Learn

Discover ⌄   Product documentation ⌄   Development languages ⌄   Topics ⌄   🔍   Sign in

PowerShell   Overview   DSC   PowerShellGet   Utility modules   Module Browser   API Browser   Resources ⌄   **Download PowerShell**

Version

Learn / PowerShell / Microsoft.PowerShell.Security /

⊕   ✏️   ⋮

ConvertTo-SecureString

Reference   👍 Feedback

Module: Microsoft.PowerShell.Security

In this article

Syntax

Description

Examples

Parameters

Show 4 more

Converts plain text or encrypted strings to secure strings.

Syntax

```PowerShell
ConvertTo-SecureString
        [-String] <String>
        [[-SecureKey] <SecureString>]
        [<CommonParameters>]
```

```PowerShell
ConvertTo-SecureString
        [-String] <String>
        [-AsPlainText]
        [-Force]
        [<CommonParameters>]
```

```PowerShell
ConvertTo-SecureString
        [-String] <String>
        [-Key <Byte[]>]
        [<CommonParameters>]
```

Description

The `ConvertTo-SecureString` cmdlet converts encrypted standard strings into secure strings. It can also convert plain text to secure strings. It is used with `ConvertFrom-SecureString` and `Read-Host`. The secure string created by the cmdlet can be used with cmdlets or functions that require a parameter of type **SecureString**. The secure string can be converted back to an encrypted, standard string using the `ConvertFrom-SecureString` cmdlet. This enables it to be stored in a file for later use.

If the standard string being converted was encrypted with `ConvertFrom-SecureString` using a specified key, that same key must be provided as the value of the **Key** or **SecureKey** parameter of the `ConvertTo-SecureString` cmdlet.

> ⓘ **Note**
>
> Note that per **DotNet**, the contents of a SecureString are not encrypted on non-Windows systems.

Examples

Example 1: Convert a secure string to an encrypted string

This example shows how to create a secure string from user input, convert the secure string to an encrypted standard string, and then convert the encrypted standard string back to a secure string.

```PowerShell
PS C:\> $Secure = Read-Host -AsSecureString
PS C:\> $Secure
System.Security.SecureString
PS C:\> $Encrypted = ConvertFrom-SecureString -SecureString $Secure
PS C:\> $Encrypted
01000000d08c9ddf0115d1118c7a00c04fc297eb010000001a114d45b8dd3f4aa11ad7c0abdae98(
02000000000003660000a8000000100000005df63cea84bfb7d70bd6842e7efa798200000000000048(
000010000000f10cd0f4a99a8d5814d94e0687d7430b100000008bf11f1960158405b2779613e93!
0000e6b7bf46a9d485ff211b9b2a2df3bd6eb67aae41
PS C:\> $Secure2 = ConvertTo-SecureString -String $Encrypted
PS C:\> $Secure2
System.Security.SecureString
```

The first command uses the **AsSecureString** parameter of the `Read-Host` cmdlet to create a secure string. After you enter the command, any characters that you type are converted into a secure string and then saved in the `$Secure` variable.

The second command displays the contents of the `$Secure` variable. Because the `$Secure` variable contains a secure string, PowerShell displays only the **System.Security.SecureString** type.

The third command uses the `ConvertFrom-SecureString` cmdlet to convert the secure string in the `$Secure` variable into an encrypted standard string. It saves the result in the `$Encrypted` variable.

The fourth command displays the encrypted string in the value of the `$Encrypted` variable.

The fifth command uses the `ConvertTo-SecureString` cmdlet to convert the encrypted standard string in the `$Encrypted` variable back into a secure string. It saves the result in the `$Secure2` variable. The sixth command displays the value of the `$Secure2` variable. The SecureString type indicates that the command was successful.

