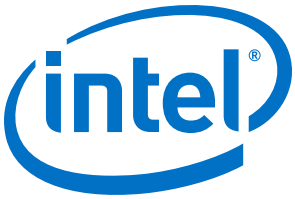


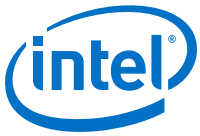
Intel® IoT Gateways Based on Intel® Atom™ Processors: Secure Boot

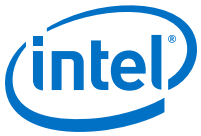
Implementation Guide

November 2015

Intel Confidential

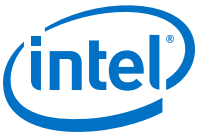
Doc. No.: 562543, Rev.: 1.0



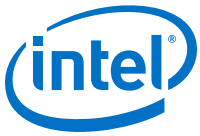


Revision History

Date	Revision	Description
November 2015	1.0	Initial release



Contents



1.0 Introduction

Applicable to This Intel® IoT Gateway Product

- Intel® IoT Gateways based on Intel® Atom™ processors, 100 gigabyte models.

How These Instructions Help You

Without the protection provided by Secure Boot, unsigned and possibly malevolent boot software can be placed onto your Gateway. Secure Boot provides a secure environment during the boot process, keeping malevolent boot software away. This guide walks you through enabling the Secure Boot features on your Gateway.

Part of the configuration steps in this guide must be completed by an original design manufacturer (ODM) and other parts can be completed by an end user:

Configuration Step Order	Boot Verification Stage	Completed By
First	Stage 2 Verification	ODM or end user
Second	Stage 1 Verification	ODM or end user
Third	Stage 3 Verification	End user

Document Terminology and Conventions

- **Terminology**
  - Gateway: An Intel® IoT Gateway.
  - Development Computer: Linux computer that you provide to develop applications for your Gateway. Ubuntu* 14.04 Base is preferred.
- **Conventions**
  - `This font is used for commands, API names, parameters, filenames, directory paths, and executables.`
  - **Bold text** is used for graphical user interface entries, buttons, and keyboard keys.

```
This font in a gray box is used for commands you must type or include in a script.
```

Required Experience Level

This guide assumes you have experience with:

- Executing Linux commands.
- Creating, editing, and executing scripts.
- Installing and configuring Linux and Windows software.
- Using a terminal emulation program like PuTTY* with serial connections between computers.

Tool Directories and Software

You will set up the following directories and populate them with software as part of your Secure Boot configuration. You will need to have an Intel CNDA (legal agreement) in place before you can access the tools. Contact your ODM for information.

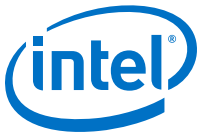
Directories you will create:

Directory	Purpose
F2a_BIOS	The tools in this directory are used to flash the BIOS from the UEFI shell. You will use these tools after the Secure Boot BIOS is complete.
Flash_Manifest_GBT	Key directory to create the Secure Boot BIOS. This directory contains tools to divide the BIOS, sign portions, hash portions, and reassemble the BIOS.
IVI_1113	Used at the end of the process. The tools in this directory create a create a final 8 MB BIOS.

Software tools you will use:

Software Tool	Directory Location	Description
BayTrail_I_NoEMMC.xml	IVI_1113	Configuration file for FITC.exe
BiosImageInfo.bin		OEM data region
CryptoCon.exe	Flash_Manifest_GBT	Hashes UEFI stages
FD_NON_PRE_BB.fd		UEFI Stage 2
FD_PRE_BB_127K.fd		IBB (127KB IBB)
FITC.exe	IVI_1113	Combines descriptor, BIOS, PDR, and Intel® TXE FW binaries into one image.
FLAMInGo.exe	Flash_Manifest_GBT	Hashing and secure boot manifest creation.
fparts.txt	F2a_BIOS	Configuration for `pt.efi` or `fpt64.efi`
fpt.efi or fpt64.efi	F2a_BIOS	Flash programming tool that programs the flash memory of individual regions or the entire flash device. One-time programming of Intel® TXE fuses of the Gateway processor.
FusesConfiguration.bat	Flash_Manifest_GBT	Sets up the manifest generation tool environment.
GenFW.exe	Flash_Manifest_GBT	
insert.exe	Flash_Manifest_GBT	
link.exe	Flash_Manifest_GBT	
ml.exe	Flash_Manifest_GBT	Microsoft* Linker
oFpfMirrorNvarValues.txt	Flash_Manifest_GBT	Used to generate optional FPT mirror values
OpenSSL		Signs binaries, creates key pairs, wraps private keys.
ORG.ROM		The 5 MB BIOS ROM that originates from the BIOS vendor
Production_VLV_SEC_REGION.bin	IVI_1113	Intel® TXE part of the BIOS

