Power Shell 101

Module 10 Hands-on Activity – PowerShell in VSCode

6/8/2019 Developed by Jin Chang
6/8/2019 Tested by Sion Yoon
Center for Information Assurance (CIAE) at City University of Seattle

Learning Outcomes

- Describe how to use Cryptography HashAlgorithm namespace to generate the message hash code with the "sha256".
- Describe how PowerShell can invoke UI components to obtain an input.

Resources

- Cryptography using the .NET hash algorithm
    - https://riptutorial.com/powershell/example/20162/security-and-cryptography
- PowerShell UI (simple text entry)
    - https://docs.microsoft.com/en-us/powershell/scripting/samples/multiple-selection-list-boxes?view=powershell-6

Activities

- How to use PowerShell cmdlet with .NET HashAlgorithm Class to generate the hash code of a given string.
- How to use UI components step-by-step

Hash Algorithm (not the same as Hash Table)

1. Hash algorithms are one of the most essential parts of providing security via cryptography technology. These functions/algorithms map binary strings of an arbitrary length to small binary strings of a fixed length, known as hash values. A cryptographic hash function has the property that it is computationally infeasible to find 2 distinct inputs that have to the same value. Hash functions are commonly used with digital signatures and for data integrity as well as backup management. (https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.hashalgorithm?view=netframework-4.8)

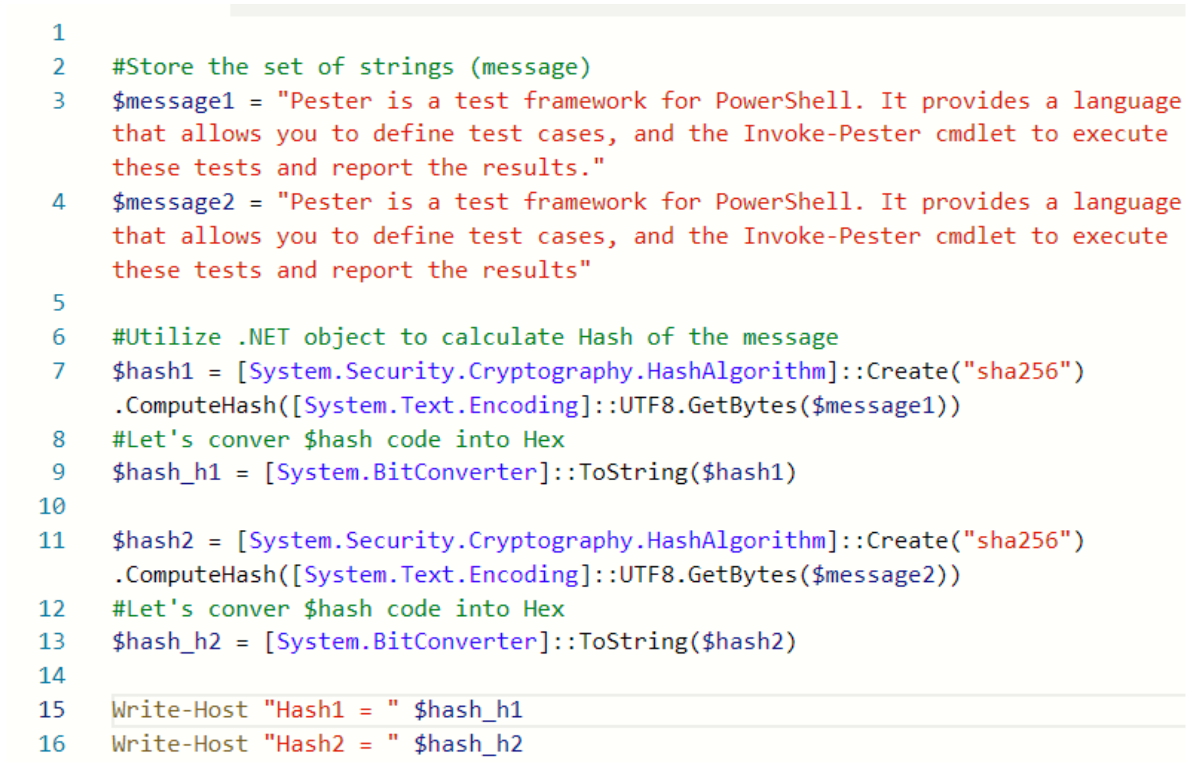
2. In this activity, we'll generate a hash code of a set of string (message) using the algorithm supported in a .NET object called System.Security.Cryptography.HashAlgorithm.

