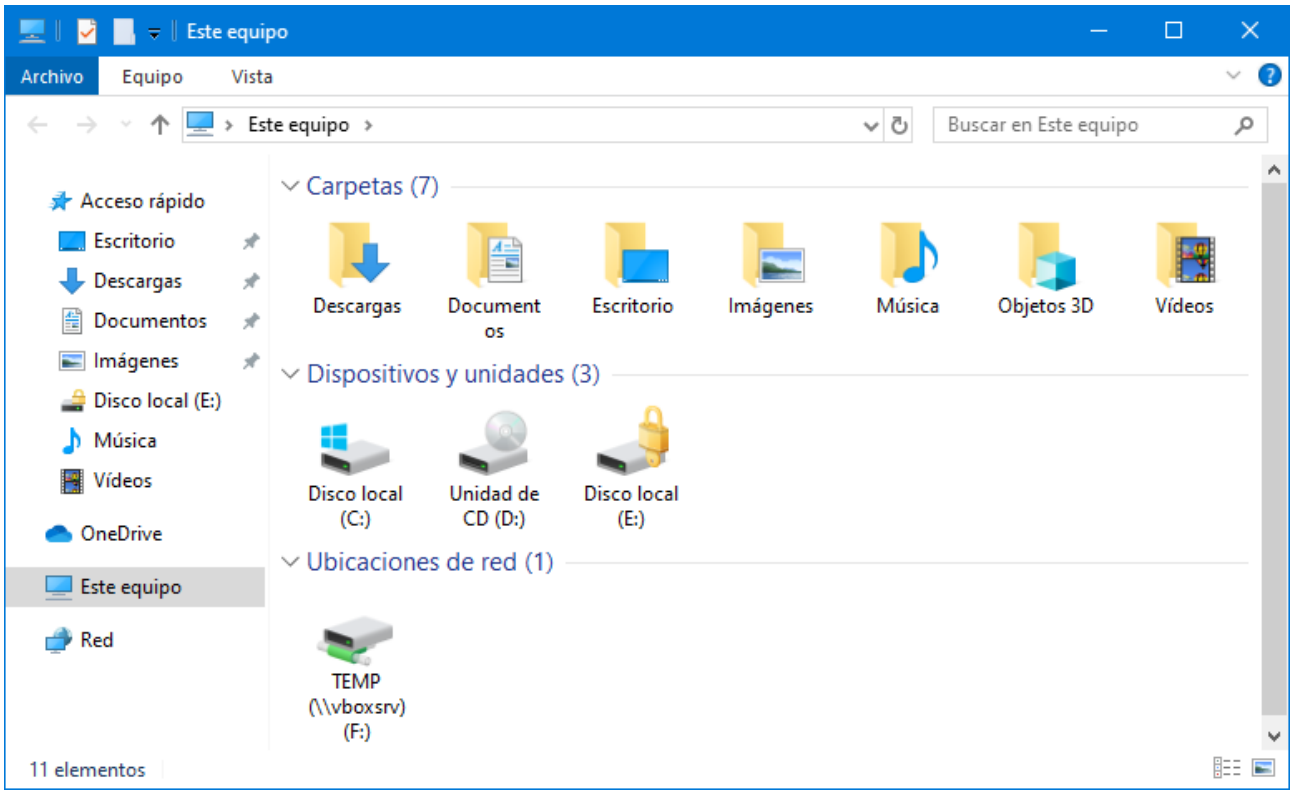
Esta operación de cifrado tardará más cuanto mayor es el disco y siempre tarda bastante si se elige cifrar la unidad entera.

Observar como aparece la unidad cifrada con BitLocker en el explorador de archivos: con un candado, abierto en este momento porque la unidad esta accesible.



