Signing UEFI images for Secure Boot feature in the 14th generation and later Dell EMC PowerEdge servers

This technical white paper provides an overview about signing UEFI images for development and test by using Microsoft Windows hosted tools.

Dell Engineering
June 2016

Authors (Dell EMC Server Solutions)

Vinod P

Sreenivasula Reddy

Sheshadri PR Rao (InfoDev)

A Dell EMC Technical White Paper

Revisions

Date	Description
June 2017	Initial release

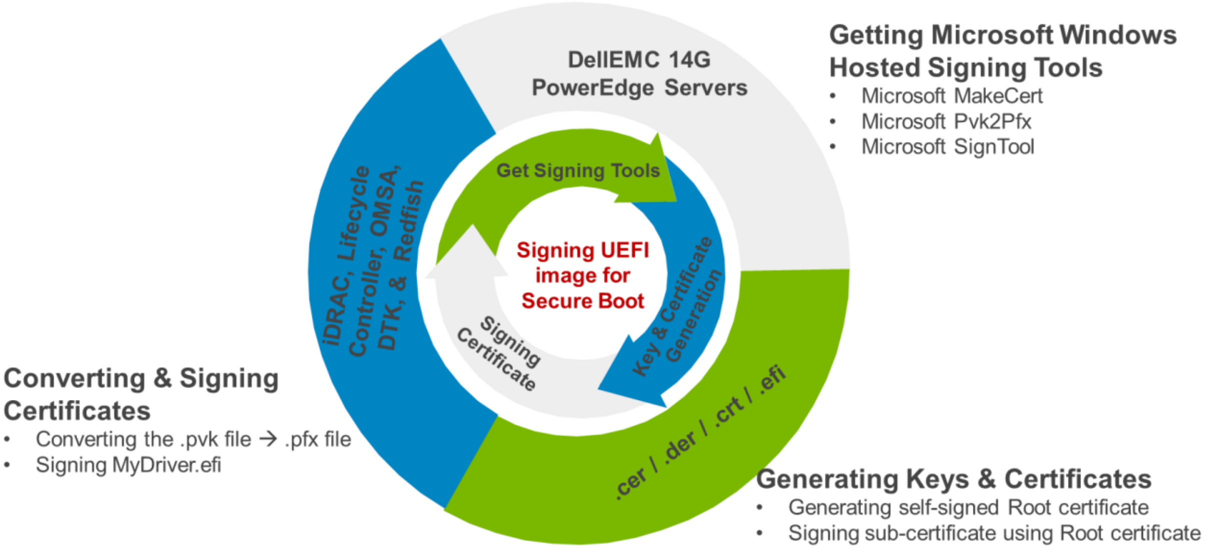
Contents

Executive summary

Before booting, the system BIOS launches a variety of code modules such as device firmware, diagnostics, and OS loaders. Secure Boot aims to distinguish trusted modules from untrusted modules. Before the system BIOS loads a module into memory, Secure Boot checks whether the module has authorization to execute. If a module does not have authorization, the system BIOS continues without loading or executing the module.

How does Secure Boot distinguish between trusted and untrusted modules? The Secure Boot policy, which the system administrator defines, specifies the rules for authorization. The policy contains keys and certificates for signed modules, and image digests for unsigned modules. Providers of pre-boot code modules sign the code with private keys, and the Secure Boot policy contains public keys needed to verify the digital signatures. Signature verification gives the system BIOS assurance that the module came from a trusted entity, and that other entities have not tampered with the module.

A Secure Boot policy consists of four components: the Platform Key (PK), the Key Exchange Key database (KEK), the Authorized Signature Database (db), and the Forbidden Signature Database (dbx). The system BIOS uses the first two components (PK and KEK) to verify changes to the Secure Boot policy itself. The last two components (db and dbx) help the system BIOS determine whether to execute a pre-boot image.


DellEMC 14G
PowerEdge Servers
Getting Microsoft Windows Hosted Signing Tools
• Microsoft MakeCert
• Microsoft Pvk2Pfx
• Microsoft SignTool
Get Signing Tools
iDRAC, Lifecycle Controller, OMSA, DTK, & Redfish
Signing UEFI image for Secure Boot
Key & Certificate Generation
Signing Certificate
.cer / .der / .crt / .efi
Converting & Signing Certificates
• Converting the .pvk file → .pfx file
• Signing MyDriver.efi
Generating Keys & Certificates
• Generating self-signed Root certificate
• Signing sub-certificate using Root certificate

Secure Boot policies contain public keys only; private keys are not saved anywhere in the system. The system BIOS uses public keys to verify signatures, while module providers use private keys to sign modules. Owners of private keys use specialized hardware and techniques for protecting the keys, such as Hardware Security Modules (HSMs), secure Smart Cards, or a Key Management System (KMS). Neither the system BIOS nor Secure Boot require private keys during the boot process.

1 Db and Dbx control image execution

First, consider the authorization of pre-boot image files:

- The Authorized Signature Database (db) contains public keys, certificates, and image digests for image files authorized to execute.
- If a pre-boot image file includes a digital signature, the BIOS verifies the signature by using the keys and certificates in the database (DB).
- If a pre-boot image file does not contain a digital signature, the system BIOS determines the digest (also known as a hash value) of the image and compares it against the image digests in db.
- The BIOS executes the image file only if it verifies the digital signature by using a key in db, or finds the digest in db.

The Forbidden Signature Database (dbx) specifies image files that must not execute even if they are allowed by the db. Similar to db, dbx may contain public keys, certificates, or hash values. The BIOS will not execute an image if it verifies the image's digital signature with a key in dbx, or finds the image's hash value in dbx.

Meaning, db acts as a “whitelist” and dbx acts as a “blacklist”. To execute an image file, Secure Boot must verify that the image file is on the “whitelist” and not on the “blacklist”. If Secure Boot does not find the image file in either list, the system BIOS will not execute the image file. Similarly, if Secure Boot finds the image file in both lists, the system BIOS will not execute the image file.

The Secure Boot policy applies to all pre-boot code image files, including device firmware and OS boot loaders. When installing expansion cards or operating systems, make sure that db includes information to authorize the images (and dbx does not forbid them). Otherwise, the images will not execute.