   Create a a file called TestHashAlg.ps1 under Module 10 folder, type the following statements in the new file.

   Copy paste these two messages rather than typing them in your script.

```
$message1 = "Pester is a test framework for PowerShell. It provides a
language that allows you to define test cases, and the Invoke-Pester cmdlet
to execute these tests and report the results."

$message2 = "Pester is a test framework for PowerShell. It provides a
language that allows you to define test cases, and the Invoke-Pester cmdlet
to execute these tests and report the results."
```

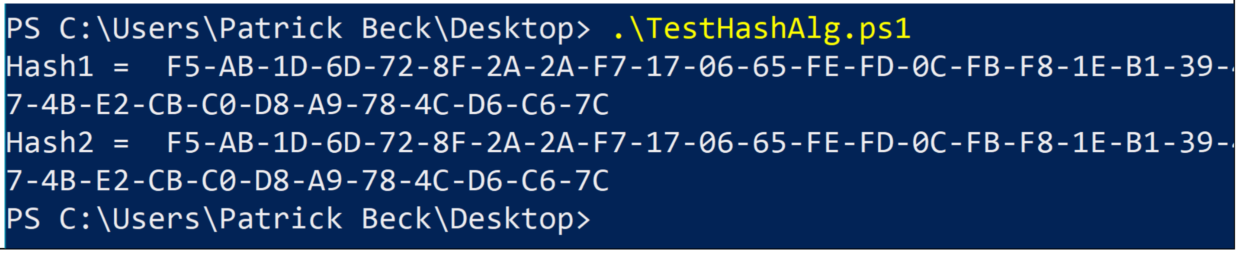
Then, now you can type the following statements in your script.



```

#Store the set of strings (message)
$message1 = "Pester is a test framework for PowerShell. It provides a language
that allows you to define test cases, and the Invoke-Pester cmdlet to execute
these tests and report the results."
$message2 = "Pester is a test framework for PowerShell. It provides a language
that allows you to define test cases, and the Invoke-Pester cmdlet to execute
these tests and report the results"

#Utilize .NET object to calculate Hash of the message
$hash1 = [System.Security.Cryptography.HashAlgorithm]::Create("sha256")
.ComputeHash([System.Text.Encoding]::UTF8.GetBytes($message1))
#Let's conver $hash code into Hex
$hash_h1 = [System.BitConverter]::ToString($hash1)

$hash2 = [System.Security.Cryptography.HashAlgorithm]::Create("sha256")
.ComputeHash([System.Text.Encoding]::UTF8.GetBytes($message2))
#Let's conver $hash code into Hex
$hash_h2 = [System.BitConverter]::ToString($hash2)

Write-Host "Hash1 = " $hash_h1
Write-Host "Hash2 = " $hash_h2
```

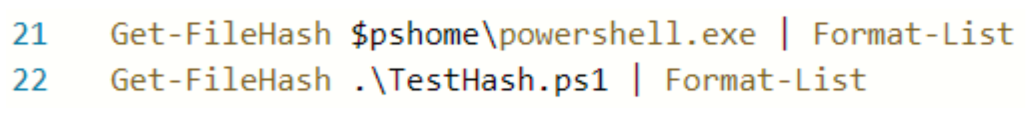
3. What discrepancy did you see between message1 and message2? Do you expect the same Hashcode from the HashAlgorithm?

Q1. In the console, run the script by typing .\TestHashAlg.ps1. Insert the screen capture of the script output.



```
PS C:\Users\Patrick Beck\Desktop> .\TestHashAlg.ps1
Hash1 =  F5-AB-1D-6D-72-8F-2A-2A-F7-17-06-65-FE-FD-0C-FB-F8-1E-B1-39-
7-4B-E2-CB-C0-D8-A9-78-4C-D6-C6-7C
Hash2 =  F5-AB-1D-6D-72-8F-2A-2A-F7-17-06-65-FE-FD-0C-FB-F8-1E-B1-39-
7-4B-E2-CB-C0-D8-A9-78-4C-D6-C6-7C
PS C:\Users\Patrick Beck\Desktop>
```

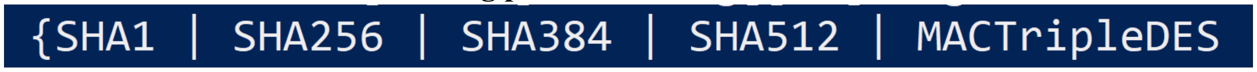
4. There is another way to use HashAlgorithm, **`Get-FileHash`**. This cmdlet calculates the hash value for a given file by using a defined algorithm. The default algorithm is set to SHA256, but there are other algorithms available. Note that a hash value is a unique value that corresponds to the content of the file. This is one of the most effective ways to identify the contents and its integrity. (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash?view=powershell-6)



```
21    Get-FileHash $pshome\powershell.exe | Format-List
22    Get-FileHash .\TestHash.ps1 | Format-List
```

**Please change ".\TestHash.ps1" to ".\TestHashAlg.ps1" on the example above (line 22).

Q2. Using Help cmdlet on Get-FileHash, find 3 different hash algorithms that provide 3 different hash codes of TestHashAlg.ps1.



