Szyfrowanie danych, plików, dysków

W środowisku cybernetycznym ochrona danych jest ważna.

Jednym z wielu zadań administratora systemu jest ustawianie kontroli bezpieczeństwa i upewnianie się, że sieć nie jest łatwym celem.

Jednym ze sposobów, aby to zrobić, jest zabezpieczenie granic sieci, aby zapobiec nieautoryzowanemu dostępowi. Co jednak, jeśli Twoja sieć i tak zostanie naruszona?

Być może ktoś fizycznie wchodzi do twojego centrum danych i kradnie serwer, aby zebrać cenne dane, które mogłeś na nim przechowywać. Jeśli Twoje dane nie są zaszyfrowane, pożegnaj się z nimi. Ale jeśli wcześniej udało Ci się zaszyfrować dane na tym serwerze, podczas gdy Twoje dane mogą nadal zniknąć, przynajmniej będziesz wiedział, że nie zostaną odczytane.

Szyfrowanie danych jest zawsze dobrym pomysłem, ale może być trudne w zarządzaniu, zwłaszcza na różnych serwerach i lokalizacjach pamięci masowej.

1. Szyfrowanie:

Otwórz edytor PowerShell ISE jako administrator i sprawdź wersję PS. Zagadnienia testowane są na pokazanej poniżej wersji 5.X PS.

```
PS C:\Windows\system32> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.22621.3810
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.22621.3810
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

Najpierw zrozummy, jak działa szyfrowanie w PowerShell.

Tutaj wzięliśmy ciąg „Witam@123” w zmiennej pwd i po prostu powtórzyliśmy go.

```
PS C:\Windows\system32> $pwd = "Witam@123"

PS C:\Windows\system32> Write-Host $pwd
Witam@123
```

Teraz spróbujmy go zaszyfrować.

Pierwszym krokiem byłoby przekonwertowanie normalnego ciągu "Witam@123"na bezpieczny ciąg za pomocą polecenia cmdlet ConvertTo-SecureString.

```
$pwd | ConvertTo-SecureString -AsPlainText -Force
```

```
PS C:\Windows\system32> $pwd | ConvertTo-SecureString -AsPlainText -Force
System.Security.SecureString
```

Ten bezpieczny ciąg musi być dalej szyfrowany za pomocą polecenia cmdlet ConvertFrom-SecureString .

Utworzenie zwykłego tekstu hasła:

```
$pwd = "Witam@123"
```

W tej linii kodu tworzony jest zwykły tekst hasła i przypisywany do zmiennej $pwd.

Konwersja zwykłego tekstu na obiekt SecureString:

```
$securepwd = $pwd | ConvertTo-SecureString -AsPlainText -Force
```

Używając cmdletu ConvertTo-SecureString, zwykły tekst hasła jest konwertowany na obiekt SecureString. Parametry -AsPlainText i -Force są używane do określenia, że tekst jest w formie zwykłego tekstu, który ma zostać przekształcony na bezpieczny obiekt SecureString. Parametr -Force jest wymagany, aby potwierdzić operację konwersji.

Konwersja obiektu SecureString z powrotem na tekst:

```
$securepwd | ConvertFrom-SecureString
```

```
PS C:\Windows\system32> $securepwd | ConvertFrom-SecureString
01000000d08c9ddf0115d1118c7a00c04fc297eb0100000034b79755021a6f4c
1088892f9a0d948e3ac041788863547e721c2ba67e91be15e0fbed000000000e
b155f8386ab9ff632000000070a99c2c23fc38ad41b6ccb2accb73a7d544f9d7
98ae477aec8e5799156a596d4b161768fae98f94b1ae675bf114fde4f7bed7bd
```

W tej linii kodu obiekt SecureString (przechowywany w zmiennej $securepwd) jest przesyłany do cmdletu ConvertFrom-SecureString, który przekształca go z powrotem na zwykły tekst. Ostatecznie rezultat tego procesu (czyli oryginalny tekst hasła) jest wyświetlany na ekranie.

Warto zauważyć, że przechowywanie haseł w formie SecureString zamiast w formie zwykłego tekstu zwiększa poziom bezpieczeństwa, ponieważ obiekty SecureString są trudniejsze do przechwycenia i odczytania niż zwykłe ciągi znaków. Jednakże, w rzeczywistości, pełne zabezpieczenie hasła zależy również od innych czynników, takich jak praktyki zarządzania hasełami i środowisko, w którym są używane.

Zaszyfrowany ciąg może być przechowywany w pliku do dalszego wykorzystania lub może być przechowywany w zmiennej zgodnie z wymaganiami.

Kod szyfrujący:

```
$pwd = “Witam@123”

$securepwd = $pwd | ConvertTo-SecureString -AsPlainText -Force
```

Zaszyfrowanie obiektu SecureString i wyświetlenie zaszyfrowanego tekstu:

```
$encryptedpwd = $securepwd | ConvertFrom-SecureString

write-host $encryptedpwd
```

Zaszyfrowane hasło jest uzyskiwane z obiektu SecureString przy użyciu cmdletu ConvertFrom-SecureString. Następnie zaszyfrowane hasło jest wyświetlane na ekranie za pomocą Write-Host.

Warto zauważyć, że zaszyfrowane hasło nie jest bezpieczne do przechowywania w celu późniejszego odzyskiwania oryginalnego hasła. Cmdlet ConvertFrom-SecureString używa klucza maszyny lokalnej, co oznacza, że zaszyfrowane hasło jest związane z konkretnym komputerem. Dlatego jest to bardziej przydatne w kontekście lokalnej sesji użytkownika na tym samym komputerze, a nie do przenoszenia hasła między różnymi maszynami. W praktyce zaleca się unikanie przechowywania haseł w formie zaszyfrowanej, a zamiast tego korzystanie z innych mechanizmów, takich jak hashowanie.

```
PS C:\Windows\system32> $pwd = "Witam@123"

PS C:\Windows\system32> $securepwd = $pwd | ConvertTo-SecureString -AsPlainText -Force

PS C:\Windows\system32> $encryptedpwd = $securepwd | ConvertFrom-SecureString

PS C:\Windows\system32> write-host $encryptedpwd
01000000d08c9ddf0115d1118c7a00c04fc297eb0100000034b79755021a6f4c9925a0b2c3c6ebc40000000
ac90ba4cc7d68e1e5009113145d6cdefe1fa4db629a1871804cf13000000000e800000000200002000000009
8f0b6037d970881d20000000c7b3b7aec8b07d526803c066fabcab2fb327fd827197311631b7884e658a6cf
d2b6230abe2b56606ae12f1492402f0fee8a88b27b03a20345aa641c733cc18ddfdfb05cdedce0f4bcfbe68
```

2. Deszyfrowanie:

Zobaczmy teraz, jak działa odszyfrowywanie w PowerShell.

Mamy zaszyfrowane hasło przechowywane w zmiennej, jak pokazano powyżej.

Kod PowerShell wykorzystuje Write-Host do wyświetlenia zaszyfrowanego hasła na ekranie.

write-host $encryptedpwd

```
PS C:\Windows\system32> write-host $encryptedpwd
01000000d08c9ddf0115d1118c7a00c04fc297eb0100000034b79755021a
ac90ba4cc7d68e1e5009113145d6cdefe1fa4db629a1871804cf1300000
8f0b6037d970881d20000000c7b3b7aec8b07d526803c066fabcab2fb327
d2b6230abe2b56606ae12f1492402f0fee8a88b27b03a20345aa641c733c
```

Teraz, aby odszyfrować zaszyfrowany ciąg, pójdziemy w odwrotną stronę. Najpierw skonwertujemy zaszyfrowany ciąg z powrotem na bezpieczny ciąg za pomocą polecenia cmdlet ConvertTo-SecureString .

$securepwd = $encryptedpwd | ConvertTo-SecureString

write-host $securepwd

```
PS C:\Windows\system32> $securepwd = $encryptedpwd | ConvertTo-SecureString

PS C:\Windows\system32> write-host $securepwd
System.Security.SecureString
```

PowerShell używa bibliotek .Net Framework, tutaj klasy Marshal z przestrzeni nazw System.Runtime.InteropServices z metodami PtrToStringAuto i SecureStringToBSTR, aby przekonwertować zaszyfrowany ciąg na zwykły tekst.