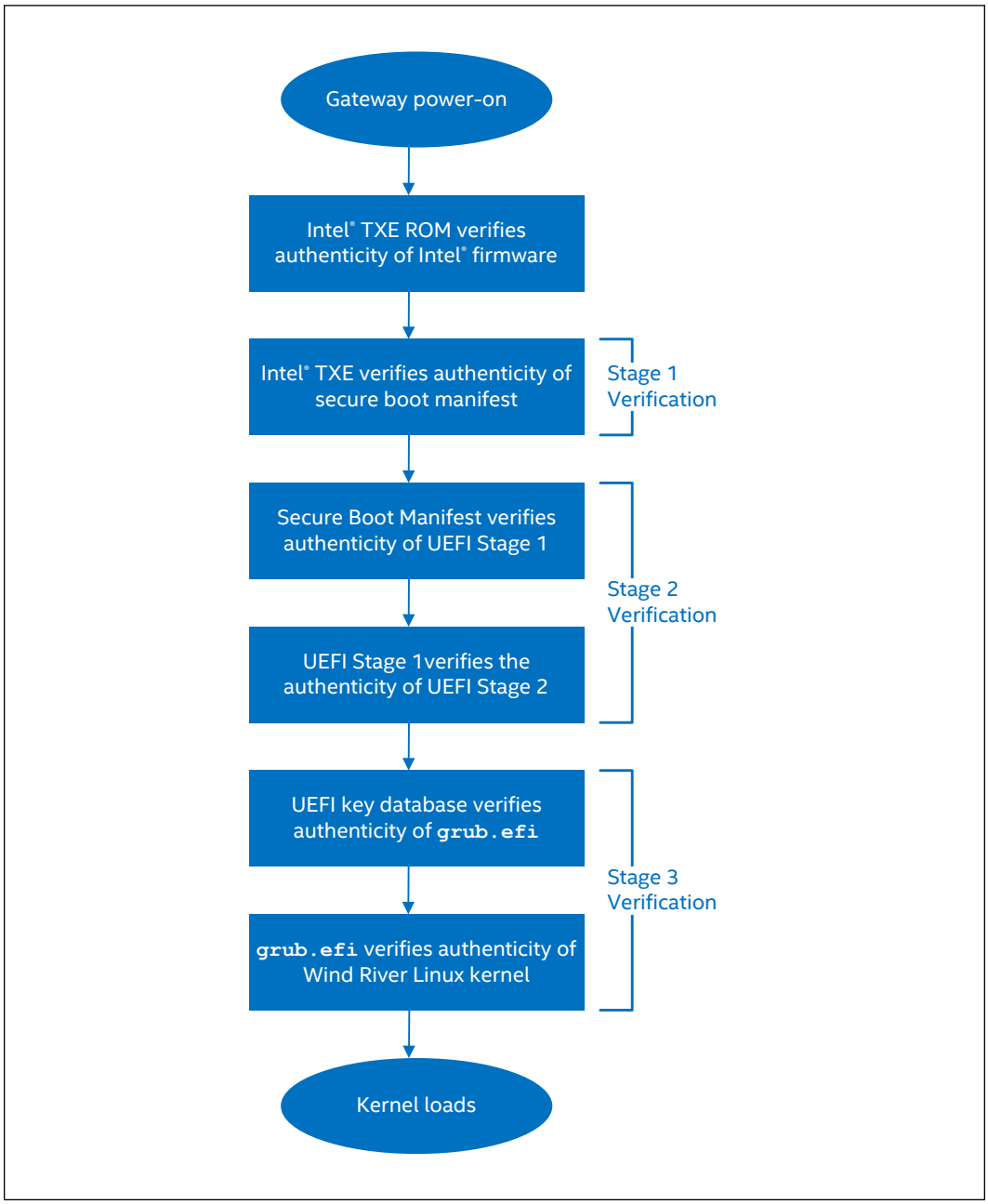
continued...