1.1 PK and KEK control policy updates

Second, consider changes to the Secure Boot policy itself. Periodically, administrators might add or remove entries in the policy, and attackers might attempt malicious updates to the policy. Anyone wanting to modify db or dbx must sign their modifications with the private PK or KEK. In this way, the BIOS can use the public keys in PK and KEK to verify updates to db and dbx. Therefore, if an attacker attempts to modify db or dbx, the signature verification with PK and KEK fails (because the attacker does not possess the private PK or KEK), and the system BIOS does not permit the modifications.

Also, any agent wanting to modify PK or KEK must possess the private half of PK. PK acts as a master key—anyone with access to the private half of PK can modify any portion of the Secure Boot policy. Figure illustrates the relationship between PK, KEK, db, and dbx.

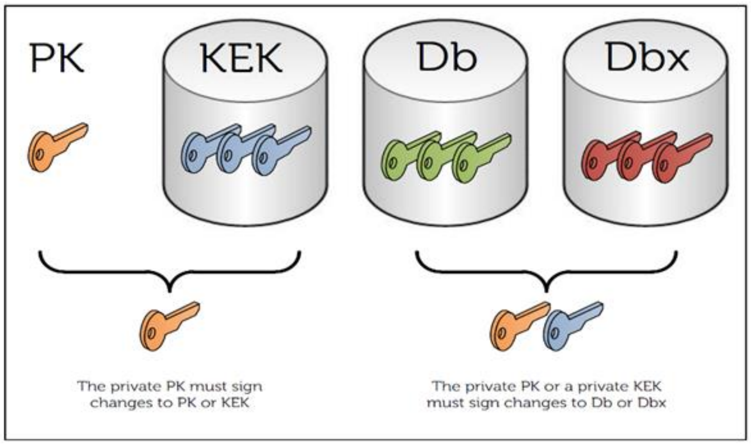

PK
KEK
Db
Dbx
The private PK must sign changes to PK or KEK
The private PK or a private KEK must sign changes to Db or Dbx

Figure 1 Relationship between PK, KEK, DB, and DBX

The Secure Boot policy contains only one key in PK, but multiple keys may reside in KEK. Ideally, either the platform manufacturer or platform owner maintains the private key corresponding to the public PK. Third parties (such as OS providers and device providers) maintain the private keys corresponding to the public keys in KEK. In this way, platform owners or third parties may add or remove entries in db or dbx.

Observe that the owner of a private KEK possesses similar authority as the owner of a private PK. Similar to the private PK owner, owners of private KEKs can authorize or prevent module execution by updating db and dbx. The private PK owner possesses slightly more authority—they can modify the contents of KEK or PK.

In summary, the Secure Boot policy uses db and dbx to authorize pre-boot image file execution. For an image file to get executed, it must associate with a key or hash value in db, and not associate with a key or hash value in dbx. Any attempts to update the contents of db or dbx must be signed by a private PK or KEK. Any attempts to update the contents of PK or KEK must be signed by a private PK.

1.1.1 Acceptable file formats

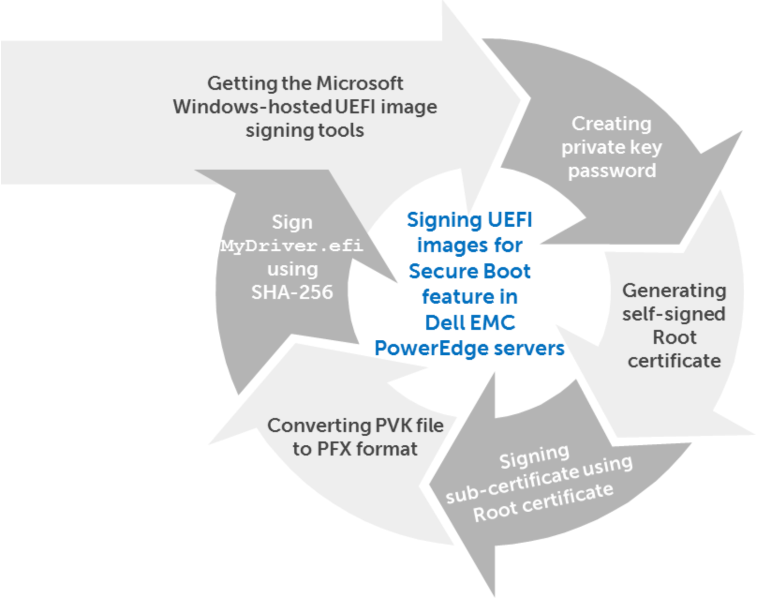
Policy Component	Acceptable File Formats	Acceptable File Extensions	Max records allowed

PK	X.509 Certificate (binary DER format only)	`.cer`
		`.der`
		`.crt`
			One

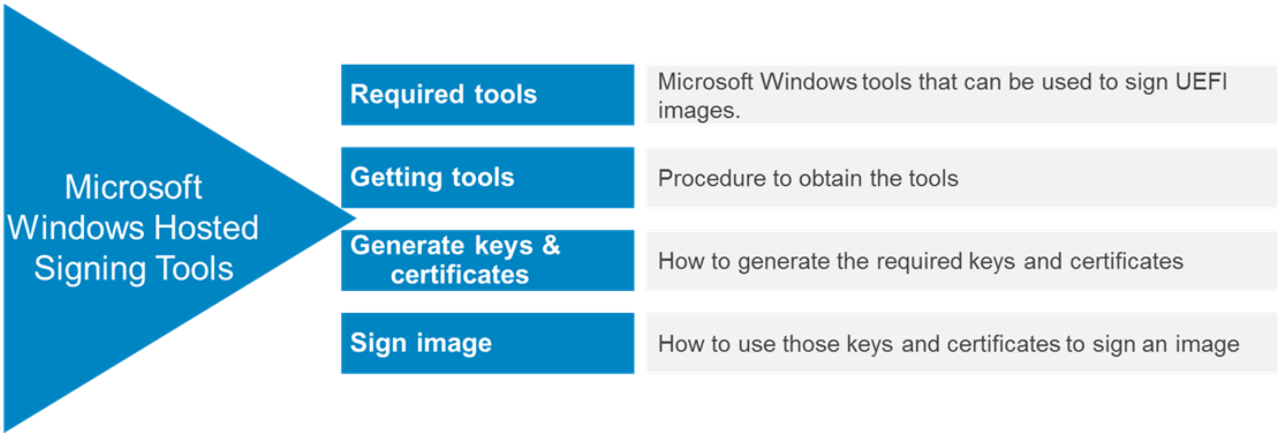
KEK	X.509 Certificate (binary DER format only)
	Public Key Store
		`.cer`
		`.der`
		`.crt`
		`.pbk`
			More than one