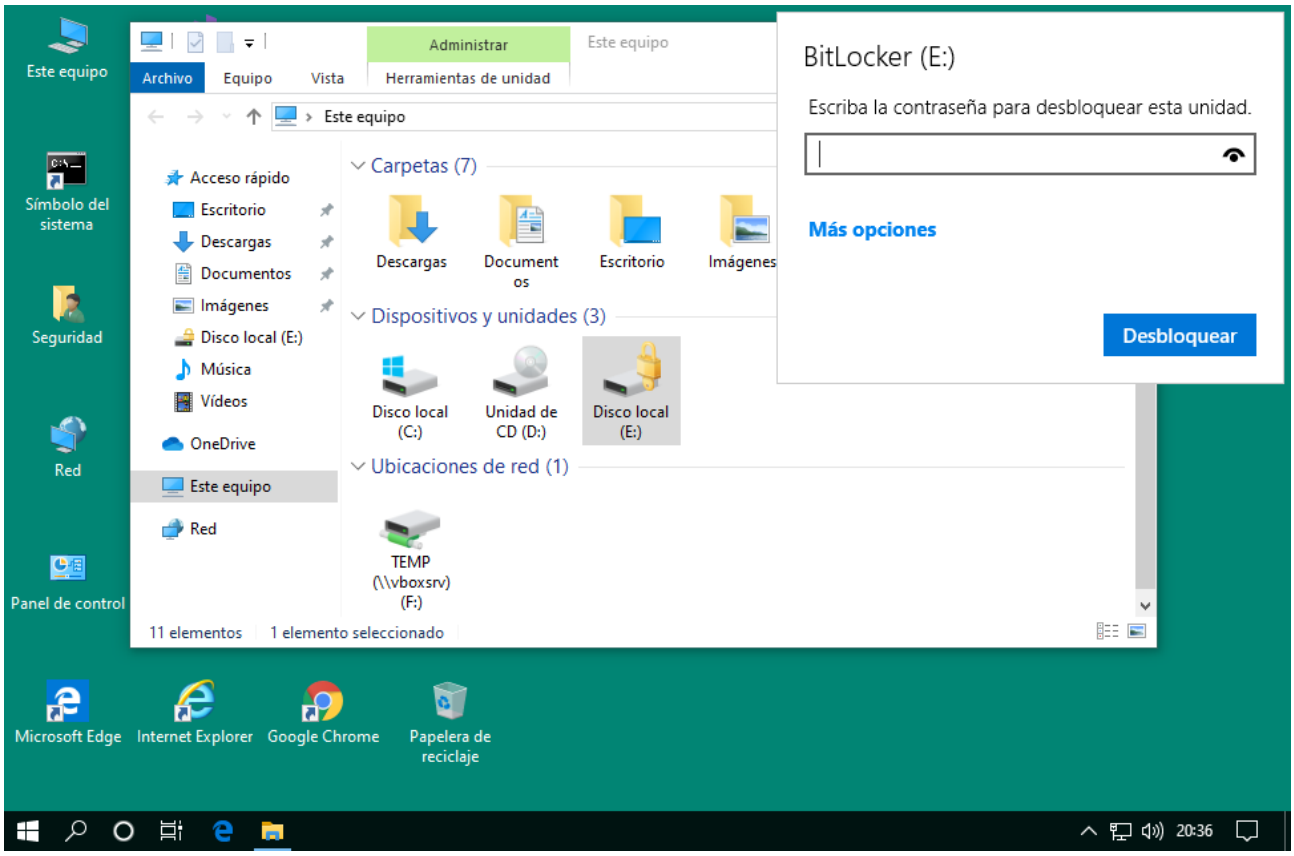
Escribe un pequeño fichero de texto para comprobar que la unidad está operativa.

Reinicia la MV. Comprueba que tras el reinicio la unidad cifrada no está accesible:



Al intentar acceder a la unidad cifrada aparece una ventana en la que se solicita la contraseña de acceso. También puedes hacer clic-derecho sobre la unidad cifrada y en el menú emergente selecciona la opción “Desbloquear unidad”.

