

**CYB631 – Lab 4: Confidentiality**

Vijaysingh Puwar

Seidenberg School of CSIS, Pace University

CYB 631: Automating Information Security with Python & PowerShell

Prof. Darren Hayes

September 29, 2025

## Contents

ABSTRACT .....	6
EXERCISES:[ENVIRONMENT: STARTING WITH POWERSHELL ISE].....	7
Step 1.....	7
Step 2.....	7
EXERCISE I — HASHING .....	8
Step 3.....	8
Step 4.....	8
Step 5.....	8
Step 6.....	8
Step 7.....	9
Step 8.....	9
Step 9-10 .....	10
Exercise II: Integrity Check for Downloading Files .....	11
Step 11.....	11
Step 12.....	11
Step 13.....	11
Step 14.....	11
Step 15.....	11
Step 16.....	12
Step 17- What does the result mean? Please explain it.....	12
Exercise III: SecureString and AES Encryption .....	13

Step 18.....	13
Step 19.....	13
Step 20.....	14
Step 21.....	14
Step 22.....	14
Step 23.....	15
Step 24.....	15
Step 25.....	16
Step 26.....	17
Step 26.....	18
Exercise IV: Secure Passwords.....	19
Step 28.....	19
Step 29.....	19
Step 30.....	19
Step 31: What would happen if the key file in the above step is missing? .....	20
Step 32: Is the key file encrypted at this point? What would happen if an adversary obtained the key file? .....	20
Exercise V: Self-Signed Certificate and Script Signing .....	21
Step 33.....	21
Step 34.....	21
Step 35.....	22
Step 36.....	22

Step 37.....	22
Step 38.....	23
Step 39.....	24
Step 40.....	24
Step 41.....	25
Step 42.....	25
Step 43.....	26
Step 43.....	27
Step 44.....	27
Step 44.....	28
Step 46.....	29
Step 47.....	30
Exercise VI: Encryption and Decryption using a Certificate.....	31
Step 48.....	31
Step 49.....	31
Step 50.....	31
Step 51.....	32
Step 52.....	32
Step 53.....	33
Step 54.....	33
Step 55.....	33

Exercise VII: Lab and Class Reflection .....	35
What did you like about this lab?.....	35
What were the challenges?.....	35
Suggestions to help me learn better: .....	35
VM power-down.....	36

## ABSTRACT

This lab operationalizes core confidentiality principles in Windows using PowerShell and X.509 public-key infrastructure. First, file integrity was demonstrated by generating cryptographic digests with Get-FileHash and building a two-column SHA-256 report, followed by validating a vendor download via hash comparison to confirm authenticity. Next, sensitive data handling was explored with SecureString, including converting secrets to AES-encrypted ciphertext and decrypting them using explicitly managed 16/24/32-byte keys stored in a protected key file. Secure credential storage was reinforced by exporting an encrypted password and analyzing the risks of exposed key material. The lab then implemented script trust through a self-signed **Code Signing** certificate, applying Set-AuthenticodeSignature and verifying signature status to satisfy execution policy controls. Finally, confidentiality at rest and in transit was exercised through CMS encryption: generating a Document Encryption certificate, protecting plaintext with Protect-CmsMessage, and recovering it with Unprotect-CmsMessage under the original user profile. Evidence was captured via reproducible commands, aligned screenshots, and brief interpretations. Collectively, these tasks connect theory to practice, illustrating how hashing, symmetric and asymmetric cryptography, and code signing mitigate integrity and confidentiality risks in day-to-day security operations.

## EXERCISES:[ENVIRONMENT: STARTING WITH POWERSHELL ISE]

### Step 1

Here I install Windows 2022 Server in Virtual box and using the CLI command to navigate the powershell

```
Powershell
```

This command help us to install the powershell. And in order to launch the powershell use this command

```
Powershell -Command "Start-Process powershell -Verb runAs"
```

### Step 2

Now, open a new PowerShell ISE and run as an administrator. Make sure that you navigate to the directory where your script is. Change the executive policy to RemoteSigned.

```
cd C:\Users\cybusr\cyb631
Set-ExecutionPolicy remotesigned
```

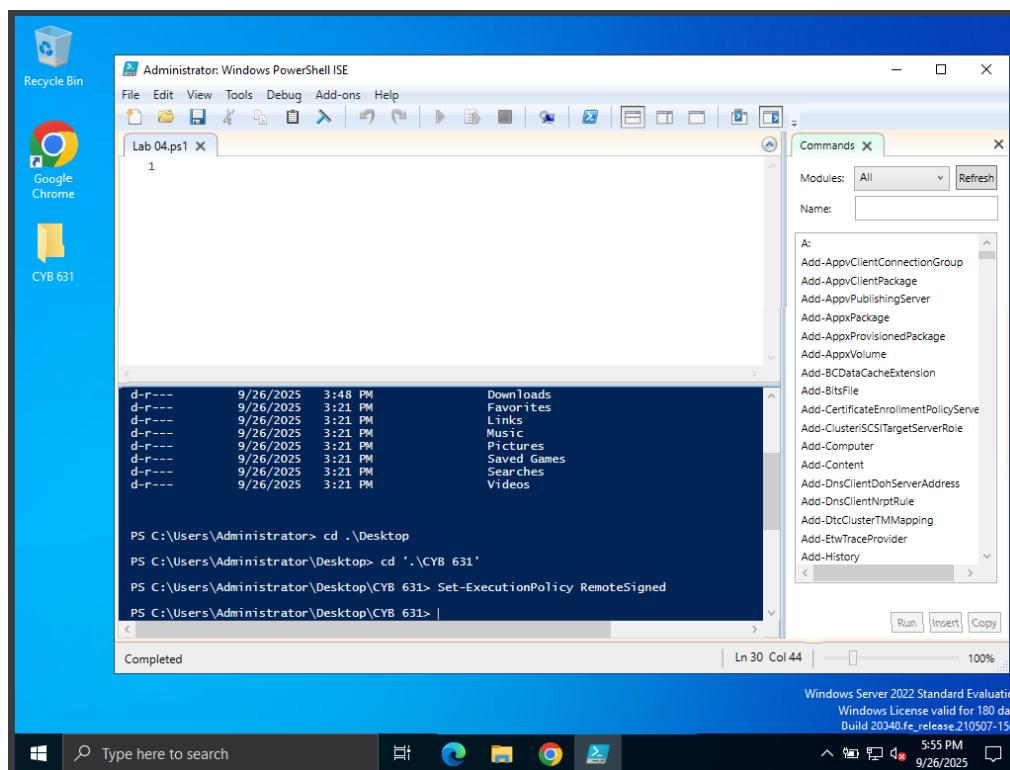


Figure 1

## EXERCISE I — HASHING

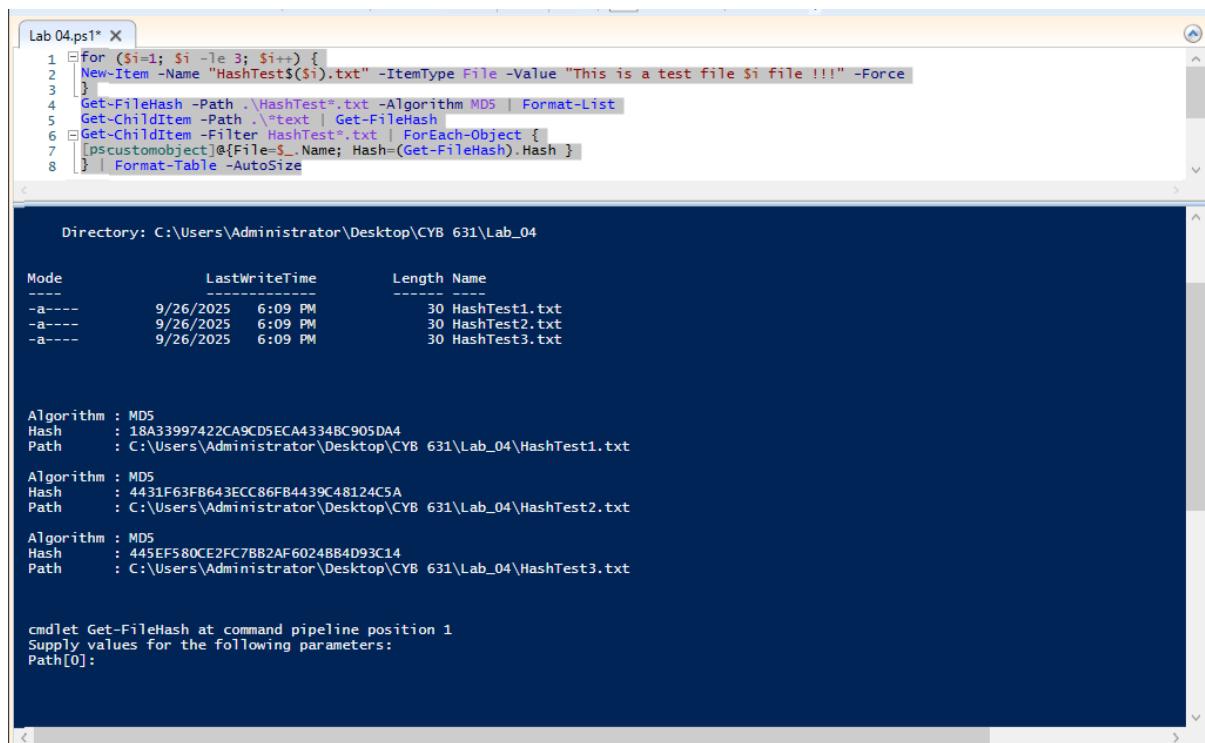
### Step 3

PowerShell has a Get-FileHash cmdlet for hashing files. The default hash function is **SHA256** if it is not specified.

### Step 4

Create some files for testing file hash. Let us learn about how to do this with a loop.