DB and DBX	X.509 Certificate (binary DER format only)
	EFI image (system BIOS will calculate and import image digest)
		`.cer`
		`.der`
		`.crt`
		`.efi`
			More than one

1.2 Process of Signing UEFI images for Secure Boot feature in Dell EMC PowerEdge servers

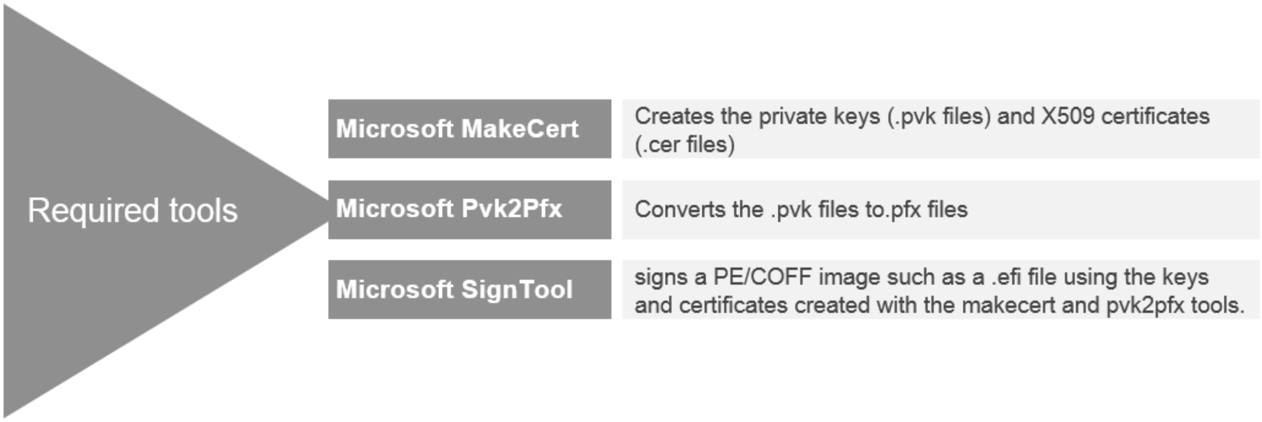

Signing UEFI images for Secure Boot feature in Dell EMC PowerEdge servers
Getting the Microsoft Windows-hosted UEFI image signing tools
Creating private key password
Generating self-signed Root certificate
Signing sub-certificate using Root certificate
Converting PVK file to PFX format
Sign MyDriver.efi using SHA-256

1.2.1 Microsoft Windows Hosted Signing Tools



Required tools	Microsoft Windows tools that can be used to sign UEFI images.
Getting tools	Procedure to obtain the tools
Generate keys & certificates	How to generate the required keys and certificates
Sign image	How to use those keys and certificates to sign an image

1.2.2 Required Tools



Microsoft MakeCert	Creates the private keys (.pvk files) and X509 certificates (.cer files)
Microsoft Pvk2Pfx	Converts the .pvk files to.pfx files
Microsoft SignTool	signs a PE/COFF image such as a .efi file using the keys and certificates created with the makecert and pvk2pfx tools.

1.2.3 Getting the tools

Currently, the most current version of these three tools are all provided in the Windows 8 Consumer Preview SDK available at: http://msdn.microsoft.com/en-us/windows/hardware/hh852363.

1.2.4 Acceptable formats of certificate image file

- pvk is a Microsoft private key file format
- cer is a X509 certificate format using ASN.1 DER encoding
- pfx format is defined by the PKCS#12 standard (http://www.rsa.com/rsalabs/node.asp?id=2138)

Note: These tools must be run from their location in the SDK because they require additional DLL and manifest files from that location. That is, the .EXEs must not be copied to and run from any another location.

1.2.5 Files Used in this Document

The scenario used in this technical white paper references the following files:

- Driver being signed: **MyDriver.efi**
- Root Certificate: Use for development and test only
- **TestRoot.cer**: Root certificate in X509 format
- **TestRoot.pvk**: The Root Certificate's private key in Microsoft PVK format. Has a password.
- **TestRoot.pfx**: Root Certificate's private key in PKCS#12 format.
- Sub-Certificate: Signed by the root certificate. Used to sign the driver. Will be enrolled in the KEK variable. • **TestSub.cer** – Sub-certificate in X509 format. Will be enrolled as KEK.
- **TestSub.pvk**: Sub-Certificate's private key in Microsoft PVK format. Has a password.
- **TestSub.pfx**: Sub-Certificate's private key in PKCS#12 format.

2 Creating Keys and Certificates for Development and Test

This section describes about creating the keys and certificates required to sign a Portable Executable (PE) or Common Object File Format (COFF) image for development and test purposes by using the Microsoft SignTool. It may also be possible to sign development and test images by using other signing tools or a Certificate Authority (CA) provided by an OSV or trusted third party.