Software Tool	Directory Location	Description
SimpleSigner.exe	Flash_Manifest_GBT	Signs an input file with a private key and provides an output signature. An ODM will need to develop their own tool.
split.exe	Flash_Manifest_GBT	Splits binary files. Separates UEFI Stages
TXE_Region_3MB.7z		The 3 MB Intel® TXE firmware

2.0 About Secure Boot

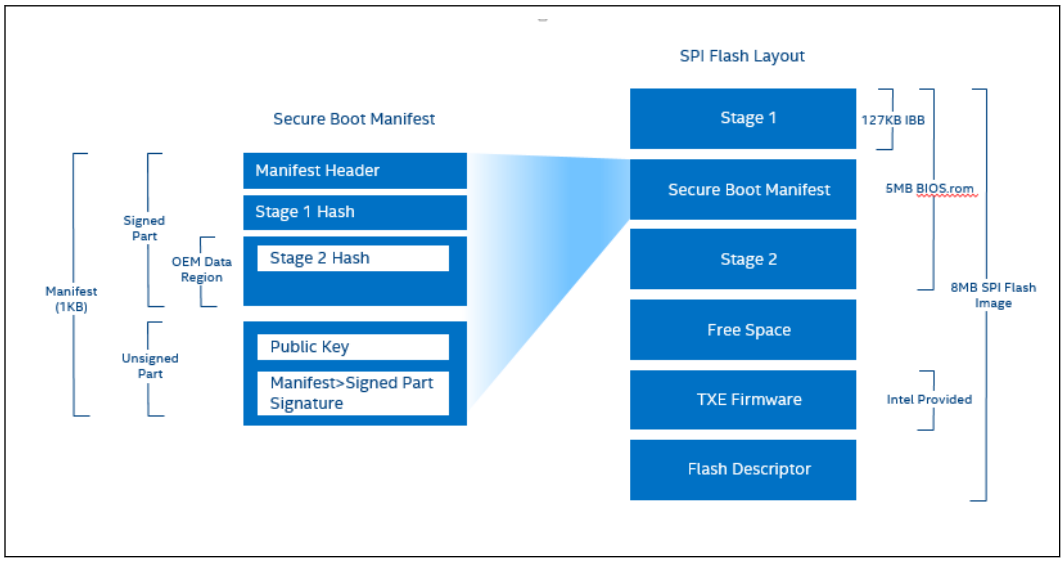
The figure below shows the high-level Secure Boot flow.



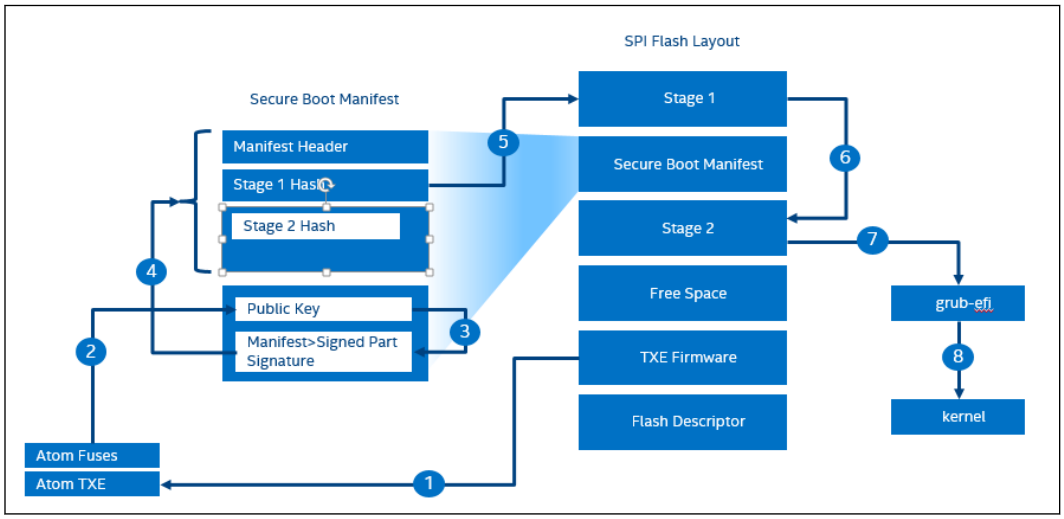
SPI flash a key part of Secure Boot. The BIOS binary in SPI flash contains:

- BIOS Stage 1 (Initial Boot Block)
- BIOS Stage 2
- Secure Boot Manifest
- Intel® Trusted Execution Engine (Intel® TXE) firmware