Can see in below screenshot.



```

Lab 04.ps1* X
1 for ($i=1; $i -le 3; $i++) {
2   New-Item -Name "HashTest$($i).txt" -ItemType File -Value "This is a test file $i file !!!" -Force
3 }
4 Get-FileHash -Path .\HashTest*.txt -Algorithm MD5 | Format-List
5 Get-ChildItem -Path .\text | Get-FileHash
6 Get-ChildItem -Filter HashTest*.txt | ForEach-Object {
7   [pscustomobject]@{File=$_.Name; Hash=(Get-FileHash).Hash }
8 } | Format-Table -AutoSize

Directory: C:\Users\Administrator\Desktop\CYB 631\Lab_04

Mode                LastWriteTime         Length Name
----                -----          ---- -
-a----       9/26/2025  6:09 PM           30 HashTest1.txt
-a----       9/26/2025  6:09 PM           30 HashTest2.txt
-a----       9/26/2025  6:09 PM           30 HashTest3.txt

Algorithm : MD5
Hash      : 18A33997422CA9CD5ECA4334BC905DA4
Path      : C:\Users\Administrator\Desktop\CYB 631\Lab_04\HashTest1.txt

Algorithm : MD5
Hash      : 4431F63FB643ECC86FB4439C48124C5A
Path      : C:\Users\Administrator\Desktop\CYB 631\Lab_04\HashTest2.txt

Algorithm : MD5
Hash      : 445EF580CE2FC7BB2AF6024BB4D93C14
Path      : C:\Users\Administrator\Desktop\CYB 631\Lab_04\HashTest3.txt

cmdlet Get-FileHash at command pipeline position 1
Supply values for the following parameters:
Path[0]:

```

*Figure 2*

### Step 5

We can then generate hash for all the files that you have just created. Here, we specify **MD5** as the hashing algorithm.

```
Get-FileHash -Path .\HashTest*.txt -Algorithm MD5 | Format-List
```

### Step 6

Here is the resule of the above command

```

Lab 04.ps1* X
1 For ($i=1; $i -le 3; $i++) {
2 New-Item -Name "HashTest$i.txt" -ItemType File -Value "This is a test file $i file !!!" -Force
3 }
4 Get-FileHash -Path .\HashTest*.txt -Algorithm MD5 | Format-List
5 Get-ChildItem -Path .\*text | Get-FileHash
6 Get-ChildItem -Filter HashTest*.txt | ForEach-Object {
7 [pscustomobject]@{File=$_.Name; Hash=(Get-FileHash).Hash }
8 } | Format-Table -AutoSize

```

Directory: C:\Users\Administrator\Desktop\CYB 631\Lab\_04

Mode	LastWriteTime	Length	Name
-a---	9/26/2025 6:09 PM	30	HashTest1.txt
-a---	9/26/2025 6:09 PM	30	HashTest2.txt
-a---	9/26/2025 6:09 PM	30	HashTest3.txt

Algorithm : MD5  
Hash : 18A33997422CA9CD5ECA43348C905DA4  
Path : C:\Users\Administrator\Desktop\CYB 631\Lab\_04\HashTest1.txt

Algorithm : MD5  
Hash : 4431F63FB643ECC86FB4439C48124C5A  
Path : C:\Users\Administrator\Desktop\CYB 631\Lab\_04\HashTest2.txt

Algorithm : MD5  
Hash : 445EF580CE2FC7BB2AF6024BB4D93C14  
Path : C:\Users\Administrator\Desktop\CYB 631\Lab\_04\HashTest3.txt

cmdlet Get-FileHash at command pipeline position 1  
Supply values for the following parameters:  
Path[0]:

Figure 3

## Step 7

Alternatively, we can search for all the files under a directory and pipe them to the `Get-FileHash` function.

```
Get-ChildItem -Path .\*.txt | Get-FileHash
```

```

14 Get-ChildItem -Path .\*.txt | Get-FileHash
PS C:\Users\Administrator\Desktop\CYB 631\Lab_04> Get-ChildItem -Path .\*.txt | Get-FileHash
Algorithm      Hash                                         Path
----          ---                                         ---
SHA256         6D6F668FD427AEF3E99C7DDE927434E85E16851AC43A4A825734ED931542F2C5  C:\Users\Administrator\Desktop\CYB 631\Lab_...
SHA256         510C800DC59C6AD323BDFBA36E0B22FA2178FDE23AA4D387397188690B7012E  C:\Users\Administrator\Desktop\CYB 631\Lab_...
SHA256         32583D9524AE1907EC3638CF38444BD97EDB02960D485C7888A1EA5D2BFBC00  C:\Users\Administrator\Desktop\CYB 631\Lab_...

```

Figure 4

## Step 8

Develop a script to print a concise two-column report (**File | Hash**) for all HashTest\*.txt files. The first column lists the file name and the second column lists the corresponding **SHA-256** hash value of the file.

```
Get-ChildItem -Filter 'HashTest*.txt' | ForEach-Object {  
$h = Get-FileHash -Algorithm SHA256 -Path $_.FullName  
'{0,-20} {1}' -f $_.Name, $h.Hash  
}
```

## Step 9-10

As you can see in the figure the hash of the file. In SHA256.

The screenshot shows a PowerShell window with the following content:

```
15  
16 Get-ChildItem -Filter 'HashTest*.txt' | ForEach-Object {  
17     $h = Get-FileHash -Algorithm SHA256 -Path $_.FullName  
18     '{0,-20} {1}' -f $_.Name, $h.Hash  
19 }  
20  
<  
PS C:\Users\Administrator\Desktop\CYB 631\Lab_04> Get-ChildItem -Filter 'HashTest*.txt' | ForEach-Object {  
$h = Get-FileHash -Algorithm SHA256 -Path $_.FullName  
'{0,-20} {1}' -f $_.Name, $h.Hash  
}  
6D6F668FD427AEF3E99C7DDE927434E85E16851AC43A4A825734ED931542F2C5  
510C800DC59C6AD323BDFBAA36E0B22FA2178FDE23AA4D387397188690B7012E  
32583D9524AE1907EC3638CF384448D97EDBD02960D4B5C7888A1EA5D2BF8C00  
PS C:\Users\Administrator\Desktop\CYB 631\Lab_04> |
```

Figure 5

## Exercise II: Integrity Check for Downloading Files

### Step 11

Further, we can use Get-FileHash to perform an integrity check for downloading a file (software) from the Internet.

### Step 12

Create a Web Client.

```

20
21 $wc = [System.Net.WebClient]::new()
22
23 $pkgurl = 'https://github.com/PowerShell/PowerShell/releases/download/v6.2.4/powershell_6.2.4-1.debian.9_amd64.deb'
24 $publishHash = '8E28E54D601F0751922DE24632C1E716B4684876255CF82304A9B19E89A9CCAC'

PS C:\Users\Administrator\Desktop\CYB 631\Lab_04> $wc = [System.Net.WebClient]::new()
PS C:\Users\Administrator\Desktop\CYB 631\Lab_04>

```

Figure 6

### Step 13

We will use a download link from Microsoft and verify it with the pre-calculated file hash of the software.

```

22
23 $pkgurl = 'https://github.com/PowerShell/PowerShell/releases/download/v6.2.4/powershell_6.2.4-1.debian.9_amd64.deb'
24 $publishHash = '8E28E54D601F0751922DE24632C1E716B4684876255CF82304A9B19E89A9CCAC'

PS C:\Users\Administrator\Desktop\CYB 631\Lab_04>
$pkgurl = 'https://github.com/PowerShell/PowerShell/releases/download/v6.2.4/powershell_6.2.4-1.debian.9_amd64.deb'
$publishHash = '8E28E54D601F0751922DE24632C1E716B4684876255CF82304A9B19E89A9CCAC'
PS C:\Users\Administrator\Desktop\CYB 631\Lab_04> |

```

### Step 14

Get the file hash from the file being downloaded via the web client link. Paste screenshot.

```
$FileHash = Get-FileHash -InputStream ($wc.OpenRead($pkgurl))
```

### Step 15

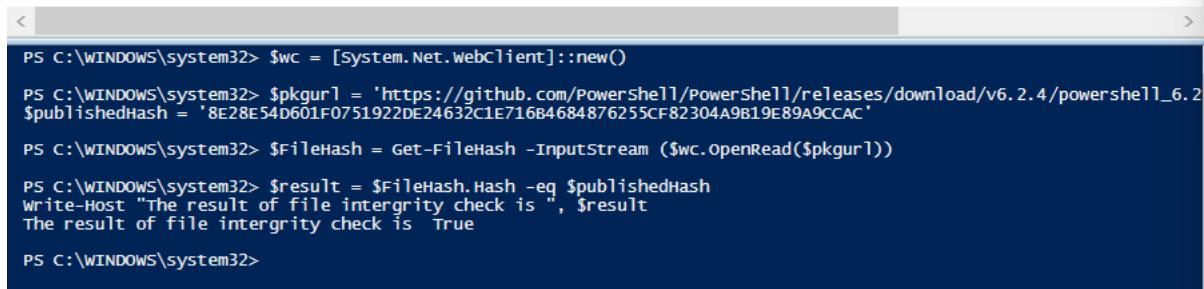
Compare the Hash

```
$result = $FileHash.Hash -eq $publishedHash
Write-Host "The result of file integrity check is ", $result
```

## Step 16

Here is the screenshot

```
1 $wc = [System.Net.WebClient]::new()
2 $pkgurl = 'https://github.com/Powershell/Powershell/releases/download/v6.2.4/powershell_6.2.4-1.debian.9_amd64.deb'
3 $publishedHash = '8E28E54D601F0751922DE24632C1E716B4684876255CF82304A9B19E89A9CCAC'
4
5 $FileHash = Get-FileHash -InputStream ($wc.OpenRead($pkgurl))
6
7
8 $result = $FileHash.Hash -eq $publishedHash
9 Write-Host "The result of file intergrity check is ", $result
```



```
PS C:\WINDOWS\system32> $wc = [System.Net.WebClient]::new()
PS C:\WINDOWS\system32> $pkgurl = 'https://github.com/Powershell/Powershell/releases/download/v6.2.4/powershell_6.2.4-1.debian.9_amd64.deb'
PS C:\WINDOWS\system32> $publishedHash = '8E28E54D601F0751922DE24632C1E716B4684876255CF82304A9B19E89A9CCAC'
PS C:\WINDOWS\system32> $FileHash = Get-FileHash -InputStream ($wc.OpenRead($pkgurl))
PS C:\WINDOWS\system32> $result = $FileHash.Hash -eq $publishedHash
Write-Host "The result of file intergrity check is ", $result
The result of file intergrity check is True
PS C:\WINDOWS\system32>
```

Figure 7

## Step 17- What does the result mean? Please explain it.

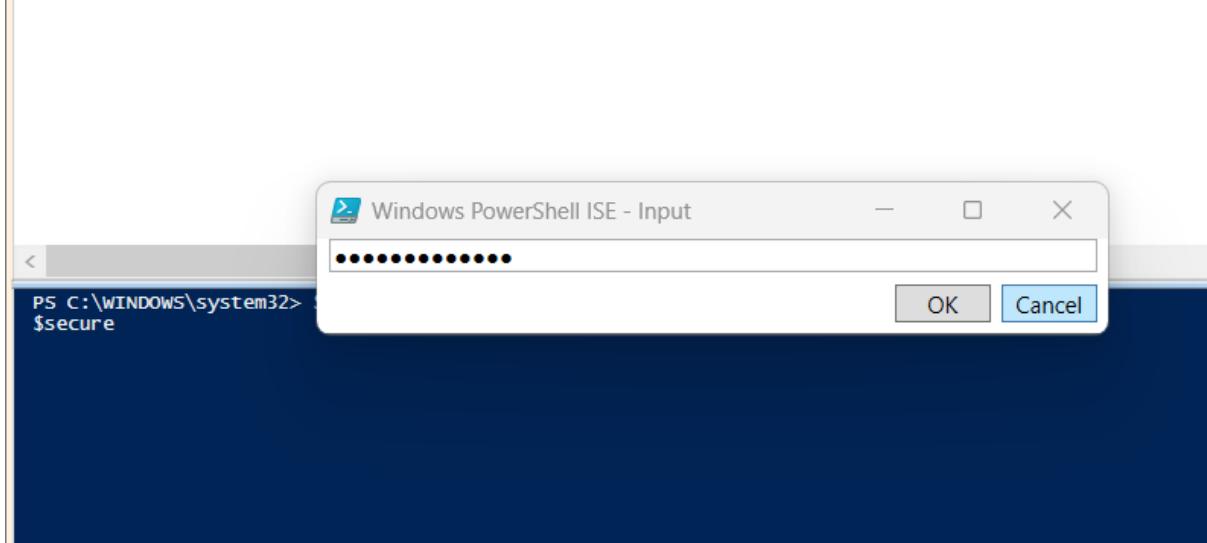
The result True indicates that the hash value of the downloaded file exactly matches the published hash provided by the software vendor. This confirms that the file has not been altered, corrupted, or tampered with during transmission and is therefore authentic. If the result had been False, it would suggest that the file might be damaged, incomplete, or potentially compromised, and it should not be trusted for installation.