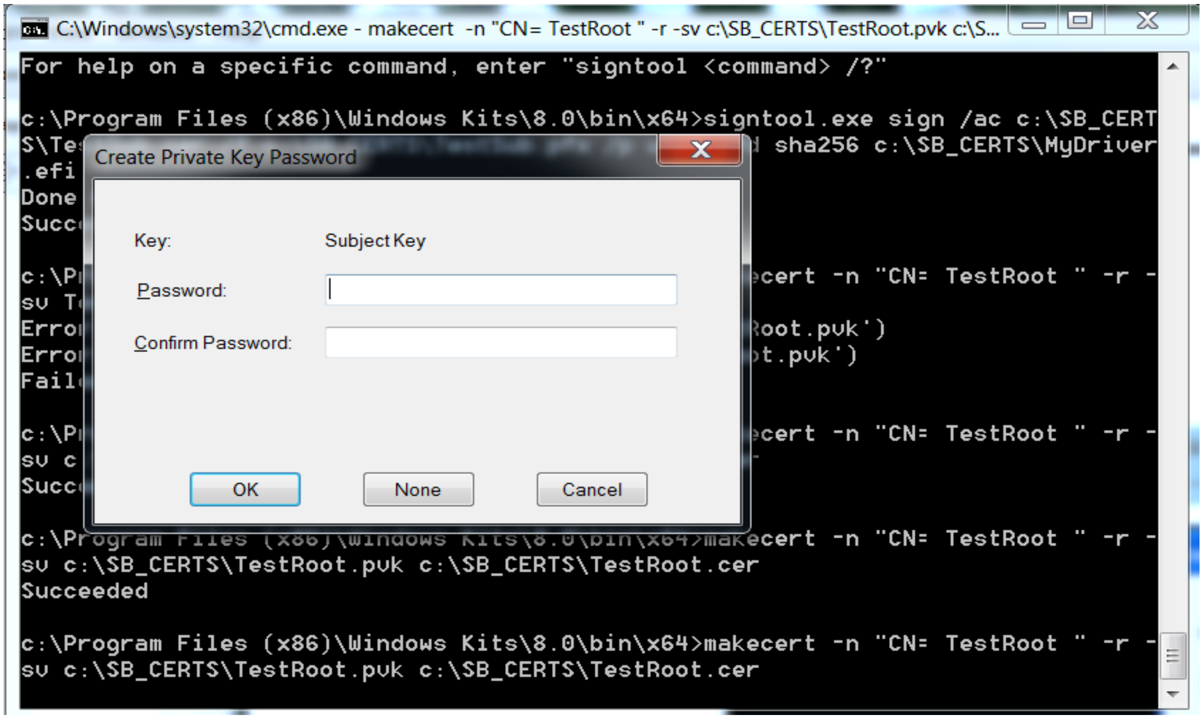
2.1 Generating a self-signed Root certificate

The certificate is used as the Root Certificate and is named as `TestRoot.cer`. It is untrusted, and intended only for development and test use. Its related private key file is `TestRoot.pvk`.

1. Enter the following at the command line interface (CLI):

```
> "C:\Program Files (x86)\Windows Kits\8.0\bin\x64"\ makecert -n "CN= TestRoot " -r
-sv c:\SB_CERTS\TestRoot.pvk c:\SB_CERTS\TestRoot.cer
```

2. In the **Create Private Key Password** dialog box, type the private key password, and then confirm. Keep the password safe for future use.



C:\Windows\system32\cmd.exe - makecert -n "CN= TestRoot " -r -sv c:\SB_CERTS\TestRoot.pvk c:\S...

```
For help on a specific command, enter "signtool <command> /?"

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>signtool.exe sign /ac c:\SB_CERT
S\Te                                          sha256 c:\SB_CERTS\MyDriver
.efi
Done
Succ

c:\Pr                                         cert -n "CN= TestRoot " -r -
sv T
Erro                                          oot.pvk')
Erro                                          t.pvk')
Fail

c:\Pr                                         cert -n "CN= TestRoot " -r -
sv c
Succ

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>makecert -n "CN= TestRoot " -r -
sv c:\SB_CERTS\TestRoot.pvk c:\SB_CERTS\TestRoot.cer
Succeeded

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>makecert -n "CN= TestRoot " -r -
sv c:\SB_CERTS\TestRoot.pvk c:\SB_CERTS\TestRoot.cer
```

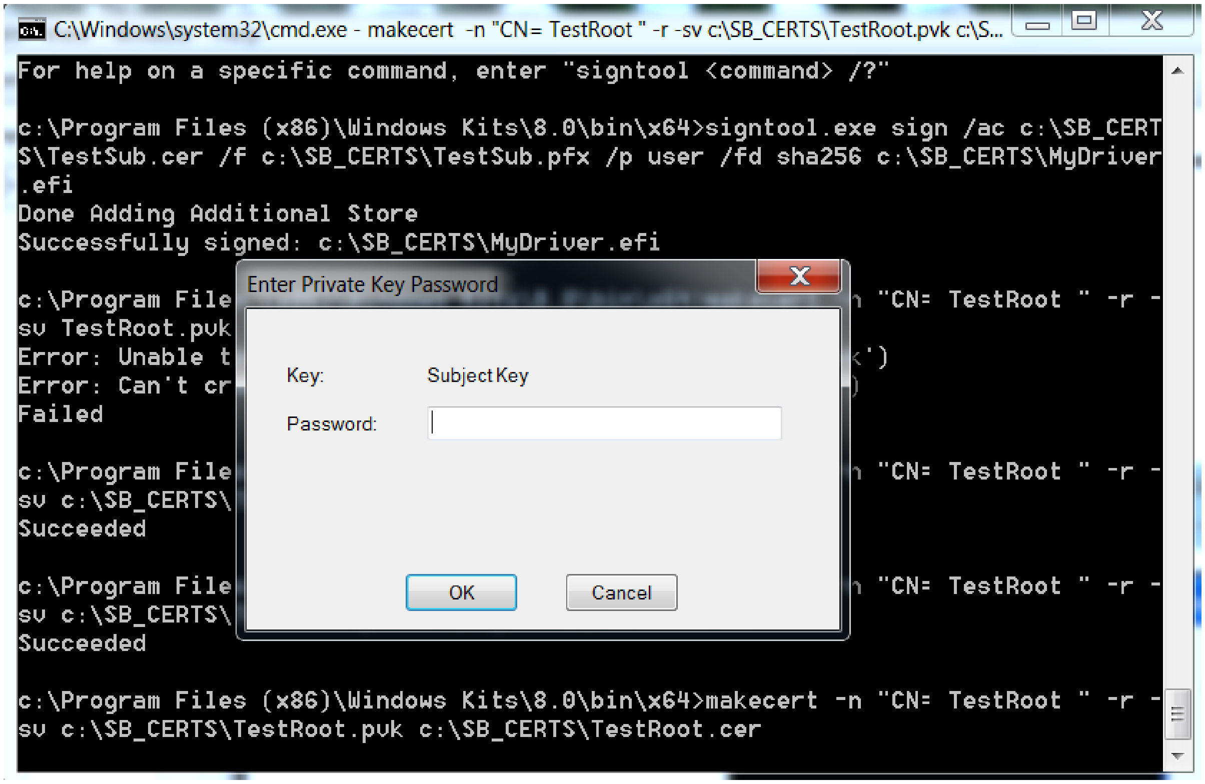
Create Private Key Password

Key: Subject Key

Password:

Confirm Password:

Figure 2 Creating private key password



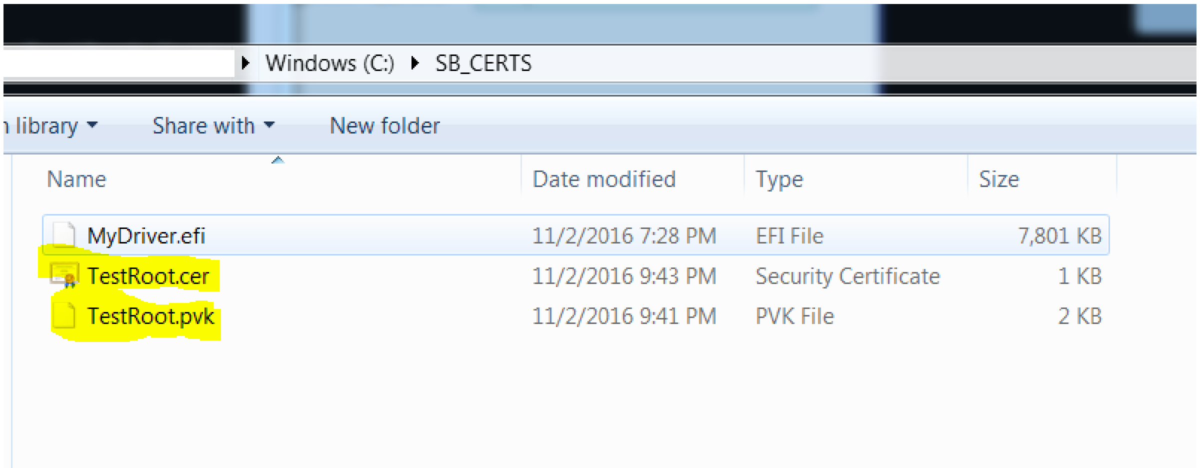
C:\Windows\system32\cmd.exe - makecert -n "CN= TestRoot " -r -sv c:\SB_CERTS\TestRoot.pvk c:\S...
Enter Private Key Password
OK
Cancel

```
For help on a specific command, enter "signtool <command> /?"

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>signtool.exe sign /ac c:\SB_CERT
S\TestSub.cer /f c:\SB_CERTS\TestSub.pfx /p user /fd sha256 c:\SB_CERTS\MyDriver
.efi
Done Adding Additional Store
Successfully signed: c:\SB_CERTS\MyDriver.efi

c:\Program File                                           n "CN= TestRoot " -r -
sv TestRoot.pvk
Error: Unable t                                           <')
Error: Can't cr                                           )
Failed

c:\Program File                                           n "CN= TestRoot " -r -
sv c:\SB_CERTS\
Succeeded

c:\Program File                                           n "CN= TestRoot " -r -
sv c:\SB_CERTS\
Succeeded

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>makecert -n "CN= TestRoot " -r -
sv c:\SB_CERTS\TestRoot.pvk c:\SB_CERTS\TestRoot.cer
```

Key: Subject Key

Password:

Figure 3 Enter configured Root Certificate Private Key Password



Name	Date modified	Type	Size
MyDriver.efi	11/2/2016 7:28 PM	EFI File	7,801 KB
TestRoot.cer	11/2/2016 9:43 PM	Security Certificate	1 KB
TestRoot.pvk	11/2/2016 9:41 PM	PVK File	2 KB

Figure 4 Self-signed Root certificate successfully generated

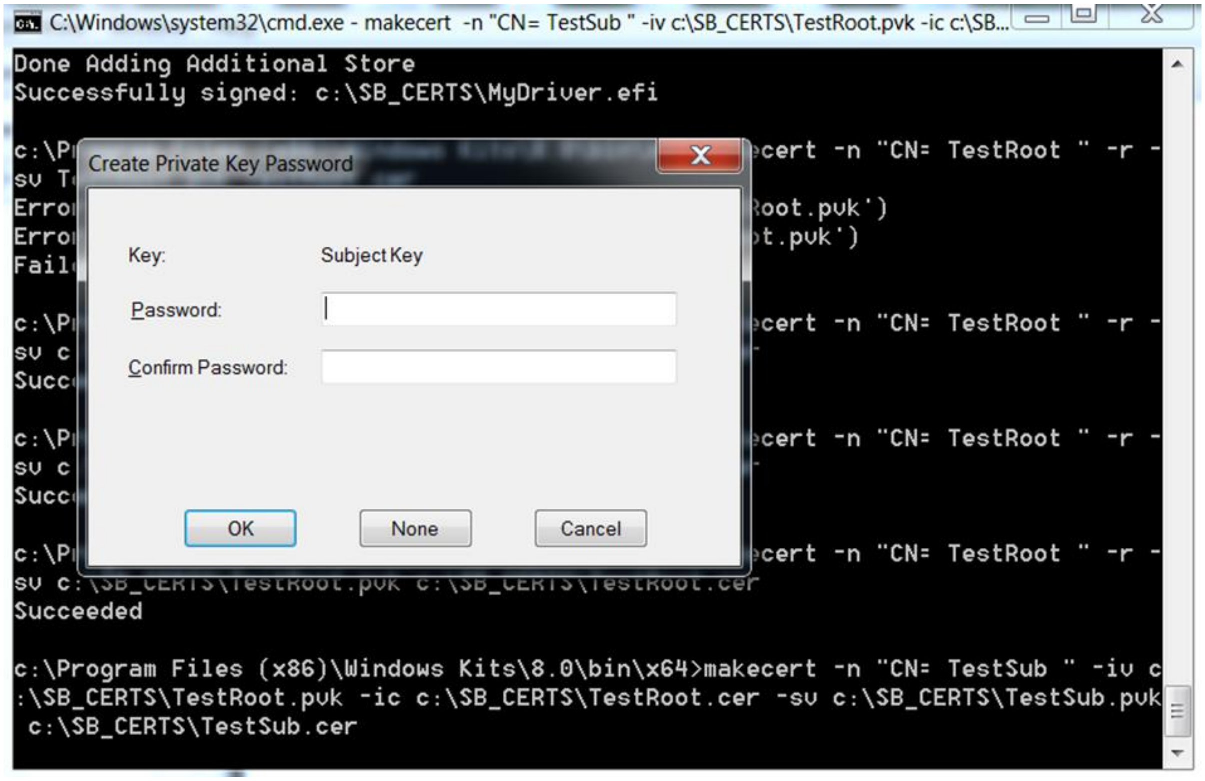
The self-signed Root certificate is generated and listed in the certificate file list. See the sample screen shot here.