See the following SPI flash region definition:



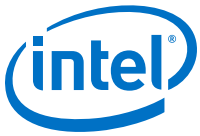
This is the detailed Secure Boot flow:



Using the figures above as a reference, the Secure Boot flow is:

1. Gateway power-on:
   - CPU is held in reset and does not execute code.
   - Intel® TXE ROM code verifies the authenticity of Intel® TXE firmware.
   - Intel® TXE in the processor fetches and executes the Intel® TXE firmware resident on the SPI flash.
2. Intel® TXE firmware:
   - Reads the internal processor fuses.
   - Performs a SHA-256 hash of the RSA public key stored in the unsigned region of the Secure Boot manifest.
   - Compares the calculated hash value against the value stored in the processor fuses.
   - If no match is found, execution halts.
3. The signature of the signed part of the Secure Boot manifest is stored in the Manifest's Unsigned Part. This signature is decrypted with the verified RSA public key. The decryption result is the SHA-256 hash of the Manifest's Signed Part.
4. Intel® TXE firmware calculates the SHA-256 hash of the Manifest's Signed Part and compares the result with the decrypted hash. If no match, execution halts.

   Note: Since the IBB and UEFI Stage 2 hashes are included in the Manifest's Signed Part, successful verification of this step guarantees the IBB and BIOS Stage 2 hashes are authentic and un-altered.
5. Calculation of SHA-256 hash of the Initial Boot Block:
   - Intel® TXE firmware calculates the hash.
   - Calculated hash is compared to the authenticated IBB hash in the Manifest. If the hashes are identical, then the IBB is authenticated. The power management controller releases the CPU from reset.
   - The CPU fetches and executes the IBB, which performs basic initialization of the platform, including memory initialization.
6. Calculation of SHA-256 hash of the UEFI Stage 2 that was authenticated in Step 4.
   - IBB code calculates the hash.
   - IBB code compares the result against the UEFI Stage 2 hash.
   - If the hashes are not identical, execution halts.
7. UEFI Stage 2 calculates the hash of `grub.efi` and compares it to the keys that are enrolled in the UEFI key database. If the hashes are not identical, execution halts.
8. `grub.efi` decrypts the signature appended to the Linux kernel and verifies the integrity. If verification passes, the kernel loads and executes. Otherwise, execution halts.



3.0 Prepare Microsoft Windows* Computer

This section is for both the ODM and end user.

Perform these steps on a computer running Microsoft Windows. These steps use Microsoft Windows 8. If you are using a different version, you may need to choose different menu selections to reach the indicated screens.

Note: You must have administrative rights on your Microsoft Windows computer to complete these steps.

Install Software

1. End user: Get `SecureBootToolsBayTrail.zip` and the associated password from your ODM.
2. In your `Documents` folder, create a folder named `Security`
3. Within the `Security` folder, create a folder named `SecureBootBT`
4. Copy `SecureBootToolsBayTrail.zip` to the `SecureBootBT` folder.
5. Unzip `SecureBootToolsBayTrail.zip` using the password from your ODM.

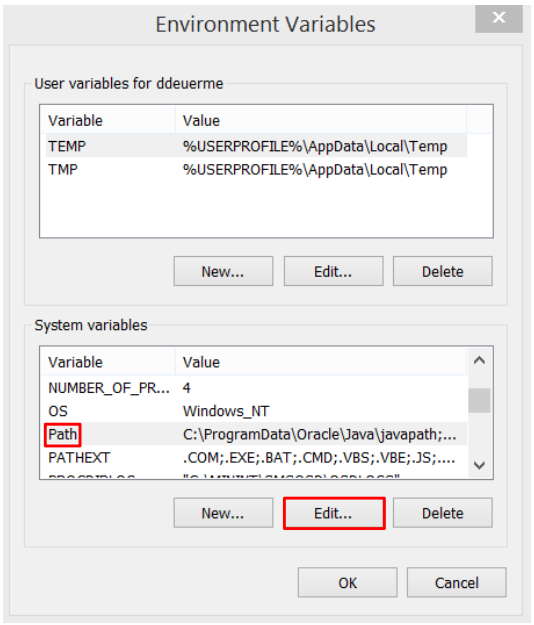
   Three folders are created and populated under `SecureBootBT`:

   - `F2a_BIOS`
   - `Flash_Manifest_GBT`
   - `IVI_1113`

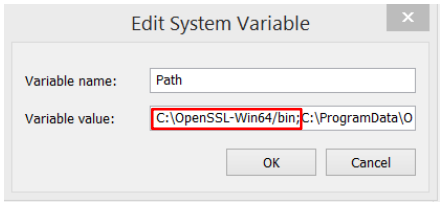
Set Up Win32OpenSSL

1. Go to http://slproweb.com/products/Win32OpenSSL.html.
2. Download and install **Win32_OpenSSL_v1.0.2d** or **Win64_OpenSSL_v1.0.2d**.