```
$Marshal = [System.Runtime.InteropServices.Marshal]
```

```
write-output $Marshal
```

```
PS C:\Windows\system32> $Marshal = [System.Runtime.InteropServices.Marshal]

PS C:\Windows\system32> write-output $Marshal

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    Marshal                                  System.Object
```

SecureStringToBSTR - przydziela niezarządzany ciąg binarny (BSTR) i kopiuje do niego zawartość zarządzanego obiektu SecureString.

```
$Bstr = $Marshal::SecureStringToBSTR($securepwd)
```

```
write-output $Bstr
```

```
PS C:\Windows\system32> $Bstr = $Marshal::SecureStringToBSTR($securepwd)

PS C:\Windows\system32> write-output $Bstr
2712804449320
```

PtrToStringAuto - przydziela zarządzany ciąg i kopiuje do niego całość lub część niezarządzanego ciągu. Zmienna pwd będzie miała zachowany ciąg tekstowy lub hasło.

```
$pwd = $Marshal::PtrToStringAuto($Bstr)
```

```
write-host $pwd
```

```
PS C:\Windows\system32> $pwd = $Marshal::PtrToStringAuto($Bstr)

PS C:\Windows\system32> write-host $pwd
Witam@123
```

ZeroFreeBSTR ma na celu wyczyszczenie niezarządzanej pamięci

```
$Marshal::ZeroFreeBSTR($Bstr)
```

Poniżej znajduje się kod deszyfrujący do odszyfrowania zaszyfrowanego ciągu lub hasła.

Kod deszyfrujący:

```
$securepwd = $encryptedpwd | ConvertTo-SecureString

$Marshal = [System.Runtime.InteropServices.Marshal]

$Bstr = $Marshal::SecureStringToBSTR($securepwd)

$pwd = $Marshal::PtrToStringAuto($Bstr)

Write-host $pwd

$Marshal::ZeroFreeBSTR($Bstr)
```

```
PS C:\Windows\system32> $securepwd = $encryptedpwd | ConvertTo-SecureString

PS C:\Windows\system32> $Marshal = [System.Runtime.InteropServices.Marshal]

PS C:\Windows\system32> $Bstr = $Marshal::SecureStringToBSTR($securepwd)

PS C:\Windows\system32> $pwd = $Marshal::PtrToStringAuto($Bstr)

PS C:\Windows\system32> Write-host $pwd
Witam@123

PS C:\Windows\system32> $Marshal::ZeroFreeBSTR($Bstr)