2.2 Signing sub-certificate by using the Root certificate

1. Enter the following at the command line interface (CLI):

```
> "C:\Program Files (x86)\Windows Kits\8.0\bin\x64"\ makecert -n "CN= TestSub " -iv
c:\SB_CERTS\TestRoot.pvk -ic c:\SB_CERTS\TestRoot.cer -sv c:\SB_CERTS\TestSub.pvk
c:\SB_CERTS\TestSub.cer
```

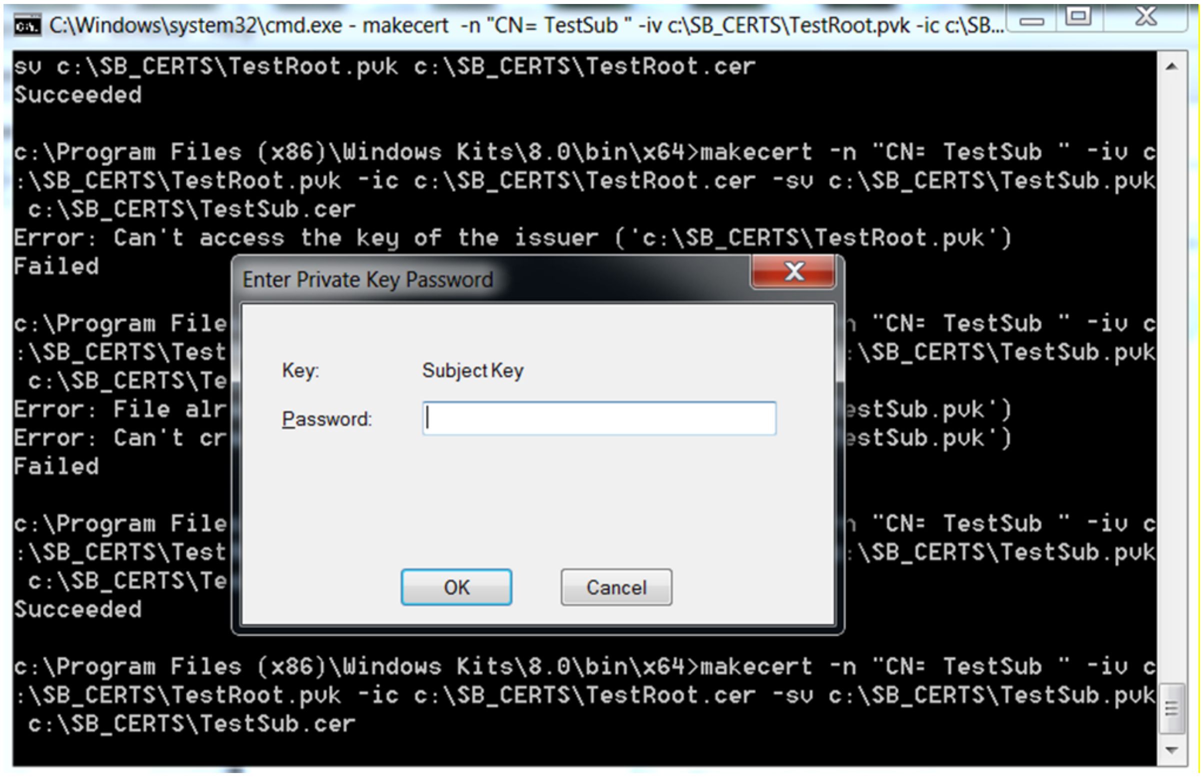
2. In the **Create Private Key Password** dialog box, type the sub-certificate's password, and then confirm. Keep the password safe for future use.


C:\Windows\system32\cmd.exe - makecert -n "CN= TestSub " -iv c:\SB_CERTS\TestRoot.pvk -ic c:\SB...
Create Private Key Password
Password:
Confirm Password:
OK
None
Cancel

Key: Subject Key

```
Done Adding Additional Store
Successfully signed: c:\SB_CERTS\MyDriver.efi

c:\P                                    ecert -n "CN= TestRoot " -r -
sv T
Erro                                    Root.pvk')
Erro                                    t.pvk')
Fail

c:\P                                    ecert -n "CN= TestRoot " -r -
sv c
Succ

c:\P                                    ecert -n "CN= TestRoot " -r -
sv c
Succ

c:\P                                    ecert -n "CN= TestRoot " -r -
sv c:\SB_CERTS\TestRoot.pvk c:\SB_CERTS\TestRoot.cer
Succeeded

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>makecert -n "CN= TestSub " -iv c
:\SB_CERTS\TestRoot.pvk -ic c:\SB_CERTS\TestRoot.cer -sv c:\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\TestSub.cer
```

Figure 5 Create Private key Password for Sub-Certificate



C:\Windows\system32\cmd.exe - makecert -n "CN= TestSub " -iv c:\SB_CERTS\TestRoot.pvk -ic c:\SB...

```
sv c:\SB_CERTS\TestRoot.pvk c:\SB_CERTS\TestRoot.cer
Succeeded

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>makecert -n "CN= TestSub " -iv c
:\SB_CERTS\TestRoot.pvk -ic c:\SB_CERTS\TestRoot.cer -sv c:\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\TestSub.cer
Error: Can't access the key of the issuer ('c:\SB_CERTS\TestRoot.pvk')
Failed

c:\Program File                                 n "CN= TestSub " -iv c
:\SB_CERTS\Test                                 :\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\Te
Error: File alr                                 estSub.pvk')
Error: Can't cr                                 estSub.pvk')
Failed

c:\Program File                                 n "CN= TestSub " -iv c
:\SB_CERTS\Test                                 :\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\Te
Succeeded

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>makecert -n "CN= TestSub " -iv c
:\SB_CERTS\TestRoot.pvk -ic c:\SB_CERTS\TestRoot.cer -sv c:\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\TestSub.cer
```