## Exercise III: SecureString and AES Encryption

## Step 18

PowerShell has a data type called `SecureString`. It stores data in the memory that cannot be viewed by users but the content is not encrypted in the memory. Read a string from command line and store it as a `SecureString`.

```
11 $secure = Read-Host -AsSecureString  
12 $secure  
13  
14
```



*Figure 8*

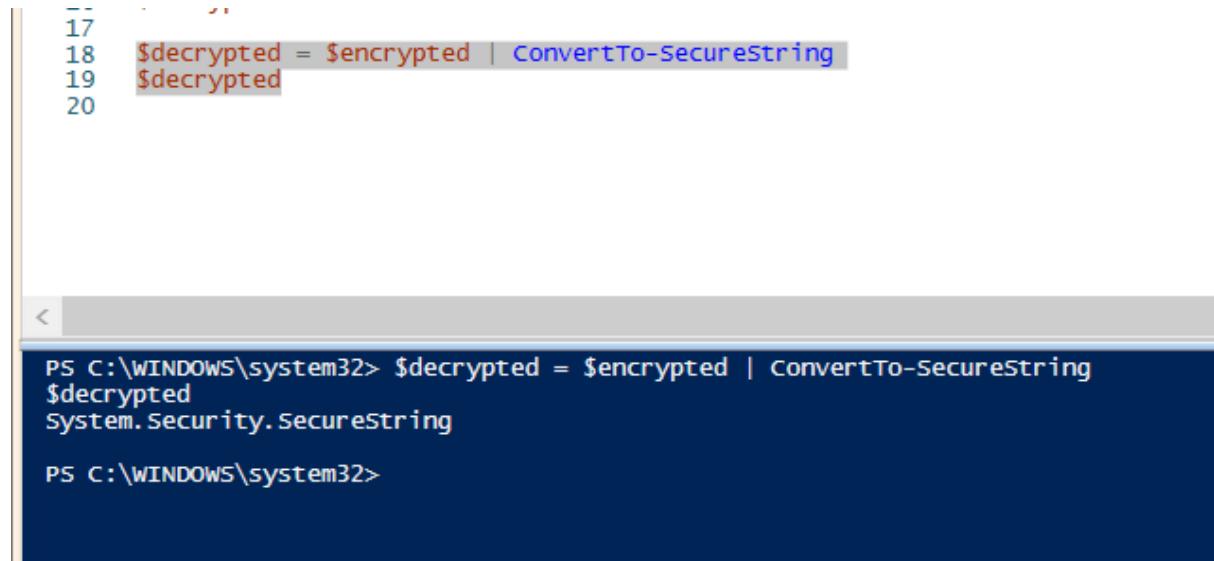
## Step 19

Convert the SecureString into an encrypted text. The default encryption used in PowerShell is AES with a key in 16 Bytes, if not specified.

*Figure 9*

## Step 20

You can always decrypt it on the same machine. On a different machine, you will need to specify a file that stores the encryption key since it is encrypted by AES previously.



```

17
18 $decrypted = $encrypted | ConvertTo-SecureString
19 $decrypted
20
PS C:\WINDOWS\system32> $decrypted = $encrypted | ConvertTo-SecureString
$decrypted
System.Security.SecureString
PS C:\WINDOWS\system32>

```

Figure 10

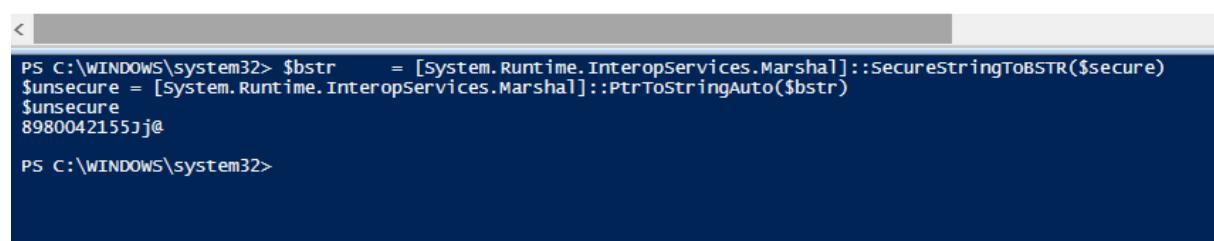
## Step 21

To convert the Secure String to plain text, you will need to copy the SecureString in the memory to an unmanaged binary string (BSTR). Then you convert the BSTR variable to a managed string.

```

20
21 $bstr    = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($secure)
22 $unsecure = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)
23 $unsecure
24

```



```

PS C:\WINDOWS\system32> $bstr    = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($secure)
$unsecure = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr)
$unsecure
8980042155@j@
PS C:\WINDOWS\system32>

```

Figure 11

## Step 22

What is the content of the three variables above?

\$secure: \_\_\_\_\_

\$encrypted: \_\_\_\_\_

\$unsecure: \_\_\_\_\_

How are they different? \_\_\_\_\_

Does SecureString date type contain encrypted information? \_\_\_\_\_

### Explanation (answer guide):

- \$secure → a SecureString object (not readable text).
- \$encrypted → AES-encrypted string (portable if you use a key).
- \$unsecure → plaintext you typed.
- **Difference:** memory-protected object vs ciphertext vs plaintext.
- **SecureString contain encrypted info?** It's protected in memory; true *portable* encryption is when you export with ConvertFrom-SecureString.

### Step 23

To specify an encryption key, you can either assign a key with a fixed value or generate a random key (a better method).

```

26
27 [Byte[]]$key = (1..16)
28 $encryptedwithkey = $secure | ConvertFrom-SecureString -Key $key
29 $encryptedwithkey
30

PS C:\WINDOWS\system32> [Byte[]]$key = (1..16)
$encryptedwithkey = $secure | ConvertFrom-SecureString -Key $key
$encryptedwithkey
76492d1116743f0423413b16050a5345MgB8AGsATABLADMASgBxAFMANwB1AEIATQBZAHUAYQBLAggAMAA5ADqAMQBrAHCAPQA9AHwAMgAwADYAY
9BjADIANwA2AGEANABjADMAOQAwAGEAOAAzADUUAZQBmADUANwBiADIAMgA2ADIAOQBjADUAMABjAdgAMwAzADEAMAAZAGMANAAZGEAZgBkAGMANw
A5AGMAMQAyADEAMAA1AGUAMQA5ADkAZQB1ADAAMQA0ADQANwA=
PS C:\WINDOWS\system32>

```

Figure 1

### Step 24

Alternatively, you can generate a random 16-byte (128 bits) key. Write the key to a key file which can be stored in the machine where decryption is needed.

The screenshot shows a Windows PowerShell window with two command blocks. The first block generates a 16-byte key and saves it to a file named 'keyfile.txt' on the desktop. The second block lists the contents of the 'keyfile.txt' file, which consists of the numbers 1 through 13.

```

32
33
34 $rkey = New-Object Byte[] 16
35 [Security.Cryptography.RNGCryptoServiceProvider]::Create().GetBytes($rkey)
36 $rkey | Out-File ".\keyfile.txt"
37
38 $key | Out-File "$env:USERPROFILE\OneDrive\Desktop\keyfile.txt"
39 ls "$env:USERPROFILE\OneDrive\Desktop\keyfile.txt"
40

key | Out-File $env:USERPROFILE\OneDrive\Desktop\keyfile.txt
ls "$env:USERPROFILE\OneDrive\Desktop\keyfile.txt"

Directory: C:\users\vpuwa\OneDrive\Desktop

Mode                LastWriteTime      Length Name
----                - - - - -          - - - - -
-a---  9/29/2025 1:41 PM           112 keyfile.txt

PS C:\users\vpuwa\OneDrive\Desktop> $rkey = New-Object Byte[] 16
[Security.Cryptography.RNGCryptoServiceProvider]::Create().GetBytes($rkey)
$rkey | Out-File ".\keyfile.txt"
ls "$env:USERPROFILE\OneDrive\Desktop\keyfile.txt"

Directory: C:\users\vpuwa\OneDrive\Desktop

Mode                LastWriteTime      Length Name
----                - - - - -          - - - - -
-a---  9/29/2025 1:42 PM           112 keyfile.txt

PS C:\users\vpuwa\OneDrive\Desktop>

```

A separate window titled 'Second L' is open, showing the contents of 'keyfile.txt' as plain text. The file contains the numbers 1 through 13.

File	Edit	View	H1
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			

Ln 5, Col 2 | 39 characters | Plain text

Figure 13

## Step 25

Write your own script to read the key from the key file, encrypt the \$secure variable from the previous step and then decryption it using the same key.

```

# === Generate a 16-byte AES key and save as raw bytes ===

$rkey = New-Object byte[] 16

[Security.Cryptography.RandomNumberGenerator]::Create().GetBytes($rkey)

Set-Content -Path .\keyfile.bin -Value $rkey -Encoding Byte # <-- raw bytes

# === Read the key back as bytes (must be 16/24/32) ===

$keyIn = Get-Content -Path .\keyfile.bin -Encoding Byte

if ($keyIn.Length -notin 16,24,32) { throw "Key must be 16, 24, or 32 bytes." }

# === Encrypt your SecureString with the key ===

```

```
$cipher = ConvertFrom-SecureString -SecureString $secure -Key $keyIn  
Set-Content -Path .\secret.enc -Value $cipher -NoNewline          # avoid extra  
newline/BOM  
  
# === Decrypt back to SecureString with the same key ===  
$secure2 = Get-Content -Path .\secret.enc -Raw | ConvertTo-SecureString -Key $keyIn  
  
# === (Optional) show plaintext to prove success ===  
$bstr2    = [Runtime.InteropServices.Marshal]::SecureStringToBSTR($secure2)  
$plaintext = [Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr2)  
Write-Host "Decrypted plaintext: $plaintext"
```