   Note: V1.0.2d was the current version as of October 2015. The version numbers may change. Select the latest available version.
3. The installation creates a new folder on your computer. Write down the full path, such as `C:\OpenSSL-Win64`
4. Open the properties settings for your computer: Click your folder icon and then **System Properties** from the menu bar at the top of the screen.
5. Click **Advanced system settings**. If prompted with a message requesting permission to access the screen, click **Yes**.
6. Select the **Advanced** tab.
7. Click **Environment Variables** near the bottom of the window.
8. Under **System variables**, scroll to select **Path** and then click **Edit**:

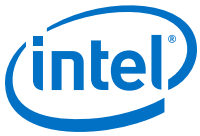


9. In the **Variable value** field, add `C:\OpenSSL-Win64\bin;` to the beginning of the field, where `C:\OpenSSL-Win64` is the full path from step 3:



Note: You can add this path setting elsewhere within the **Variable value** field. Make sure each entry in the line is separated by a semicolon.

Important: Do not remove any exiting content from the **Variable value** field.



4.0 Create the Secure Boot Image

Creating a Secure Boot image entails the following steps:

- ODM steps: Split UEFI stages, produce stage hashes, and create the data region.
- End user or ODM steps:
  - Create a public and private key pair.
  - Create the BIOS with Secure Boot manifest.
  - Create the SPI flash binary.
  - Load the FPF configuration file for FPF mirroring (if necessary).

ODM Note: If you generate the key pair and you are keeping the private key secure, then you can optionally complete this step for your end user(s).

End User Note: If you want to generate your own key pair, then ask your ODM for the files you need to create the secure boot firmware image.

Split UEFI Stages, Produce Stage Hashes, Create Data Region

Only an ODM can complete these steps.

1. Open a command prompt window with administrative rights.
2. Go to the `Flash_Manifest_GBT` directory where <user> is your user name:

```
CD \Users\<user>\Documents\Security\SecureBootBT\Flash_Manifest_GBT
```

3. The BIOS vendor provided a 5 MB file called `ORG.ROM`. Split this into IBB and UEFI Stage 2:

```
split -f ORG.ROM -s 0x4e0000 -o FD_NON_PRE_BB.fd -t FD_PRE_BB.fd
split -f FD_PRE_BB.fd -s 1024 -o FD_PRE_BB_1K.fd -t FD_PRE_BB_127K.fd
```

4. Generate the IBB and UEFI Stage 2 hashes:

```
split -f ORG.ROM -s 0x50000 -o HASHL.fd -t HASHR.fd
split -f HASHR.fd -s 0x40e000 -o STG2.fd
split -f ORG.ROM -s 0x3fe000 -o FvBbD.fd -t left.fd
split -f left.fd -s 0x60000 -o FV_BB.Fv
CryptoCon.exe -h2 -f STG2.fd -o HashSecondStageKey.bin
CryptoCon.exe -h2 -f FV_BB.Fv -o HashFvBbKey.bin
```

5. Generate the OEM data region:

```
insert.exe CreateIncFromBin HashSecondStageKey.inc HashSecondStageKey.bin
insert.exe CreateIncFromBin HashFvBbKey.inc HashFvBbKey.bin
ml /c /nologo /Fo GenBiosImageInfo.obj GenBiosImageInfo.asm
link /NOENTRY /FIXED /DLL GenBiosImageInfo.obj /OUT:GenBiosImageInfo.dll
genfw --exe2bin GenBiosImageInfo.dll -o GenBiosImageInfo.bin
split -f GenBiosImageInfo.bin -s 0x64 -o BiosImageInfo.bin
```

6. If the end user will be creating the secure boot firmware image, provide the following files:

Filename	Description
FD_PRE_BB_127K.fd	IBB (127 KB IBB)
FD_NON_PRE_BB.fd	BIOS Stage 2
BiosImageInfo.bin	OEM Data Region
ORG.ROM	5 MB BIOS ROM from the BIOS vendor

Create a Public and Private Key Pair

Either the end-user or the ODM can complete the steps in this section. An Intel CNDA is required to access the necessary tools. If you are an end user, contact your ODM for information.