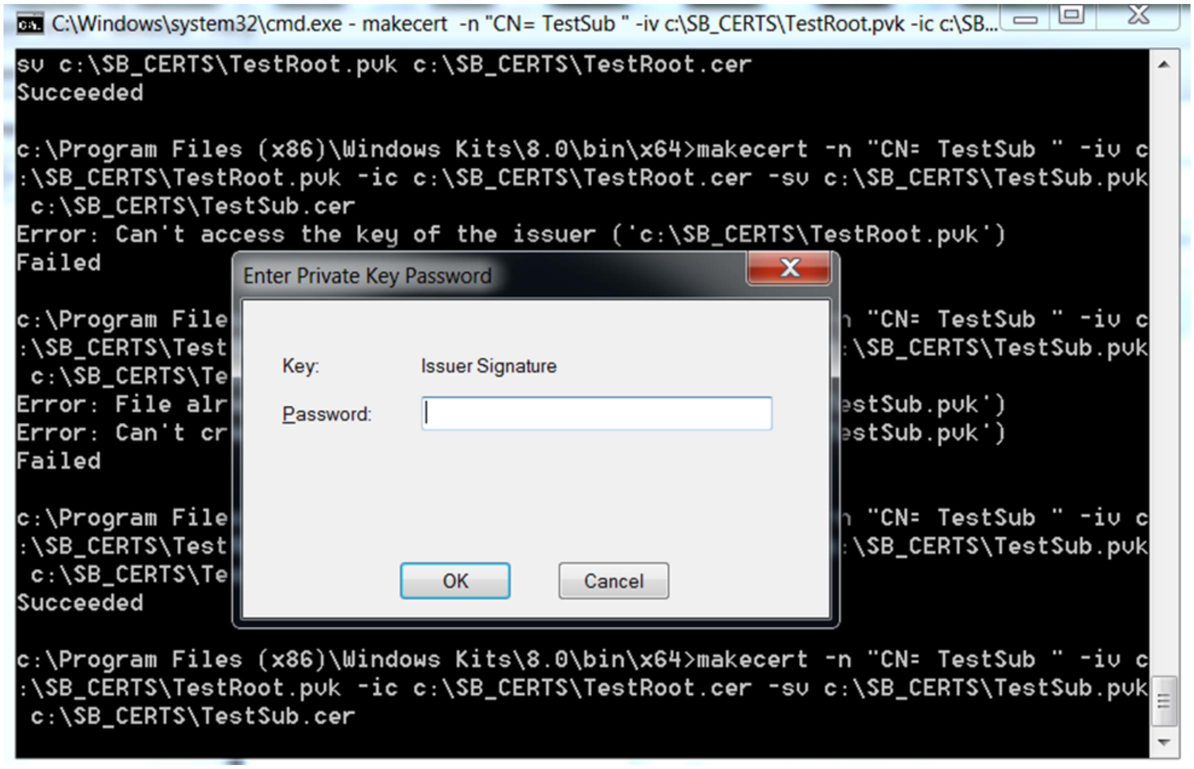
Enter Private Key Password

Key: Subject Key

Password:

Figure 6 Enter configured Sub-Certificate private key password

3. Enter the Passwords of Root certificate with which sub-certificate is signed.



C:\Windows\system32\cmd.exe - makecert -n "CN= TestSub " -iv c:\SB_CERTS\TestRoot.pvk -ic c:\SB...

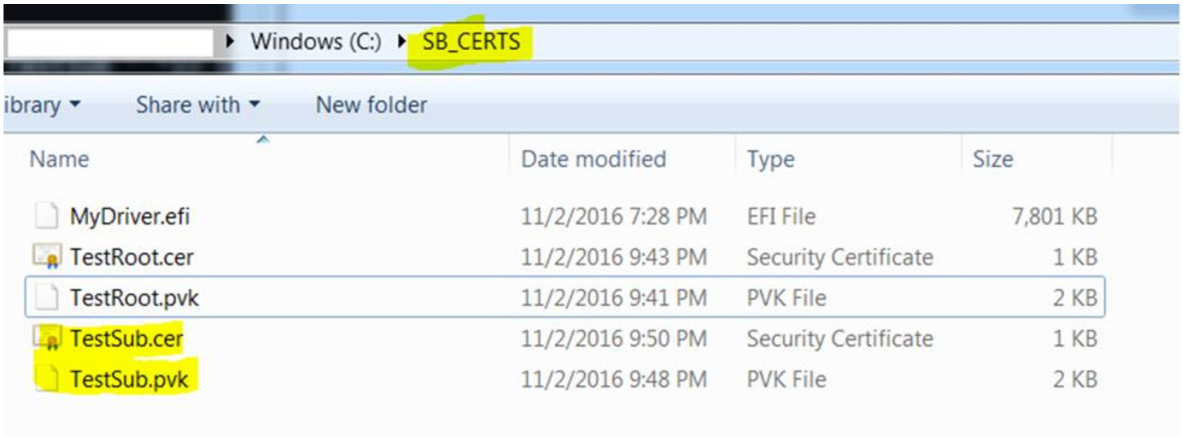
```
sv c:\SB_CERTS\TestRoot.pvk c:\SB_CERTS\TestRoot.cer
Succeeded

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>makecert -n "CN= TestSub " -iv c
:\SB_CERTS\TestRoot.pvk -ic c:\SB_CERTS\TestRoot.cer -sv c:\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\TestSub.cer
Error: Can't access the key of the issuer ('c:\SB_CERTS\TestRoot.pvk')
Failed

c:\Program File                                        n "CN= TestSub " -iv c
:\SB_CERTS\Test                                        :\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\Te
Error: File alr                                        estSub.pvk')
Error: Can't cr                                        estSub.pvk')
Failed

c:\Program File                                        n "CN= TestSub " -iv c
:\SB_CERTS\Test                                        :\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\Te
Succeeded

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>makecert -n "CN= TestSub " -iv c
:\SB_CERTS\TestRoot.pvk -ic c:\SB_CERTS\TestRoot.cer -sv c:\SB_CERTS\TestSub.pvk
 c:\SB_CERTS\TestSub.cer
```