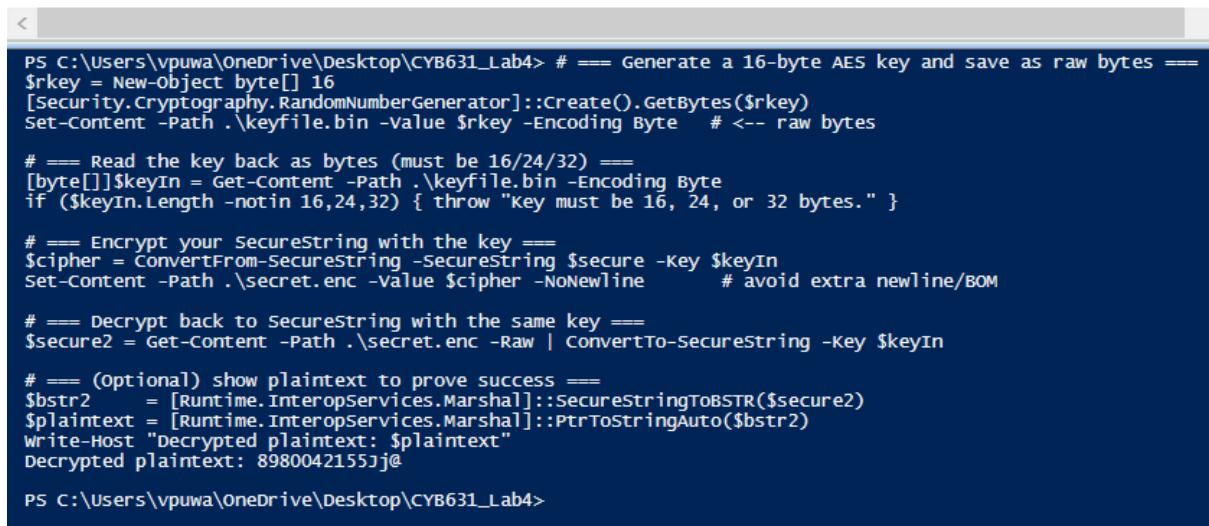
## Step 26

Here is the result:

```

33 # === Generate a 16-byte AES key and save as raw bytes ===
34 $rkey = New-Object byte[] 16
35 [Security.Cryptography.RandomNumberGenerator]::Create().GetBytes($rkey)
36 Set-Content -Path .\keyfile.bin -value $rkey -Encoding Byte # <-- raw bytes
37
38 # === Read the key back as bytes (must be 16/24/32) ===
39 [byte[]]$keyIn = Get-Content -Path .\keyfile.bin -Encoding Byte
40 if ($keyIn.Length -notin 16,24,32) { throw "Key must be 16, 24, or 32 bytes." }
41
42 # === Encrypt your SecureString with the key ===
43 $cipher = ConvertFrom-Securestring -Securestring $secure -Key $keyIn
44 Set-Content -Path .\secret.enc -value $cipher -NoNewline # avoid extra newline/BOM
45
46 # === Decrypt back to Securestring with the same key ===
47 $secure2 = Get-Content -Path .\secret.enc -Raw | ConvertTo-Securestring -Key $keyIn
48
49 # === (Optional) show plaintext to prove success ===
50 $bstr2 = [Runtime.InteropServices.Marshal]::SecureStringToBSTR($secure2)
51 $plaintext = [Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr2)
52 Write-Host "Decrypted plaintext: $plaintext"
53
54

```



```

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> # === Generate a 16-byte AES key and save as raw bytes ===
$Rkey = New-Object byte[] 16
[Security.Cryptography.RandomNumberGenerator]::Create().GetBytes($Rkey)
Set-Content -Path .\keyfile.bin -value $Rkey -Encoding Byte # <-- raw bytes

# === Read the key back as bytes (must be 16/24/32) ===
[Byte[]]$keyIn = Get-Content -Path .\keyfile.bin -Encoding Byte
if ($keyIn.Length -notin 16,24,32) { throw "Key must be 16, 24, or 32 bytes." }

# === Encrypt your SecureString with the key ===
$cipher = ConvertFrom-Securestring -Securestring $secure -Key $keyIn
Set-Content -Path .\secret.enc -value $cipher -NoNewline # avoid extra newline/BOM

# === Decrypt back to Securestring with the same key ===
$secure2 = Get-Content -Path .\secret.enc -Raw | ConvertTo-Securestring -Key $keyIn

# === (Optional) show plaintext to prove success ===
$bstr2 = [Runtime.InteropServices.Marshal]::SecureStringToBSTR($secure2)
$plaintext = [Runtime.InteropServices.Marshal]::PtrToStringAuto($bstr2)
Write-Host "Decrypted plaintext: $plaintext"

Decrypted plaintext: 8980042155j@
```

Figure 14

## Step 26

The result of Exercise is in above screenshot.

## Exercise IV: Secure Passwords

### Step 28

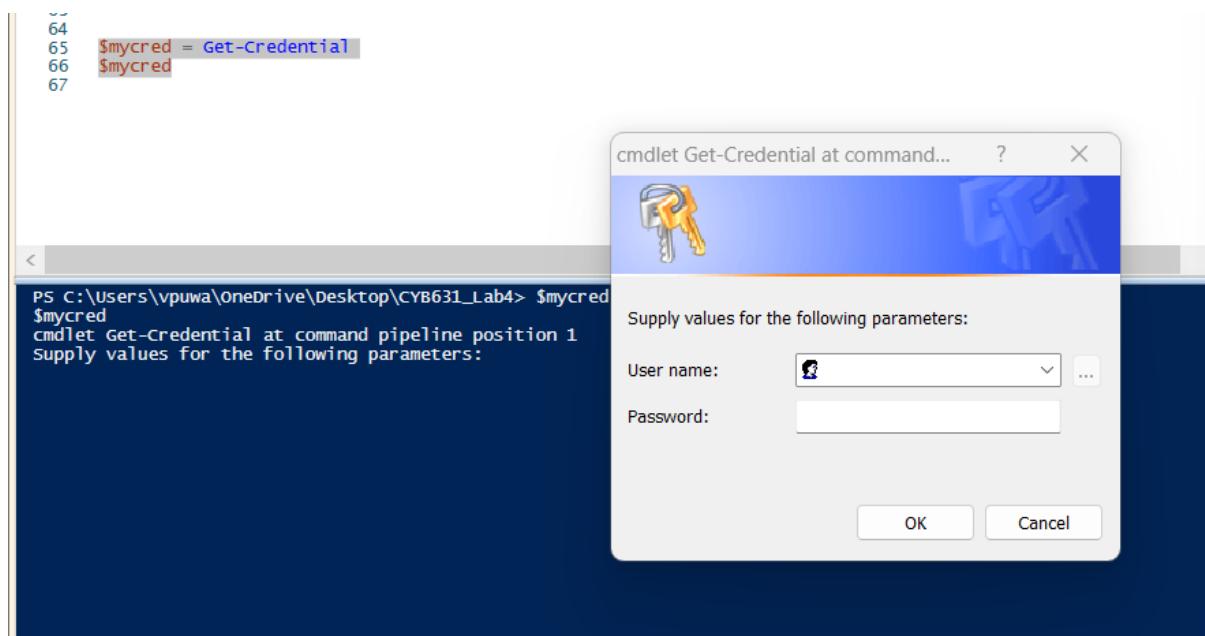
You can obtain credential information from users using Get-Credential cmdlet, which returns an object called “PSCredential”.

```
$mycred = Get-Credential
```

```
$mycred
```

### Step 29

The result is shown in the screenshot below



*Figure 15*

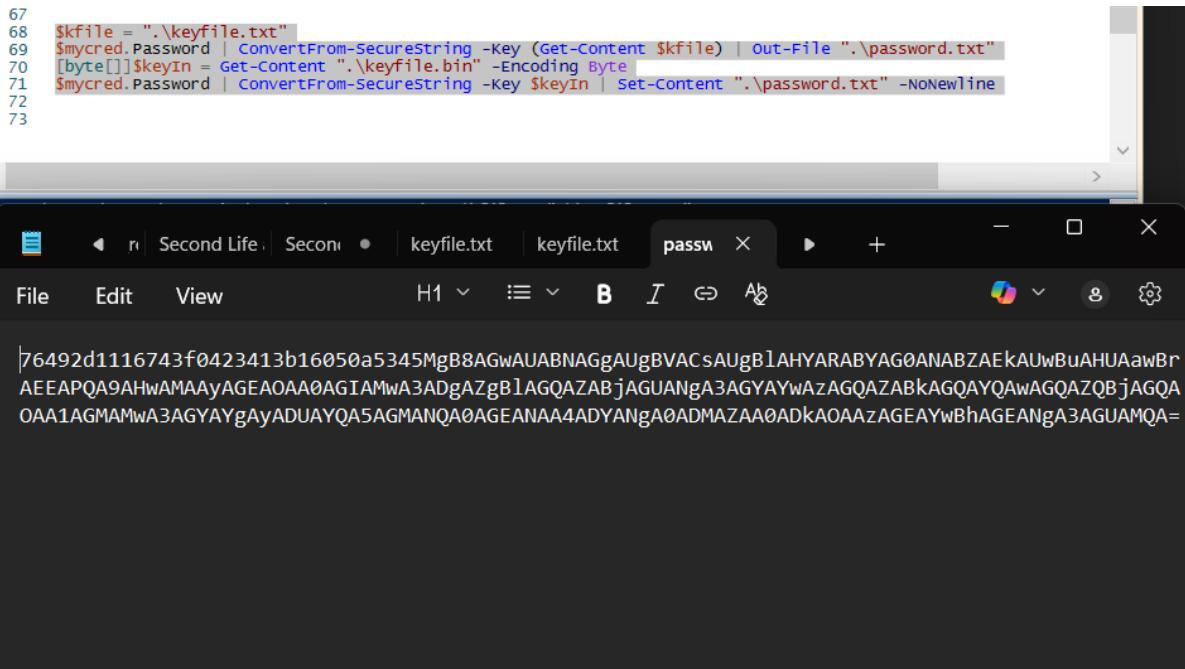
### Step 30

You can obtain the key that you generated from the previous step in the key file to encrypt the password in the credential. The encrypted password can then be stored in a file.

```
$kfile = ".\keyfile.txt"
```

```
$mycred.Password | ConvertFrom-SecureString -Key (Get-Content $kfile) | Out-File
```

```
".\password.txt"
```



The screenshot shows a Windows Notepad application window titled "password". The content of the window is a long string of characters, representing a Base64 encoded password. The code above the window shows how this password was generated from a key file and a key byte array.

```

67 $keyfile = ".\keyfile.txt"
68 $mycred.Password | ConvertFrom-SecureString -Key (Get-Content $keyfile) | Out-File ".\password.txt"
69 [byte[]]$keyIn = Get-Content ".\keyfile.bin" -Encoding Byte
70 $mycred.Password | ConvertFrom-SecureString -Key $keyIn | Set-Content ".\password.txt" -NoNewline
71
72
73

```

Figure 16

### **Step 31: What would happen if the key file in the above step is missing?**

If the key file is missing, the encrypted password stored in password.txt cannot be decrypted. The ciphertext (like the Base64 string in your file) is useless without the exact same AES key. Any attempt to run ConvertTo-SecureString -Key ... will fail, and the password cannot be recovered.