1. Open a command prompt with administrative rights.
2. Generate a public and private key pair to sign and verify the Secure Boot Manifest. Modify the `openssl` parameters based on your security policy. Change `c:\OpenSSL-Win64\bin` to the full path to your `openssl.cnf` file.

```
openssl req -batch -x509 -nodes -days 9000 -newkey rsa:2048 -keyout
"privatekey.pem" -out "publickey.pem" -config "c:\OpenSSL-Win64\bin\openssl.cnf"
```

   Two files are placed into your working directory:

   - `privatekey.pem`
   - `publickey.pem`

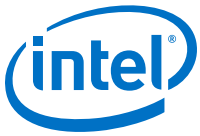
3. Change the encoding of the private key to PKCS#8:

```
openssl pkcs8 -topk8 -in privatekey.pem -out privatekey-pk8.pem -nocrypt
```

4. Rename `privatekey-pk8.pem` to `privatekey.pem`.

```
move privatekey-pk8.pem privatekey.pem
```

   The result is `privatekey.pem` in PKCS#8 encoding.



Create the BIOS With Secure Boot Manifest

Either the end-user or the ODM can complete the steps in this section. An Intel CNDA is required to access the necessary tools. If you are an end user, contact your ODM for information.

1. Open a command prompt with administrative rights.
2. Hash the public key using the FLAMInGo tool from Intel:

```
FLAMInGo.exe HashKey publickey.pem publickeyhash.txt
```

   The result is `publickeyhash.txt` in your working directory.

3. Create the Intel® TXE fuse mirror values that will be programmed into the processor's Field Programmable Fuses:

```
FusesConfiguration.bat publickeyhash.txt oFpfMirrorNvarValues.txt
FpfMirrorNvarValues.txt
```

   The result is `oFpfMirrorNvarValues.txt` in your working directory.

4. Create the manifest structure and the hash of the Manifest Signed Part:

```
FLAMInGo.exe SBManCreate FpfMirrorNvarValues.txt AmiManifest FD_PRE_BB_127K.fd 2
publickey.pem -OEMDataFile BiosImageInfo.bin
```

   The result is two files in your working directory:

   - `AmiManifest_SB_config.xml`
   - `AmiManifest_SB_hash.bin`

5. Sign the manifest's hash with your private key with the tool of your choice. SimpleSigner is an example:

```
SimpleSigner.exe privatekey.pem AmiManifest_SB_hash.bin
AmiManifest_SB_signature.bin
```

   The result is `AmiManifest_SB_signature.bin` in your working directory.

6. Create the Secure Boot Manifest using these inputs:

   - Fuse mirror values: `SBManComplete`
   - Manifest XML structure: `FpfMirrorNvarValues.txt`
   - Manifest signature: `AmiManifest_SB_signature.bin`

```
FLAMInGO.exe SBManComplete FpfMirrorNvarValues.txt AmiManifest
AmiManifest_SB_signature.bin
```

   The result is `AmiManifest_SB_Manifest.bin` in your working directory.

7. Merge BIOS Stage 2 with the Manifest and IBB:

```
copy /Y /B FD_NON_PRE_BB.fd+AmiManifest_SB_manifest.bin AMI.ROM
```

   The result is a 5 MB file called `AMI.ROM` in your working directory.

8. Copy `AMI.ROM` and `FpfMirrorNvarValues.txt to the IVI_1113` directory:

```
copy AMI.ROM ..\IVI_1113
copy FpfMirrorNvarValues.txt ..\IVI_1113
```