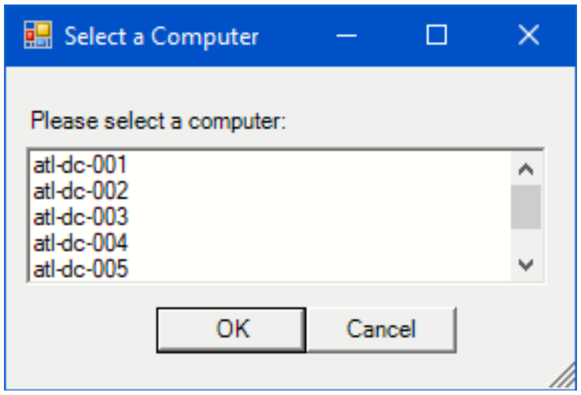
```
{SHA1 | SHA256 | SHA384 | SHA512 | MACTripleDES
```

Q3. When would you use the Hash code of a given file? (Hint: data integrity)

The type of hash code used depends on how secure you want the data that is being encrypted. SHA1 uses 20 bit, while SHA256 and 512 use 32 and 64 bit, respectively.

How to Create a Custom Input Dialog Box

5. We learned how to make an InputBox couple weeks ago but let's dive deeper into creating a custom input dialog box.



Please select a computer:

atl-dc-001
atl-dc-002
atl-dc-003
atl-dc-004
atl-dc-005

If you look at the dialog box, there are several UI components being implemented.

- Title bar ("Select a Computer")
- Label ("Please select a computer")
- List Box that contains various computer names ("atl-dc…")
- Two push buttons, "OK" and "Cancel"

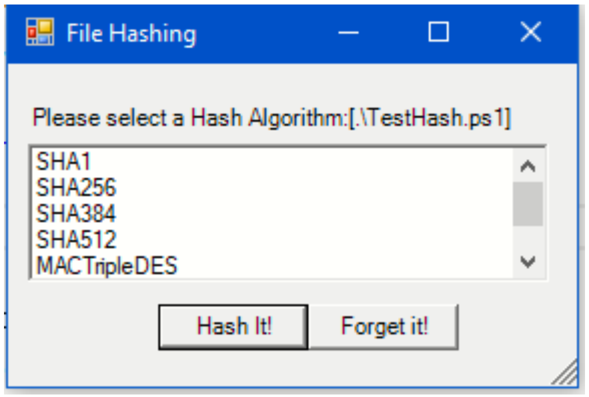
6. Notice that we have a file named *TestHash.ps1* under Module 10 folder, that contains the code that creates this dialog box from the BB link. All the necessary comments have been added, and go through each step and it is self-explanatory as you read. Then, simply run the script to see the dialog box pop-up.

Q4. Create a dialog box that displays the following UI components by modifying TestHash.ps1.

UI components to be modified:

1. **Title bar**
2. **Label by adding text called TestHash.ps1**
3. **List box to be populated by the Hash Algorithms (5 algorithms)**
4. **Push button labels.**
5.

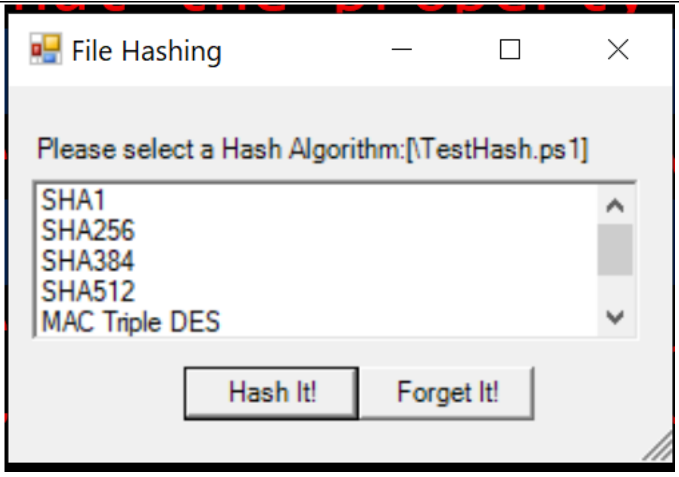
This is an example of how it should look like:



Please select a Hash Algorithm:[.\TestHash.ps1]

SHA1
SHA256
SHA384
SHA512
MACTripleDES

Insert a screen capture of your dialog box here:

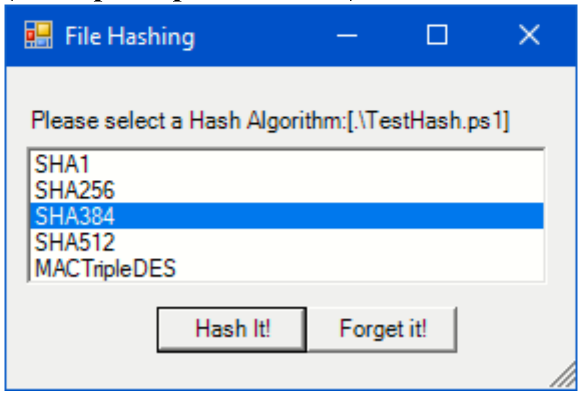


Please select a Hash Algorithm:[\TestHash.ps1]

SHA1
SHA256
SHA384
SHA512
MAC Triple DES

Q5. Using a `Get-FileHash` cmdlet from the previous step, compose the cmdlet so that the dialog box processes the input algorithm to display the hash code of the TestHash.ps1 file.

(Example Input Selection)



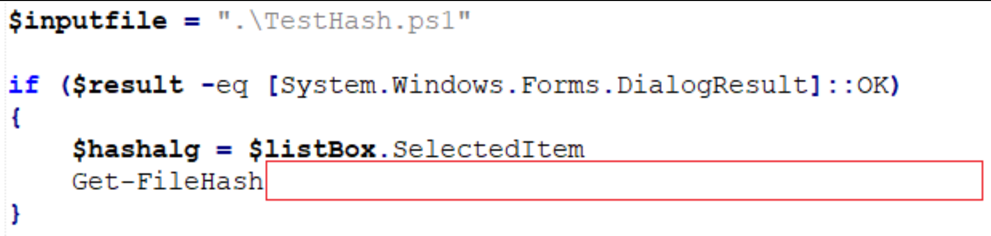
Please select a Hash Algorithm:[.\TestHash.ps1]

SHA1
SHA256
SHA384
SHA512
MACTripleDES

(Example Output in the Console)

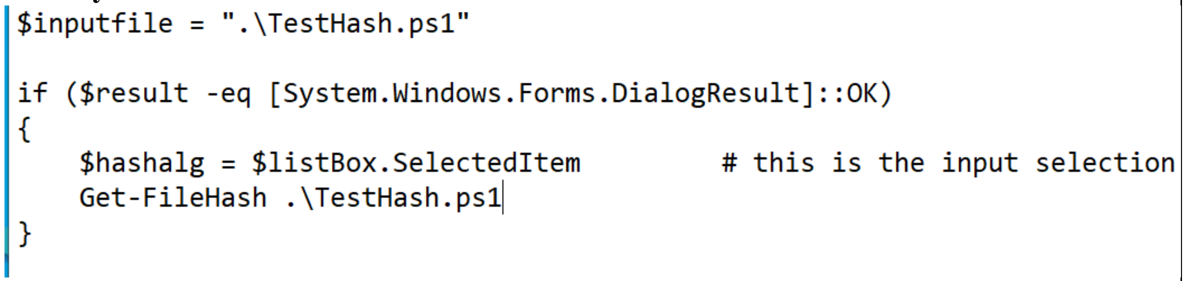
```
Algorithm : SHA384
Hash      : 23264EC73A2A846025E42C00563272A2CA3F97
Path      : C:\isec505\module10\TestHash.ps1
```

HINT: Modify the last lines of your code as shown below and fill in the area after Get-FileHash within the red box.



```
$inputfile = ".\TestHash.ps1"

if ($result -eq [System.Windows.Forms.DialogResult]::OK)
{
    $hashalg = $listBox.SelectedItem
    Get-FileHash
}
```

Insert your code here:



```
$inputfile = ".\TestHash.ps1"

if ($result -eq [System.Windows.Forms.DialogResult]::OK)
{
    $hashalg = $listBox.SelectedItem         # this is the input selection
    Get-FileHash .\TestHash.ps1
}
```