### **Step 32: Is the key file encrypted at this point? What would happen if an adversary obtained the key file?**

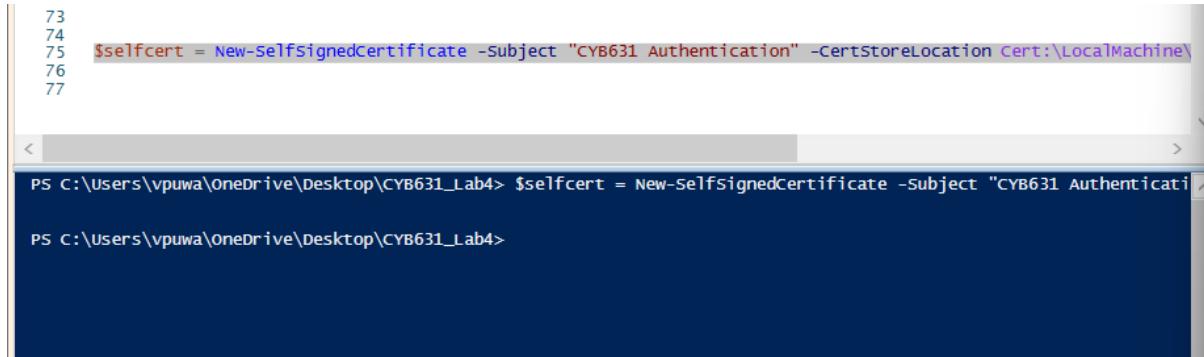
- The key file itself is **not encrypted**; it is stored in plain form as raw bytes or text.
- If an adversary obtains the key file, they can use it to decrypt any password or SecureString data encrypted with that key. This would completely compromise confidentiality.

**Best practice:** Protect the key file with strict permissions (NTFS ACLs), or use DPAPI/Windows credential vault instead of storing raw keys in the filesystem.

## Exercise V: Self-Signed Certificate and Script Signing

### Step 33

PowerShell requires scripts to be signed as a security precaution so that others cannot run malicious scripts on a protected host. Create a self-signed certificate for the computer.



```

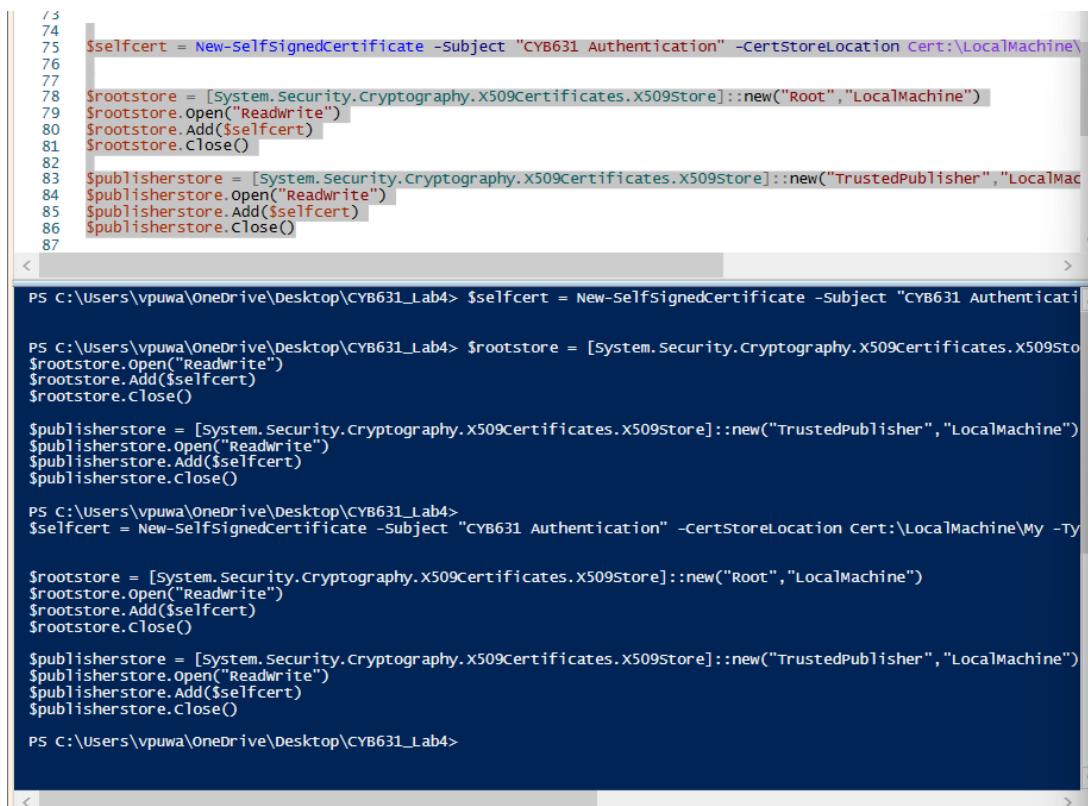
73
74 $selfcert = New-SelfSignedCertificate -Subject "CYB631 Authentication" -CertStoreLocation Cert:\LocalMachine\
75
76
77
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> $selfcert = New-SelfSignedCertificate -subject "CYB631 Authentication"
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 17

### Step 34

Store the self-signed certificate, selcert, in the machine's Root Store and Trusted Publisher's Store.



```

73
74 $selfcert = New-SelfSignedCertificate -Subject "CYB631 Authentication" -CertStoreLocation Cert:\LocalMachine\
75
76
77
78 $rootstore = [System.Security.Cryptography.X509Certificates.X509Store]::new("Root", "LocalMachine")
79 $rootstore.Open("ReadWrite")
80 $rootstore.Add($selfcert)
81 $rootstore.Close()
82
83 $publisherstore = [System.Security.Cryptography.X509Certificates.X509Store]::new("TrustedPublisher", "LocalMachine")
84 $publisherstore.Open("ReadWrite")
85 $publisherstore.Add($selfcert)
86 $publisherstore.Close()
87
88
89
90 PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> $selfcert = New-SelfSignedCertificate -subject "CYB631 Authentication"
91
92 PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> $rootstore = [System.Security.Cryptography.X509Certificates.X509Store]::new("Root", "LocalMachine")
93 $rootstore.Open("ReadWrite")
94 $rootstore.Add($selfcert)
95 $rootstore.Close()
96
97 $publisherstore = [System.Security.Cryptography.X509Certificates.X509Store]::new("TrustedPublisher", "LocalMachine")
98 $publisherstore.Open("ReadWrite")
99 $publisherstore.Add($selfcert)
100 $publisherstore.Close()
101
102
103
104 PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> $selfcert = New-SelfSignedCertificate -Subject "CYB631 Authentication" -CertStoreLocation Cert:\LocalMachine\My -Type CodeSign
105
106 $rootstore = [System.Security.Cryptography.X509Certificates.X509Store]::new("Root", "LocalMachine")
107 $rootstore.Open("ReadWrite")
108 $rootstore.Add($selfcert)
109 $rootstore.Close()
110
111 $publisherstore = [System.Security.Cryptography.X509Certificates.X509Store]::new("TrustedPublisher", "LocalMachine")
112 $publisherstore.Open("ReadWrite")
113 $publisherstore.Add($selfcert)
114 $publisherstore.Close()
115
116
117
118 PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 18

### Step 35

Store the self-signed

certmgr.msc

### Step 36

You should be able to see the certificate, like the window below. *Paste screenshot.*

**Explanation:** Show CYB631 Authentication under the appropriate store(s).

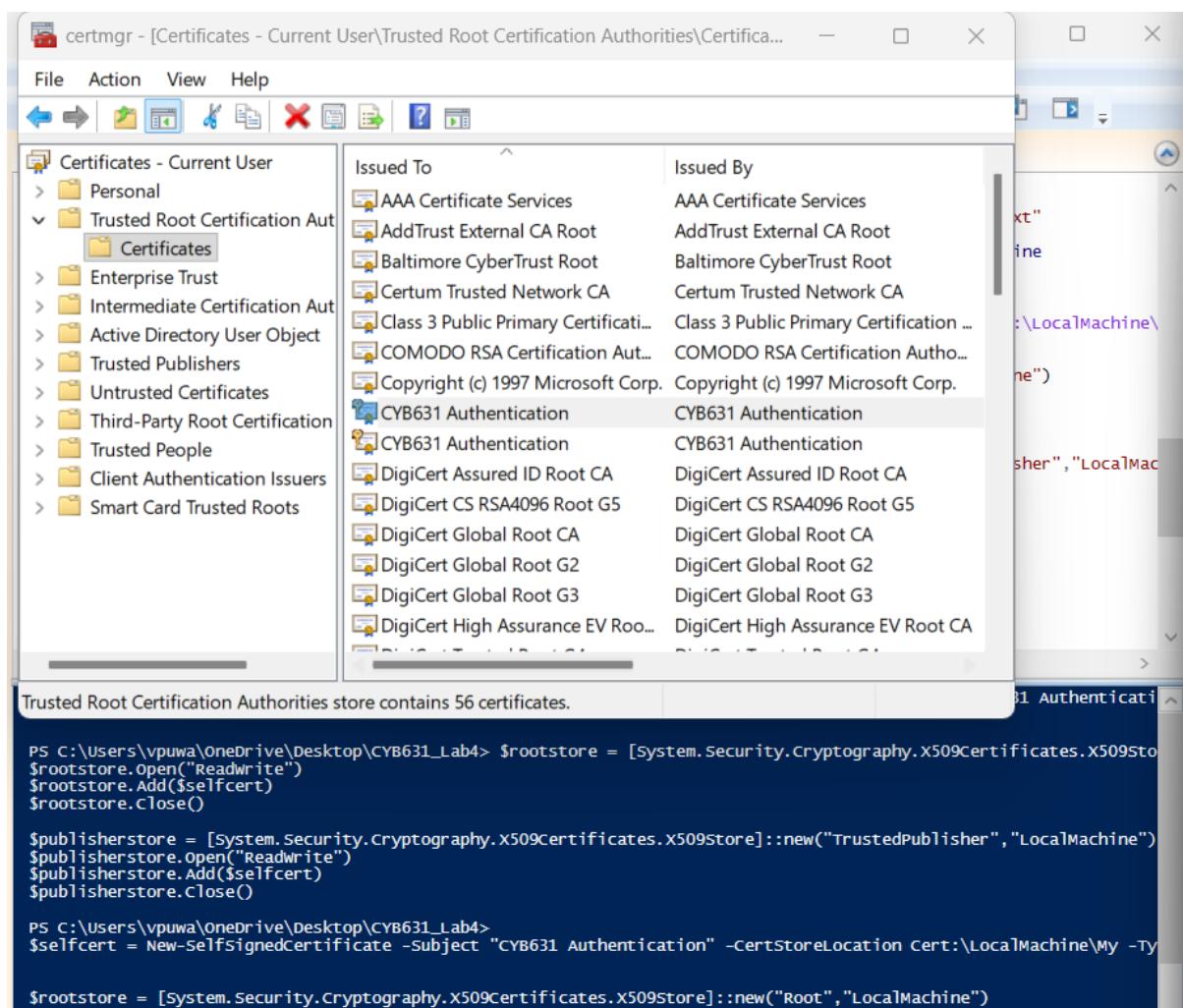


Figure 19

### Step 37

Double-click on “CYB631 Authentication” to review the content of the certificate.