Example 2: Create a secure string from an encrypted string in a file

This example shows how to create a secure string from an encrypted standard string that is saved in a file.

```PowerShell
$Secure = Read-Host -AsSecureString
$Encrypted = ConvertFrom-SecureString -SecureString $Secure -Key (1..16)
```

```
$Encrypted | Set-Content Encrypted.txt
$Secure2 = Get-Content Encrypted.txt | ConvertTo-SecureString -Key (1..16)
```

The first command uses the **AsSecureString** parameter of the `Read-Host` cmdlet to create a secure string. After you enter the command, any characters that you type are converted into a secure string and then saved in the `$Secure` variable.

The second command uses the `ConvertFrom-SecureString` cmdlet to convert the secure string in the `$Secure` variable into an encrypted standard string by using the specified key. The contents are saved in the `$Encrypted` variable.

The third command uses a pipeline operator (`|`) to send the value of the `$Encrypted` variable to the `Set-Content` cmdlet, which saves the value in the Encrypted.txt file.

The fourth command uses the `Get-Content` cmdlet to get the encrypted standard string in the Encrypted.txt file. The command uses a pipeline operator to send the encrypted string to the `ConvertTo-SecureString` cmdlet, which converts it to a secure string by using the specified key. The results are saved in the `$Secure2` variable.

Example 3: Convert a plain text string to a secure string

This command converts the plain text string `P@ssW0rD!` into a secure string and stores the result in the `$Secure_String_Pwd` variable.

Starting in PowerShell 7, the **Force** parameter is not required when using the **AsPlainText** parameter. However, including the **Force** parameter ensures the statement is compatible with earlier versions.

PowerShell   Copy

```
$Secure_String_Pwd = ConvertTo-SecureString "P@ssW0rD!" -AsPlainText -Force
```

> ⊗ **Caution**
>
> You should avoid using plain text strings in script or from the command line. The plain text can show up in event logs and command history logs.

Parameters

-AsPlainText

Specifies a plain text string to convert to a secure string. The secure string cmdlets help protect confidential text. The text is encrypted for privacy and is deleted from computer memory after it is used. If you use this parameter to provide plain text as input, the system cannot protect that input in this manner.

⌄⌄ Expand table

Type:	SwitchParameter
Position:	1
Default value:	None
Required:	False
Accept pipeline input:	False

Accept wildcard characters:	False

-Force

Beginning in PowerShell 7, The **Force** parameter is no longer required when using the **AsPlainText** parameter. While the parameter is not used, it was not removed to provide compatibility with earlier versions of PowerShell.

⧉ **Expand table**

Type:	SwitchParameter
Position:	2
Default value:	None
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-Key

Specifies the encryption key used to convert the original secure string into the encrypted standard string. Valid key lengths are 16, 24 and 32 bytes.

⧉ **Expand table**

Type:	Byte[]
Position:	Named
Default value:	None
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-SecureKey

Specifies the encryption key used to convert the original secure string into the encrypted standard string. The key must be provided in the format of a secure string. The secure string will be converted to a byte array to be used as the key. Valid secure key lengths are 8, 12 and 16 code points.

⧉ **Expand table**

Type:	SecureString
Position:	1
Default value:	None
Required:	False
Accept pipeline input:	False
Accept wildcard characters:	False

-String

Specifies the string to convert to a secure string.

⬚ Expand table

Type:	String
Position:	0
Default value:	None
Required:	True
Accept pipeline input:	True
Accept wildcard characters:	False

Inputs

String

You can pipe a standard encrypted string to this cmdlet.

Outputs

SecureString

This cmdlet returns the created **SecureString** object.

Notes

Some characters, such as emoticons, correspond to several code points in the string that contains them. Avoid using these characters because they may cause problems and misunderstandings when used in a password.

Related Links

- ConvertFrom-SecureString
- Read-Host

⬡ **Collaborate with us on GitHub**

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

▷ **PowerShell feedback**

PowerShell is an open source project. Select a link to provide feedback:

🐞 Open a documentation issue

🗨 Provide product feedback