Enter Private Key Password

Key: Issuer Signature

Password:

Figure 7 Enter configured Root Certificate private key password



Windows (C:) ▸ SB_CERTS

Name	Date modified	Type	Size
MyDriver.efi	11/2/2016 7:28 PM	EFI File	7,801 KB
TestRoot.cer	11/2/2016 9:43 PM	Security Certificate	1 KB
TestRoot.pvk	11/2/2016 9:41 PM	PVK File	2 KB
TestSub.cer	11/2/2016 9:50 PM	Security Certificate	1 KB
TestSub.pvk	11/2/2016 9:48 PM	PVK File	2 KB

Figure 8 Sub Certificates generated successfully

The sub-certificate is successfully signed and the .pvk file is listed in the certificate file list. See the sample screen shot here.

3 Signing a UEFI Image

To sign a UEFI Secure Boot image:

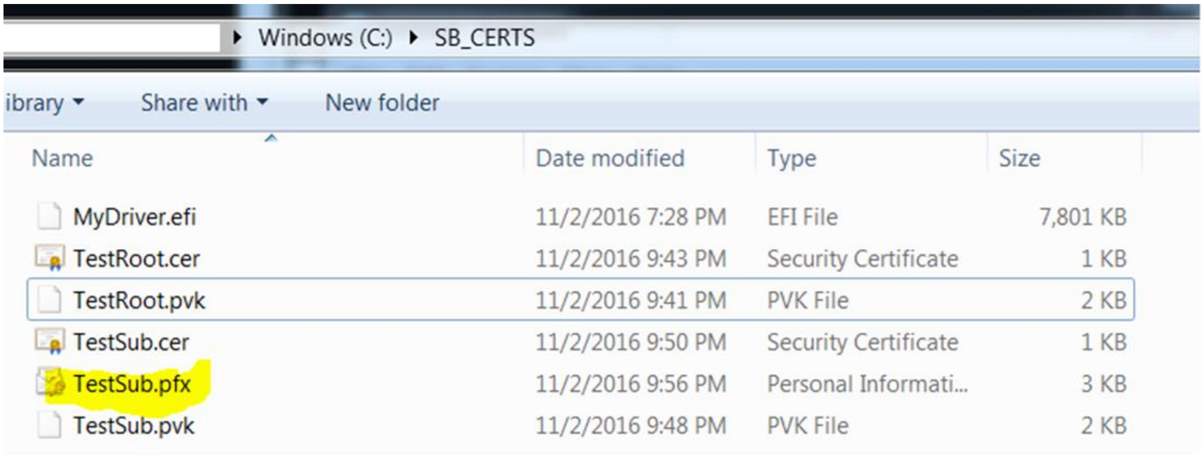
- Convert the .pvk file to .pfx format
- Sign MyDriver.efi by using SHA-256

3.1 Converting the PVK file to PFX format (PKCS#12)

The command line under DOS environment is (xxx stands for SubIssuer.pvk password):

1. At the CLI, enter:

```
> "C:\Program Files (x86)\Windows Kits\8.0\bin\x64"\ pvk2pfx.exe -pvk
c:\SB_CERTS\TestSub.pvk -pi xxx -spc c:\SB_CERTS\TestSub.cer -pfx
c:\SB_CERTS\TestSub.pfx -f
```



Name	Date modified	Type	Size
MyDriver.efi	11/2/2016 7:28 PM	EFI File	7,801 KB
TestRoot.cer	11/2/2016 9:43 PM	Security Certificate	1 KB
TestRoot.pvk	11/2/2016 9:41 PM	PVK File	2 KB
TestSub.cer	11/2/2016 9:50 PM	Security Certificate	1 KB
TestSub.pfx	11/2/2016 9:56 PM	Personal Informati...	3 KB
TestSub.pvk	11/2/2016 9:48 PM	PVK File	2 KB

Figure 9 The TextSub.pvk Is successfully converted to TextSub.pfx

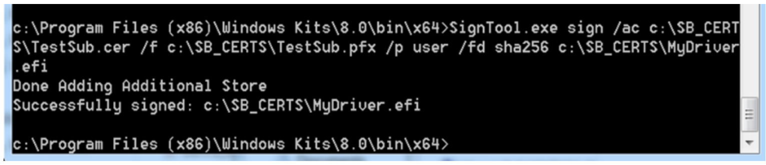
2. The .pvk file is converted to .pfx file and listed in the certificate file list. See the sample screen shot here.

3.2 Sign MyDriver.efi using SHA-256

1. At the CLI, enter:

```
SignTool.exe sign /ac c:\SB_CERTS\TestSub.cer /f c:\SB_CERTS\TestSub.pfx /p xxx /fd sha256
c:\SB_CERTS\MyDriver.efi
```

Note: xxx indicates the TestSub.pvk password.



```
c:\Program Files (x86)\Windows Kits\8.0\bin\x64>SignTool.exe sign /ac c:\SB_CERT
S\TestSub.cer /f c:\SB_CERTS\TestSub.pfx /p user /fd sha256 c:\SB_CERTS\MyDriver
.efi
Done Adding Additional Store
Successfully signed: c:\SB_CERTS\MyDriver.efi

c:\Program Files (x86)\Windows Kits\8.0\bin\x64>
```

Figure 10 Signing MyDriver.efi

Conclusion

Secure Boot is UEFI standard which removes the Legacy Threat and provides software identity checking at every step of boot process: Platform Firmware, Option Cards, and OS Boot loader. UEFI Secure Boot is a technology to eliminate a major security void during handoff from UEFI firmware to UEFI OS. The concept of UEFI secure boot is to have each component in the chain validated and authorized against a given policy before allowing it to execute.

This technical white paper is intended for firmware architects, engineers, and IT administrators who want to explore the advantages of using Secure Boot feature on Dell EMC PowerEdge servers and who need to design and plan implementations by using it. This whitepaper describes the process of generating certificates and signing UEFI images by using those certificates for development and test purposes.