*Paste screenshot.*

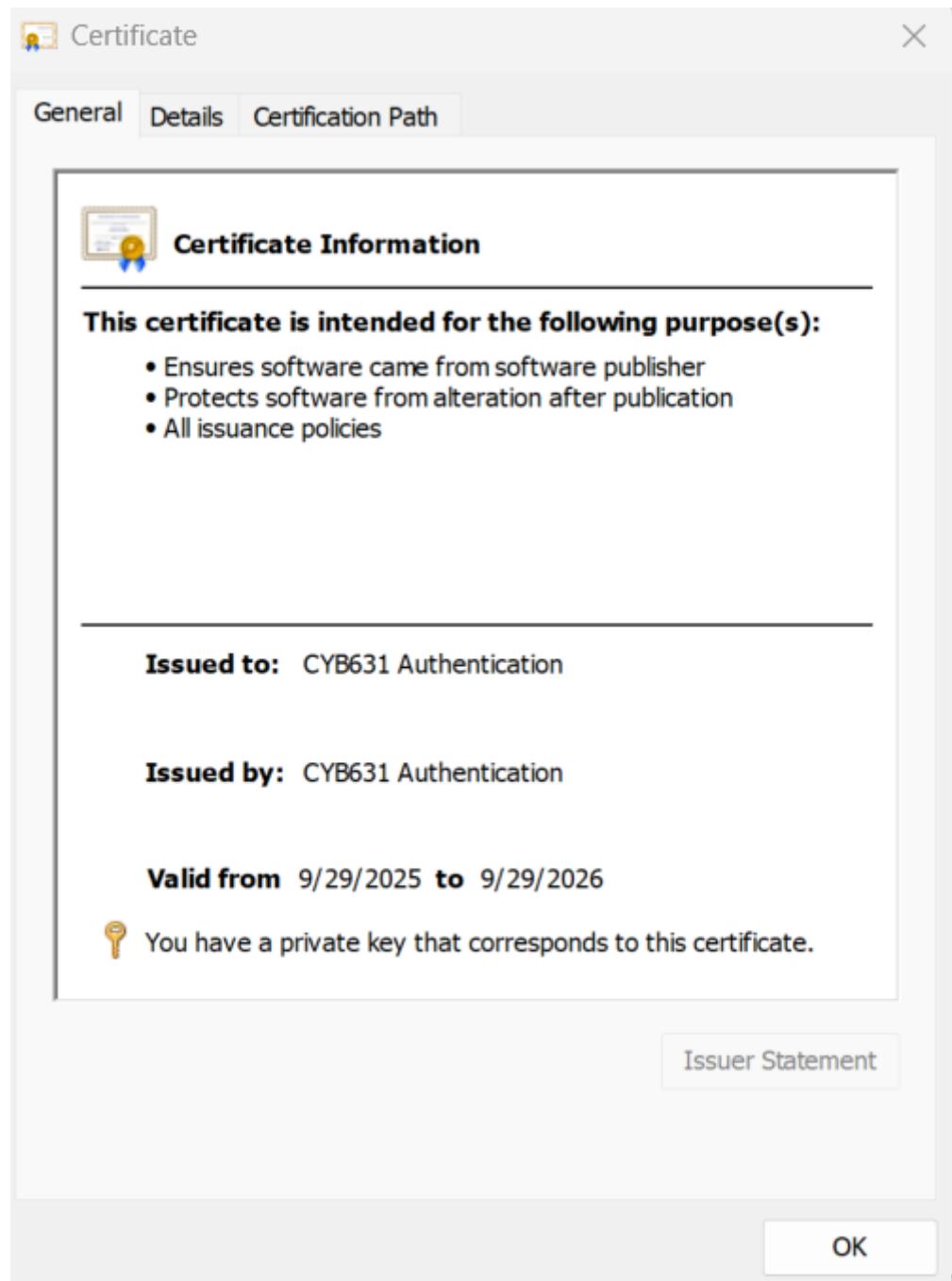


Figure 20

### Step 38

You can also use PowerShell to review the certificate in your own path, in the root store and in the trusted publisher's store. Make sure that you have done the previous step correctly.

```

Get-ChildItem Cert:\LocalMachine\My | Where-Object { $_.Subject -eq "CN=CYB631 Authentication" }

Get-ChildItem Cert:\LocalMachine\Root | Where-Object { $_.Subject -eq "CN=CYB631 Authentication" }

Get-ChildItem Cert:\LocalMachine\TrustedPublisher | Where-Object { $_.Subject -eq "CN=CYB631 Authentication" }

```

### Step 39

This is evidence you'll cite when signing.

```

91 Get-ChildItem Cert:\LocalMachine\My | Where-Object { $_.Subject -eq "CN=CYB631 Authentication" }
92 Get-ChildItem Cert:\LocalMachine\Root | Where-Object { $_.Subject -eq "CN=CYB631 Authentication" }
93 Get-ChildItem Cert:\LocalMachine\TrustedPublisher | Where-Object { $_.Subject -eq "CN=CYB631 Authentication" }
94
95

Get-ChildItem Cert:\LocalMachine\Root | Where-Object { $_.Subject -eq "CN=CYB631 Authentication" }
Get-ChildItem Cert:\LocalMachine\TrustedPublisher | Where-Object { $_.Subject -eq "CN=CYB631 Authentication" }

PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\My
Thumbprint                               Subject
-----                                 -----
C8637F3353554A1890EEB80E4195D7682F7F466C CN=CYB631 Authentication
5A89E5AC7BE4FDA4E6B88BD671C34B887FD9C4A7 CN=CYB631 Authentication

PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\Root
Thumbprint                               Subject
-----                                 -----
C8637F3353554A1890EEB80E4195D7682F7F466C CN=CYB631 Authentication
5A89E5AC7BE4FDA4E6B88BD671C34B887FD9C4A7 CN=CYB631 Authentication

PSParentPath: Microsoft.PowerShell.Security\Certificate::LocalMachine\TrustedPublisher
Thumbprint                               Subject
-----                                 -----
C8637F3353554A1890EEB80E4195D7682F7F466C CN=CYB631 Authentication
5A89E5AC7BE4FDA4E6B88BD671C34B887FD9C4A7 CN=CYB631 Authentication

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 21

### Step 40

Create a simple PowerShell script for testing certificate signing. *Paste screenshot*. We will sign one of them using the self-signed certificate. For example, creating a file called “myscript.ps1” containing the following codes:

```
Write-Host "I am on cloud nine!"
```

## Step 41

By default, any new PowerShell scripts have to be signed unless the Execution Policy is changed for that script. You should encounter an error when running myscript.ps1.

```
.\myscript.ps1
```

```
Untitled1.ps1* myscript.ps1* X
1 write-Host "I am on cloud nine!"
2
3
4
5
6 $codecertificate = Get-childItem Cert:\LocalMachine\My | where-object { $_.Subject -eq "CN=CYB631 Authentication" }
7 $codecertificate
8
9 Set-Authenticodesignature -FilePath .\myscript.ps1 -Certificate $codecertificate -TimestampServer http://timestamp.digicert.com
10

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> Write-Host "I am on cloud nine!"
I am on cloud nine!
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>
$codecertificate = Get-childItem Cert:\LocalMachine\My | where-object { $_.Subject -eq "CN=CYB631 Authentication" }
$codecertificate
Set-Authenticodesignature -FilePath .\myscript.ps1 -Certificate $codecertificate -TimestampServer http://timestamp.digicert.com

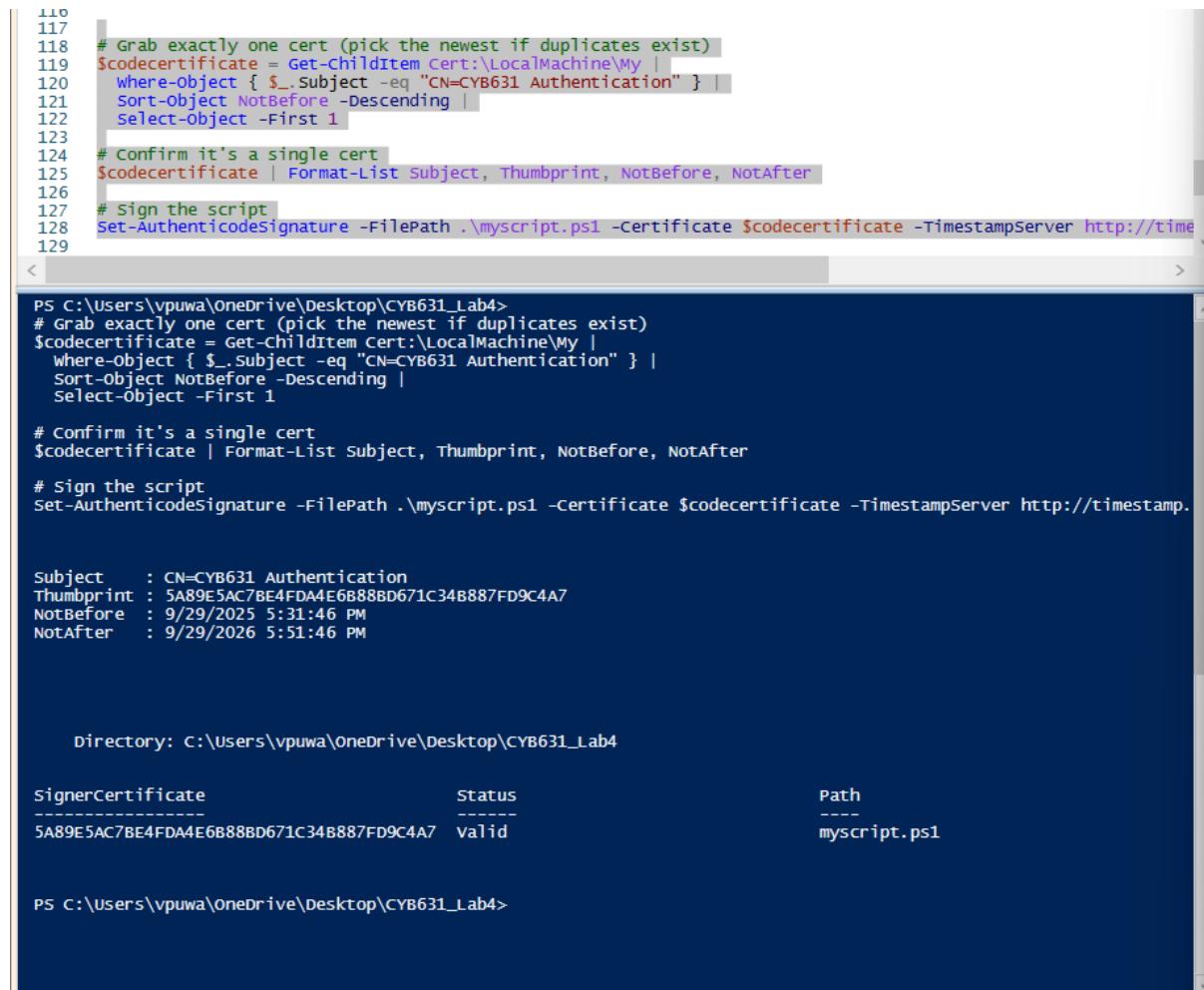
PSParentPath: Microsoft.PowerShell.Security\certificate::LocalMachine\My
Thumbprint                               Subject
-----                                 -----
C8637F3353554A189DEEB80E4195D7682F7F466C CN=CYB631 Authentication
5A89E5AC7BE4FDA4E6888BD671C348887FD9C4A7 CN=CYB631 Authentication
Set-Authenticodesignature : Cannot convert 'System.Object[]' to the type
 'System.Security.Cryptography.X509Certificates.X509Certificate2' required by parameter 'Certificate'. Specified
 method is not supported.
At line:6 char:65
+ ... nature -FilePath .\myscript.ps1 -Certificate $codecertificate -Timest ...
+               + CategoryInfo          : InvalidArgument: (:) [Set-Authenticodesignature], ParameterBindingException
+               + FullyQualifiedErrorId : CannotConvertArgument,Microsoft.PowerShell.Commands.SetAuthenticodesignatureComma
nd

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>
```

## Step 42

We will sign myscript.ps1 with the certificate that we just created. You will need to retrieve it from your own path and then use it to sign. We are using DigiCert's time server for

stamping the certificate. Since DigiCert's time server is on the Internet, you will need Internet access for this script to work.