PS C:\Windows\system32>
```

Szyfruj pliki

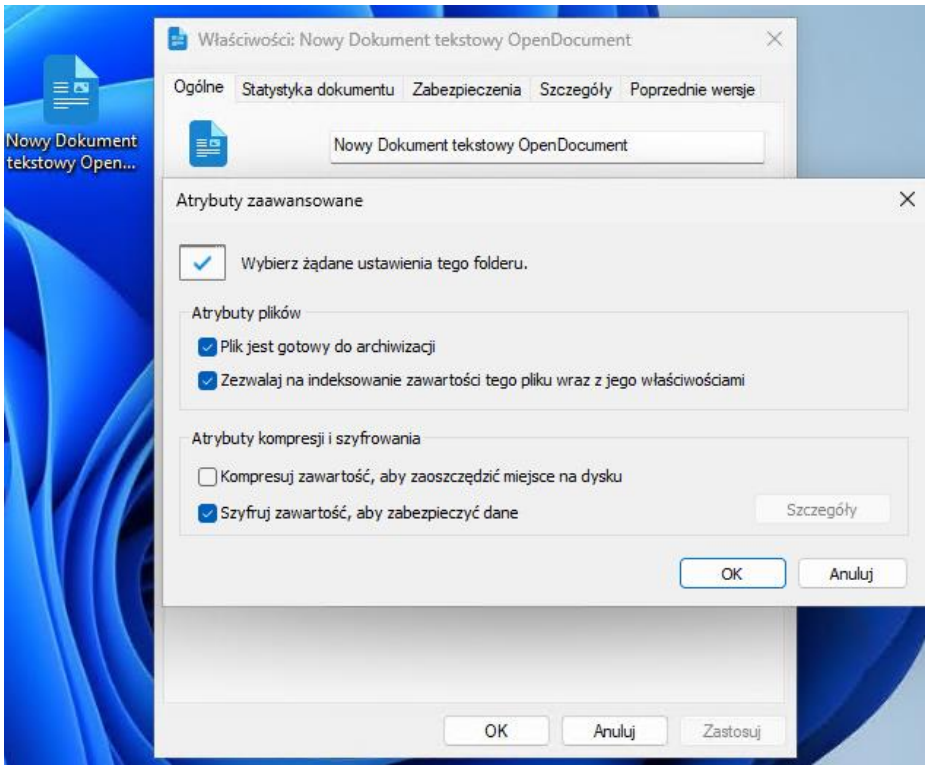
Korzystając z wbudowanej technologii szyfrowania plików (EFS) firmy Microsoft i programu PowerShell, zadanie szyfrowania i odszyfrowywania jednego, dwóch lub milionów plików i folderów w centrum danych może być znacznie łatwiejsze.

Pokażę, jak ręcznie zaszyfrować i odszyfrować pliki za pomocą EFS za pomocą GUI.

Omówię trochę kodu PowerShell, który pozwoli wykonać zadanie w wielu różnych lokalizacjach jednocześnie.

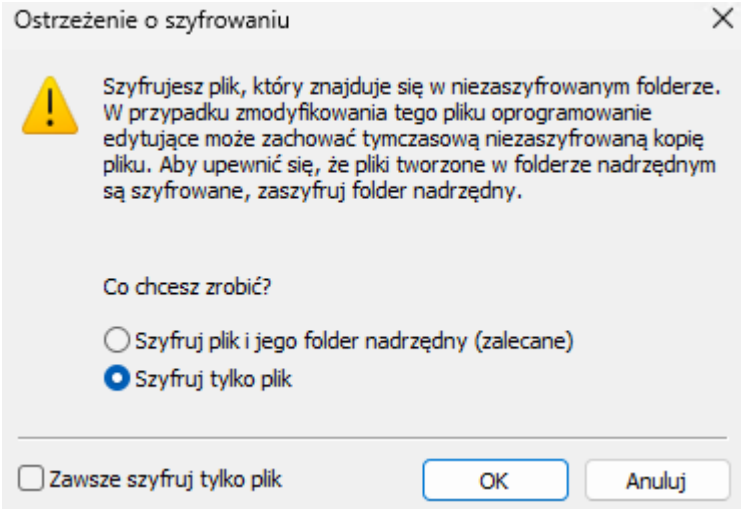
3. Szyfruj pliki przez GUI

Najpierw musisz znaleźć plik, który chcesz zaszyfrować w Eksploratorze Windows. Kliknij plik prawym przyciskiem myszy i wybierz Właściwości. Następnie w panelu Właściwości zobaczysz przycisk Zaawansowane. Kliknij to, a zobaczysz opcję zaszyfrowania pliku.



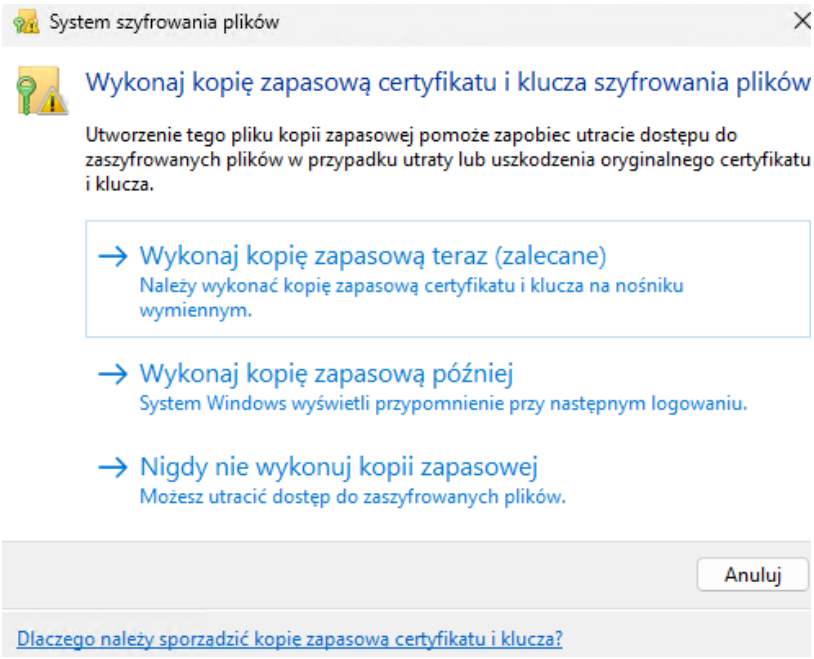
Pokazuje, jak zaszyfrować plik za pomocą GUI.

Zaznacz pole wyboru „Szyfruj zawartość, aby zabezpieczyć dane”> OK i zastosuj zmianę, aby natychmiast zaszyfrować plik.



Zauważysz, że ikona pliku się zmieni.

Pojawi się komunikat aby wykonać kopie zapasową certyfikatu i klucza szyfrowania plików



Wybierz format, którego chcesz użyć:

- ○ Certyfikat X.509 szyfrowany binarnie algorytmem DER (CER)
- ○ Certyfikat X.509 szyfrowany algorytmem Base-64 (CER)
- ○ Standard składni wiadomości kryptograficznych — certyfikaty PKCS #7 (P7B)