Aparece una ventana emergente para introducir la contraseña a partir de la cual se deriva la clave utilizada para cifrar y descifrar:



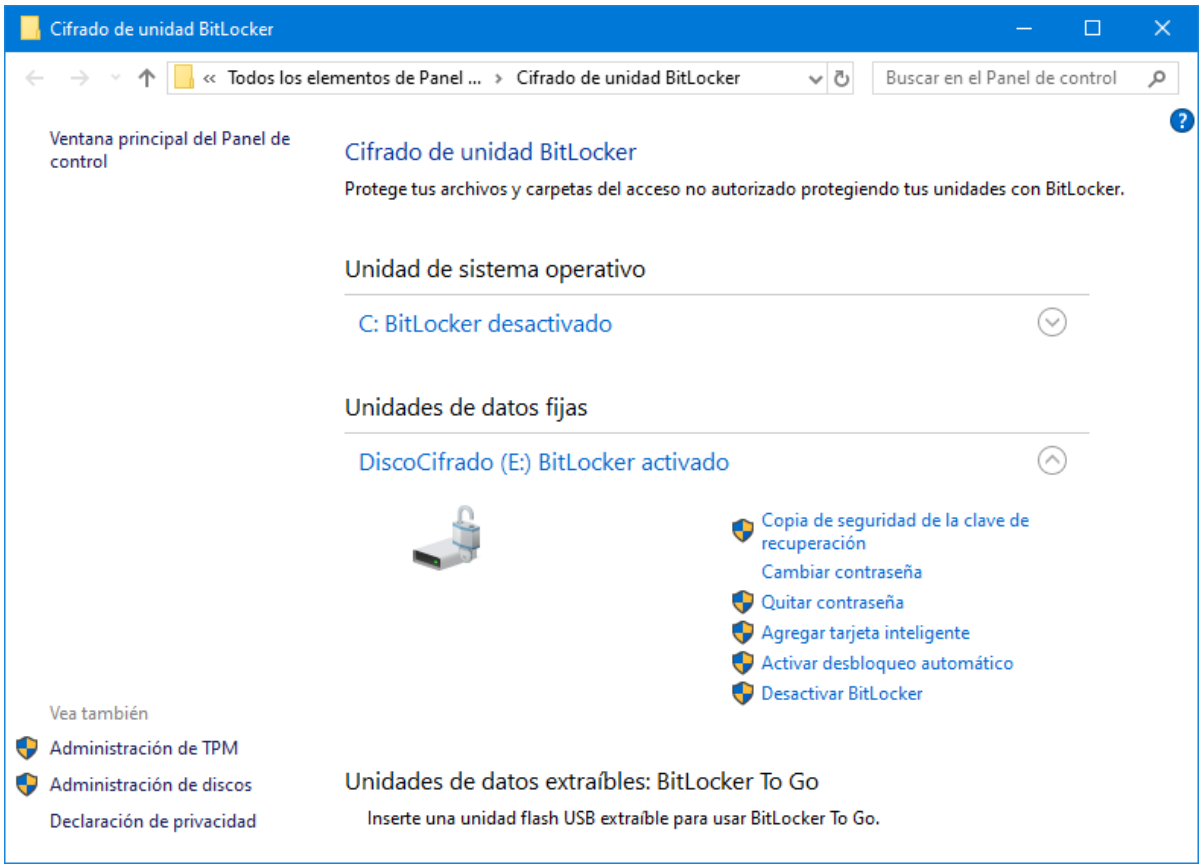
Tras introducir la contraseña el candado aparece abierto y la unidad cifrada es accesible.

Comprueba que si se cierra una sesión de usuario, al volver a abrirla, la unidad cifrada con BitLocker permanece accesible.

¿Qué ocurre si se nos olvida la contraseña? En este caso selecciona “Más opciones” y entonces aparece la opción “Escribir clave de recuperación”. Al seleccionar esta opción aparece una ventana en la que se muestra el inicio del identificador de la clave de recuperación de 48 dígitos. El identificador de la clave de recuperación forma parte del nombre del fichero en el que almacenó la clave.

Copia la clave del fichero y pégala en la ventana en la que se solicita. Podrás comprobar que la unidad cifrada se desbloquea.

Haciendo clic-derecho sobre la unidad de disco cifrada aparece un menú en el que al seleccionar la opción “Administrar BitLocker” aparece:



Prueba las principales funciones administrativas, como realizar una copia de seguridad de la clave de recuperación, cambiar la contraseña, quitarla, etc.

Observa que en la esquina inferior-izquierda de la ventana previa aparece la opción “Administración de TPM”. Selecciónala y veras como aparece la consola de administración de TPM indicando que no se encuentra un módulo TPM compatible en el equipo.