```

116
117 # Grab exactly one cert (pick the newest if duplicates exist)
118 $codecertificate = Get-ChildItem Cert:\LocalMachine\My |
119     Where-Object { $_.Subject -eq "CN=CYB631 Authentication" } |
120     Sort-Object NotBefore -Descending |
121     Select-Object -First 1
122
123 # Confirm it's a single cert
124 $codecertificate | Format-List Subject, Thumbprint, NotBefore, NotAfter
125
126 # Sign the script
127 Set-Authenticodesignature -FilePath .\myscript.ps1 -Certificate $codecertificate -TimestampServer http://timestamp
128
129

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>
# grab exactly one cert (pick the newest if duplicates exist)
$codecertificate = Get-ChildItem Cert:\LocalMachine\My |
    Where-Object { $_.Subject -eq "CN=CYB631 Authentication" } |
    Sort-Object NotBefore -Descending |
    Select-Object -First 1

# confirm it's a single cert
$codecertificate | Format-List Subject, Thumbprint, NotBefore, NotAfter

# Sign the script
Set-Authenticodesignature -FilePath .\myscript.ps1 -Certificate $codecertificate -TimestampServer http://timestamp.

Subject      : CN=CYB631 Authentication
Thumbprint   : 5A89E5AC7BE4FDA4E6B88BD671C34B887FD9C4A7
NotBefore    : 9/29/2025 5:31:46 PM
NotAfter     : 9/29/2026 5:51:46 PM

Directory: C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4

signerCertificate          Status          Path
-----                    -----          -----
5A89E5AC7BE4FDA4E6B88BD671C34B887FD9C4A7  Valid        myscript.ps1

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 22

### Step 43

View the signature block

```

130
131  notepad .\myscript.ps1
132
133

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> notepad .\myscript.ps1
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

myscript.ps1

File Edit View

Write-Host "I am on cloud nine!"

# SIG # Begin signature block
# MIIb4wYJKoZIhvCNAQcCoIIB1DCG9ACAQExCzAJBgUrDgMCggUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQABBAfzDtgwUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAh0FAAQUqp8KfwLW4i740Gtbx65MnNm3
# TDugghZOMIIDEDCCAfigAwIBAgIQIgqGz1EPoaJGIYDxXkZaHDANBgkqhkiG9w0B
# AQsFADAgMR4wHAYDVQQDBVDWUI2MzEgQXV0aGVudGljYXRpb24wHhcNMjUwOTI5
# MjEZMTQ2WhcNMjYwOTI5MjE1MTQ2WjAgMR4wHAYDVQQDBVDWUI2MzEgQXV0aGVu
# dGljYXRpb24wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDtNHdeI8TH
# 9rVQh3ad40v0zQNIkkYgGMj8rKlpPZh8vt2l6eiv+ZLM2ukw9q8lwgeETxb2HaT
# C/ajD5ayV2gkWpAiz9tFjYKHJkx1MfbqT1v6V7qqoVXEepiN/ZaXNjewLgbGAJJ
# OnT7i7FMi0ethhRm0H0dPm0k1niY4h0sHFVdWdkF89nT3rNX0x238n8Vcw0n2nhn

Ln 1, Col 1 | 10,063 characters | Plain text | 100% | Windows (CRLF) | UTF-8 with BOM

```

Figure 23

**Step 43**

Screenshot available in the above figure.

**Step 44**

Paste a screenshot of the script including its original script (Write-Host "I am on cloud nine!") and the first five lines of the signature block.

```
131 notePad .\myscript.ps1
132
133 Get-AuthenticodeSignature -FilePath .\myscript.ps1 | Select-Object Status, StatusMessage, SignerCertificate
134
135
136 .\myscript.ps1
137

< >
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> Get-AuthenticodeSignature -FilePath .\myscript.ps1 | select-object
.\myscript.ps1

Status StatusMessage      SignerCertificate
----- -----
valid signature verified. [Subject]...
.\myscript.ps1 : File C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4\myscript.ps1 cannot be loaded because running
scripts is disabled on this system. For more information, see about_Execution_Policies at
https://go.microsoft.com/fwlink/?LinkID=135170.
At line:3 char:1
+ .\myscript.ps1
+ ~~~~~
    + CategoryInfo          : SecurityError: () [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>
```

Figure 24

#### Step 44

Validate the signature of the file.

```

138 # See what's enforcing the block
139 Get-ExecutionPolicy -List
140
141 # Allow only signed scripts for THIS PowerShell session (safest for the lab)
142 Set-ExecutionPolicy -Scope Process -ExecutionPolicy AllSigned -Force
143
144 # (If the file was downloaded or copied from the internet, clear Zone.Identifier)
145 Unblock-File .\myscript.ps1
146
147 # Run again
148 .\myscript.ps1
149
150
151 powershell.exe -NoProfile -ExecutionPolicy Bypass -File .\myscript.ps1
152
153
154
155

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> # See what's enforcing the block
Get-ExecutionPolicy -List

# Allow only signed scripts for THIS PowerShell session (safest for the lab)
Set-ExecutionPolicy -Scope Process -ExecutionPolicy AllSigned -Force

# (If the file was downloaded or copied from the internet, clear Zone.Identifier)
Unblock-File .\myscript.ps1

# Run again
.\myscript.ps1

powershell.exe -NoProfile -ExecutionPolicy Bypass -File .\myscript.ps1


I am on cloud nine!
    Scope ExecutionPolicy
    -----
MachinePolicy      Undefined
UserPolicy        Undefined
Process          Undefined
CurrentUser       Restricted
LocalMachine     Undefined
I am on cloud nine!

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 25

## Step 46

Run myscript.ps1 again. It should work this time since it is now signed.

.\myscript.ps1

→ **Explanation:** If your Execution Policy is strict, temporarily allow signed scripts only:

Set-ExecutionPolicy -Scope Process -ExecutionPolicy AllSigned -Force

Unblock-File .\myscript.ps1

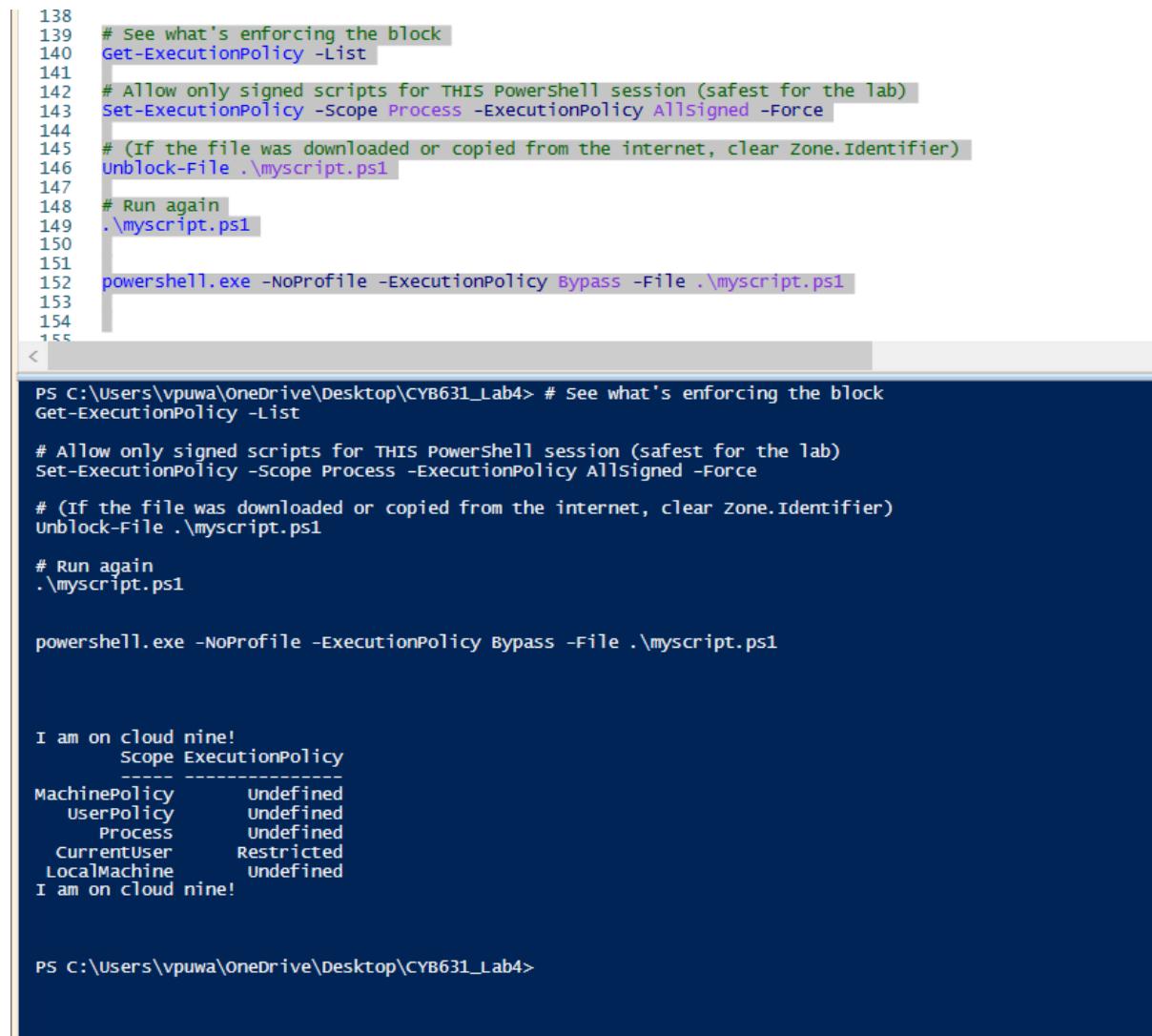
.\myscript.ps1

Or, as a fallback for testing:

```
powershell.exe -NoProfile -ExecutionPolicy Bypass -File .\myscript.ps1
```

## Step 47

The result for running is here.



```

138 # See what's enforcing the block
139 Get-ExecutionPolicy -List
140
141 # Allow only signed scripts for THIS PowerShell session (safest for the lab)
142 Set-ExecutionPolicy -Scope Process -ExecutionPolicy AllSigned -Force
143
144 # (If the file was downloaded or copied from the internet, clear Zone.Identifier)
145 Unblock-File .\myscript.ps1
146
147 # Run again
148 .\myscript.ps1
149
150
151
152 powershell.exe -NoProfile -ExecutionPolicy Bypass -File .\myscript.ps1
153
154
155

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> # See what's enforcing the block
Get-ExecutionPolicy -List

# Allow only signed scripts for THIS PowerShell session (safest for the lab)
Set-ExecutionPolicy -Scope Process -ExecutionPolicy AllSigned -Force

# (If the file was downloaded or copied from the internet, clear zone.Identifier)
Unblock-File .\myscript.ps1

# Run again
.\myscript.ps1

powershell.exe -NoProfile -ExecutionPolicy Bypass -File .\myscript.ps1

I am on cloud nine!
Scope ExecutionPolicy
-----
MachinePolicy      Undefined
UserPolicy        Undefined
Process          Undefined
CurrentUser       Restricted
LocalMachine     Undefined
I am on cloud nine!

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 26

## Exercise VI: Encryption and Decryption using a Certificate

### Step 48

Digital certificates facilitate public key encryption. Protect-CmsMessage uses public key encryption. The public key is contained in the certificate.