  - ☐ Jeśli jest to możliwe, dołącz wszystkie certyfikaty ze ścieżki certyfikacji
- ◉ Wymiana informacji osobistych — PKCS #12 (PFX)
  - ☑ Jeśli jest to możliwe, dołącz wszystkie certyfikaty do ścieżki certyfikacji
  - ☐ Usuń klucz prywatny, jeśli eksport został zakończony pomyślnie
  - ☐ Eksportuj wszystkie właściwości rozszerzone
  - ☑ Włącz funkcję prywatności certyfikatu
- ○ Magazyn certyfikatów seryjnych firmy Microsoft (SST)

☑ Hasło:

••••

Potwierdź hasło:

••••

Szyfrowanie: TripleDES-SHA1

Nazwa pliku:

C:\Users\admin\Desktop\cert_szyfr.pfx

Kreator eksportu certyfikatów — Zapraszamy!

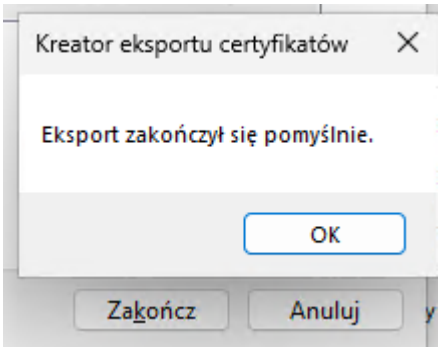
Ten kreator pozwala kopiować certyfikaty, listy zaufania certyfikatów oraz listy odwołania certyfikatów z magazynu certyfikatów na dysk.

Certyfikat, wystawiany przez urząd certyfikacji, stanowi potwierdzenie tożsamości użytkownika i zawiera informacje używane do ochrony danych lub do ustanawiania bezpiecznych połączeń sieciowych. Magazyn certyfikatów jest obszarem systemowym, w którym przechowywane są certyfikaty.

Aby kontynuować, kliknij przycisk Dalej.

Wybrane zostały następujące ustawienia:

Nazwa pliku	C:\Users\admin\Desktop\cert_szyf
Eksportuj klucze	Tak
Dołącz wszystkie certyfikaty ze ścieżki certyfikacji	Tak
Format pliku	Wymiana informacji osobistych (*.



4. Automatyzacja szyfrowania plików

Przygotuj przykładowy plik:

```
New-Item -Path "C:\Groups.csv" -ItemType "File"

# Pobierz listę wszystkich grup w systemie

$groups = Get-LocalGroup

# Eksportuj listę grup do pliku CSV

$groups | Export-Csv -Path "C:\Groups.csv" -NoTypeInformation
```

W środowisku biznesowym prawdopodobnie będziesz musiał zaszyfrować cały folder lub wiele różnych folderów w różnych lokalizacjach. Jeśli wolisz nie spędzać czasu na ich ręcznym szyfrowaniu, jest lepszy sposób: użyj PowerShell.

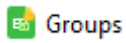
Korzystając ze skryptu PowerShell, możesz zbudować kod, który pozwoli Ci przekazać do niego dowolną liczbę plików lub folderów, aby automatycznie je zaszyfrować, niezależnie od tego, gdzie się znajdują.

Na szczęście Microsoft był dla nas miły i nie wymaga wielu skryptów, aby tak się stało. Czynność szyfrowania i odszyfrowywania pliku jest tak prosta, jak wywołanie metod Encrypt() i Decrypt() na określonym typie obiektu, co można łatwo uzyskać za pomocą Get-Item lub, w przypadku całego folderu(ów) , z Get-ChildItem .

Zaszyfruj jak w poniższym przykładzie tylko w jednej linii kodu.

```
(Get-Item -Path C:\Groups.csv).Encrypt()
```

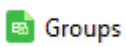
```
PS C:\Windows\system32> (Get-Item -Path C:\Groups.csv).Encrypt()
```



Aby odszyfrować:

```
(Get-Item -Path C:\Groups.csv).Decrypt()
```

```
PS C:\Windows\system32> (Get-Item -Path C:\Groups.csv).Decrypt()
```



Szyfrowanie i/lub odszyfrowywanie całego folderu jest równie łatwe. Ale zamiast używać Get-Item , musisz użyć Get-ChildItem, aby pobrać wszystkie pliki z tego folderu.

0. Przygotowanie. Wykonaj w PowerShell poniższe polecenia:

```
cd /

"Documents", "Desktop"  | %{New-Item -Name "$_" -ItemType "Directory"}

New-Item -Path 'C:\Documents\New Folder\file.txt' -ItemType File -f -v "nnn"

New-Item -Path 'C:\Documents\cars.xml' -ItemType File -f -v "xml_nnn"
```

```
New-Item -Path 'C:\Documents\New Folder\kealtcheck.html' -ItemType File -f -v "html_nnn"
New-Item -Path 'C:\Documents\servicereport.html' -ItemType File -f -v "service_html_nnn"
New-Item -Path 'C:\Desktop\style.css' -ItemType File -f -v "css_nnn"
New-Item -Path 'C:\Desktop\test.xlsx' -ItemType File -f -v "xlsx_nnn"
New-Item -Path 'C:\Desktop\testcsv.csv' -ItemType File -f -v "csv_nnn"
```

Spróbuj wykonać

```
(Get-ChildItem -Path C:\Documents).Encrypt()
```

Błąd "Method invocation failed because [System.IO.DirectoryInfo] does not contain a method named 'Encrypt'" oznacza, że metoda Encrypt nie jest dostępna dla obiektów typu System.IO.DirectoryInfo, które są zwracane przez Get-ChildItem dla katalogów.

Metoda Encrypt jest dostępna tylko dla plików (System.IO.FileInfo), a nie dla katalogów.

Aby zaszyfrować wszystkie pliki w katalogu C:\Documents, możesz użyć poniższego skryptu:

```
# Sprawdź, czy katalog C:\Documents istnieje
if (Test-Path -Path "C:\Documents") {
    # Pobierz wszystkie pliki z katalogu C:\Documents
    $files = Get-ChildItem -Path "C:\Documents" -File
    # Sprawdź, czy katalog zawiera jakieś pliki
    if ($files) {
        # Zaszyfruj każdy plik w katalogu
        foreach ($file in $files) {
            $file.Encrypt()
        }
        Write-Output "All files in C:\Documents have been encrypted."
    } else {
        Write-Output "The directory C:\Documents does not contain any files."
    }
} else {
    Write-Output "The directory C:\Documents does not exist."
}
```

Efekt:

```
All files in C:\Documents have been encrypted.
PS C:\>
```

5. Używanie funkcji PowerShell do szyfrowania plików

Używaj funkcji i poleceń cmdlet PowerShell zamiast metod .NET, takich jak Encrypt() i Decrypt() . Zbuduje funkcje, które pozwolą mi zamiast tego używać Enable- FileEncryption i Disable-FileEncryption . Aby wyjaśnić, jak to działa, spójrzmy na skrypt.

Pobierz przykładowy skrypt, aby to przetestować. Aby użyć tego skryptu, otwórz konsolę PowerShell i uruchom skrypt w bieżącej sesji.

```
Set-ExecutionPolicy Unrestricted
.\C:\EFS.ps1
```

Wprowadzi to każdą funkcję zadeklarowaną w skrypcie. Możesz teraz używać funkcji do szyfrowania i odszyfrowywania dowolnych plików.

Aby zaszyfrować plik, użyj Enable-FileEncryption.

```
Get-Item C:\Groups.csv | Enable-FileEncryption
```

Aby odszyfrować plik, zrób odwrotnie.

```
Get-Item C:\Groups.csv | Disable-FileEncryption
```

Aby sprawdzić, czy plik jest zaszyfrowany, oraz aby go zaszyfrować w PowerShell, możesz użyć właściwości Attributes obiektu pliku i metody Encrypt(). Poniżej przedstawiam szczegółowy sposób, jak to zrobić. Sprawdzanie czy plik jest zaszyfrowany

Plik jest zaszyfrowany, jeśli ma ustawiony atrybut Encrypted. Możesz to sprawdzić w ten sposób:

```
$file = Get-Item -Path "C:\Groups.csv"
if ($file.Attributes -band [System.IO.FileAttributes]::Encrypted) {
    Write-Output "The file is encrypted."
} else {
    Write-Output "The file is not encrypted."
```

Aby zaszyfrować folder, użyj Get-ChildItem do wyliczenia wszystkich plików w folderze.

```
Get-ChildItem -Path 'C:\Documents' -File -Recurse | ForEach-Object {$_ | Enable-FileEncryption}
```

Deszyfrowanie wszystkich plików w folderze: Analogicznie, aby odwrócić proces i usunąć szyfrowanie z plików, możesz użyć funkcji Disable-FileEncryption:

```
Get-ChildItem -Path 'C:\Folder' -File -Recurse | ForEach-Object {$_ | Disable-FileEncryption}
```

Szyfrować plików w wielu folderach. Zastosuj Get-ChildItem do pobrania plików z kilku różnych lokalizacji (takich jak C:\Documents i C:\Desktop) i przekazać je przez potok do funkcji Enable-FileEncryption w jednym poleceniu PowerShell:

```
Get-ChildItem -Path "C:\Documents", "C:\Desktop" -File -Recurse | Enable-FileEncryption
```

Deszyfrowanie plików w wielu folderach.

```
Get-ChildItem -Path "C:\Documents", "C:\Desktop" -File -Recurse | Disable-FileEncryption
```

Aby sprawdzić, czy pliki są zaszyfrowane w wielu folderach takich jak C:\Documents i C:\Desktop, użyj poniższego skryptu PowerShell. Skrypt ten wykorzystuje właściwość Attributes obiektu pliku, aby określić, czy plik jest zaszyfrowany przy użyciu systemu szyfrowania EFS (Encrypting File System). Uruchom

```
# Pobierz pliki z kilku lokalizacji i sprawdź, czy są zaszyfrowane

Get-ChildItem -Path "C:\Documents", "C:\Desktop" -File -Recurse | ForEach-Object {

    $file = $_

    if ($file.Attributes -band [System.IO.FileAttributes]::Encrypted) {

        Write-Output "$($file.FullName) is encrypted."

    } else {

        Write-Output "$($file.FullName) is not encrypted."

    }

}
```

Następnym razem, gdy będziesz musiał zaszyfrować jeden lub więcej plików, pamiętaj, że kontrole bezpieczeństwa można wykonać w PowerShell.