```

157
158 New-SelfSignedCertificate -DnsName cyberusr -CertStoreLocation "cert:\currentUser\My" -KeyUsage KeyEncipherment, DataEncipherment
159
160
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> New-SelfSignedCertificate -DnsName cyberusr -CertStoreLocation "cert:\currentUser\My" -KeyUsage KeyEncipherment, DataEncipherment

PSParentPath: Microsoft.PowerShell.Security\certificate::currentUser\My
Thumbprint          Subject
-----            -----
C553B9609482E5065792AE1B9092495D3DBF2E4D CN=cyberusr

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 27

### Step 49

You can view the certificate by the following.

```

161
162 Get-ChildItem -Path Cert:\CurrentUser\My -DocumentEncryptionCert
163
164
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> Get-ChildItem -Path Cert:\CurrentUser\My -DocumentEncryptionCert

PSParentPath: Microsoft.PowerShell.Security\certificate::currentUser\My
Thumbprint          Subject
-----            -----
C553B9609482E5065792AE1B9092495D3DBF2E4D CN=cyberusr

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 28

### Step 50

Encrypt a message and store it in a file p1.txt. The message is encrypted using the public key stored in the certificate.

```

164 "This is a secret!" | Protect-CmsMessage -To CN=cyberusr -OutFile p1.txt
165
166 PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> "This is a secret!" | Protect-CmsMessage -To CN=cyberusr -OutFile p1.txt
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> LS

Directory: C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4

Mode                LastWriteTime       Length Name
----                -----        ---- 
-a---    9/29/2025  1:54 PM           16 keyfile.bin
-a---    9/29/2025  1:45 PM          144 keyfile.txt
-a---    9/29/2025  5:59 PM        10219 myscript.ps1
-a---    9/29/2025  6:41 PM           630 p1.txt
-a---    9/29/2025  5:17 PM          276 password.txt
-a---    9/29/2025  1:54 PM          276 secret.enc

PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 29

**Step 51**

Examine p1.txt. Paste a screenshot of the file to show the first 5 lines.

```

-----BEGIN CMS-----
MIIBqwYJKoZIhvcNAQcDoIIBnDCCAZgCAQAxggFDMIIBPwIBADAnMBMxETAPBgNVBAMCGN5YmVydXNyAhBnGlnmIYU5rUu68TR9b+1gMA0GCSqGSIb3DQEBBzAABIIBALSj8vfyY0NsmPq9AKXI0x3uGPnBEE52oe+0kJITtLmePjTVPBL6DGZdHiWZo/TuKsmtM1weI7GfssRfaZQMSCTMwUwMdAcy0UPl2s14jWUpHqeB5Dcsu+KoY5CLavKvw9Cp3osDu+gx0imw2MOE8osozGeHTtnWVBj/tmLjelAiY0wuCXcMEBVUX6JuTAEy7GFBIjYb5drRrtJy5SFBLEXg/BG83TrxgKRDNoDl48MYh83B+mgxJyLjEZp/+22q6W91ZCtjHRJgqw1hCUF1z4xusM5Jq4Q+s1G/5MNGqchbSz9uN0LVBDjj64Ihok1VGvV3hXxDxn4j9HsnK8zzGowTAYJKoZIhvcNAQcBMB0GCWCGSASF1AwQBKgQQh+cgEpHFpdld+XKe4e7oLYAgzxAHhz8Wgau9lse+/IWMygfke1xwWD+B5WFE2mcFGso=
-----END CMS-----

```

Figure 30

**Step 52**

To decrypt the file, you will need to log in with the user credential that created the certificate and has the private key (the password for the account).

```

171
172 Unprotect-CmsMessage -Path p1.txt
173
<
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> Unprotect-CmsMessage -Path p1.txt
This is a secret!
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 31

**Step 53**

For others (other users or other hosts) to send you an encrypted file, you will need to export your certificate for others to import. There is a set of PKI cmdlets that can be used to manage digital certificates. (*Reference: Microsoft Learn PKI cmdlets*)

**Step 54**

```

166
167
168 $encCert = Get-ChildItem Cert:\CurrentUser\My -DocumentEncryptionCert | Where-Object { $_.Subject -eq "CN=cyberusr" }
169 Export-Certificate -Cert $encCert -FilePath .\cyberusr.cer
170
<
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>
$encCert = Get-ChildItem Cert:\CurrentUser\My -DocumentEncryptionCert | Where-Object { $_.Subject -eq "CN=cyberusr" }
Export-Certificate -Cert $encCert -FilePath .\cyberusr.cer

Directory: C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4

Mode                LastWriteTime          Length Name
----                -----          ----
-a----   9/29/2025 6:45 PM           784  cyberusr.cer

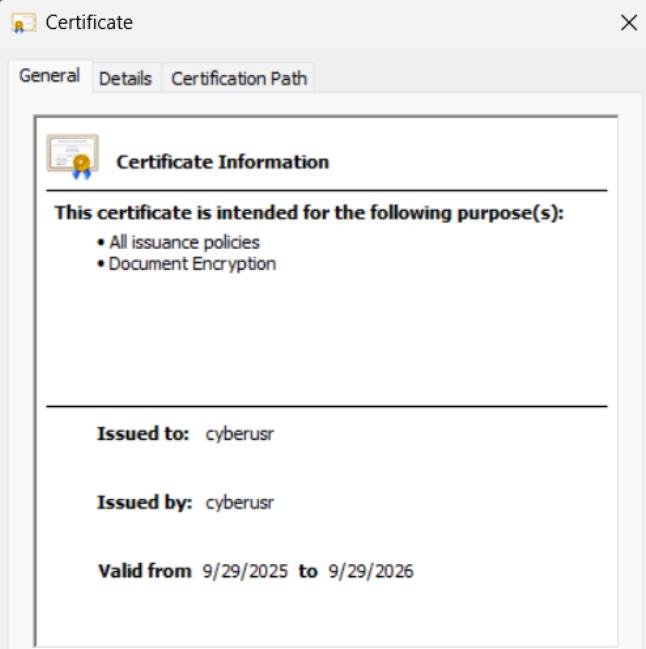
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>

```

Figure 32

**Step 55**

Paste a snapshot of the content in the certificate file exported above to show that you completed it.

```
1/3  
176 $cert = Get-childItem Cert:\CurrentUser\My | where-object { $_.Subject -eq "CN=cyberusr" }  
177  
178 $rootStore = New-Object System.Security.Cryptography.X509Certificates.X509Store "Root", "CurrentUser"  
179 $rootStore.Open("ReadWrite")  
180 $rootStore.Add($cert)  
181 $rootStore.Close()  
182  
183  
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4> $cert = Get-childItem Cert:\CurrentUser\My | where-object { $_.Subject -eq "CN=cyberusr" }  
$rootStore = New-Object System.Security.Cryptography.X509Certificates.X509Store "Root", "CurrentUser"  
$rootStore.Open("ReadWrite")  
$rootStore.Add($cert)  
$rootStore.Close()  
PS C:\Users\vpuwa\OneDrive\Desktop\CYB631_Lab4>  
  


The screenshot shows a Windows certificate dialog box. At the top, it says 'Certificate' with tabs for 'General', 'Details', and 'Certification Path'. The 'General' tab is selected. It displays the following information:



- Certificate Information
- This certificate is intended for the following purpose(s):
  - All issuance policies
  - Document Encryption
- Issued to: cyberusr
- Issued by: cyberusr
- Valid from 9/29/2025 to 9/29/2026



At the bottom right of the dialog box are buttons for 'Install Certificate...' and 'Issuer Statement'. The status bar at the bottom of the window says 'Completed'.


```

Figure 33

## Exercise VII: Lab and Class Reflection

### What did you like about this lab?

I liked that it connected core confidentiality concepts to real, hands-on PowerShell tasks. Hashing files with Get-FileHash, verifying vendor downloads, working with SecureString and explicit AES keys, signing scripts with a self-signed **Code Signing** cert, and using CMS (Protect-CmsMessage / Unprotect-CmsMessage) made the theory tangible. The evidence-driven workflow (commands → screenshot → one-line takeaway) also mirrors how we'd document in a SOC or audit.

### What were the challenges?

- Managing certificates across the correct stores (CurrentUser\My vs Root) and choosing the right cert **type** (CodeSigning vs DocumentEncryptionCert).
- Execution policy and tooling on Windows Server 2022 (PowerShell ISE not installed by default; needing VS Code or adding the ISE feature).
- Consistency of key material (binary vs text) for ConvertFrom-SecureString -Key/ConvertTo-SecureString -Key.
- CMS recipient selection (matching CN exactly) and understanding that CMS encrypts to the **public key**, so decryption must use the original user profile/private key.
- Small but time-consuming issues: path/permission/UAC prompts, and aligning screenshots with clear captions.

### Suggestions to help me learn better:

- Provide a short “pre-flight checklist” (tooling, execution policy, cert store notes, timestamp server URL).

- Include one end-to-end example per section (hashing → compare, CMS → encrypt/decrypt, signing → verify) with a model screenshot + caption so expectations are crystal clear.
- Ship a tiny sample dataset (3–5 files) for hashing and a known safe download link + published hash to avoid web hunting.
- Recommend VS Code as the default editor on Server 2022 and note how to install ISE if required.
- Add a troubleshooting appendix (common CMS errors, cert selection tips, key length rules for AES, “signature status: Unknown” fixes).
- Optional: a cleanup script to remove generated keys/certs/files after submission.

## VM power-down

Acknowledged. I've powered off my Windows VM on the server after completing and verifying all screenshots.