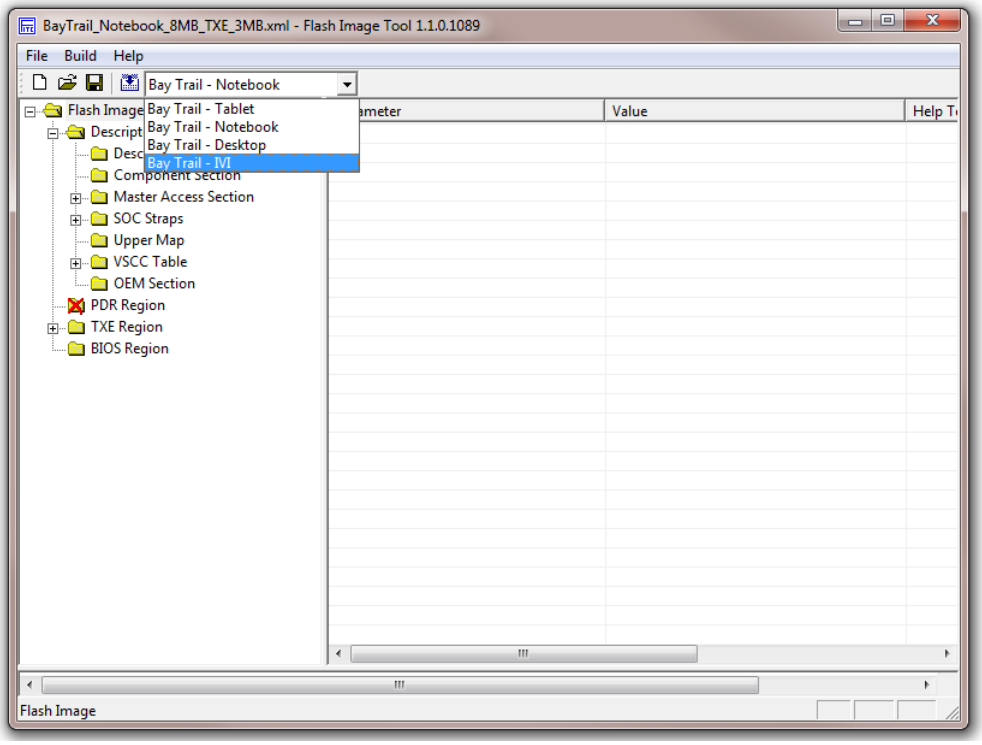
Create the SPI Flash Binary

Either the end-user or the ODM can complete the steps in this section. An Intel CNDA is required to access the necessary tools. Contact your ODM for information.

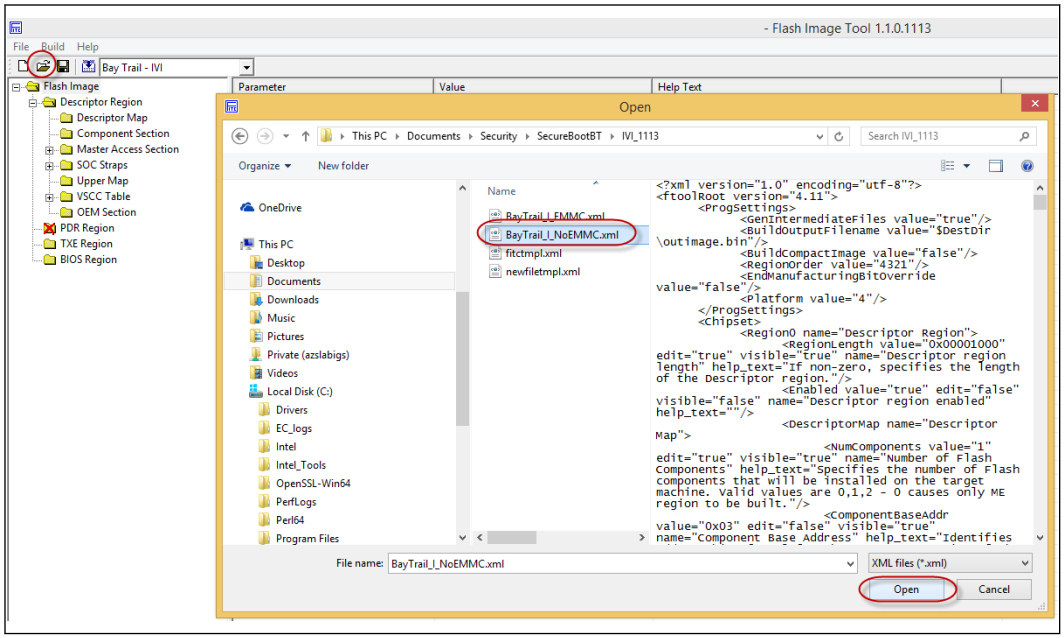
In this section you will merge `AMI.ROM` with the Intel® TXE firmware. This is the binary that will be programmed into the Gateway's SPI flash device.

1. From within your Windows environment (not from a command prompt), go to the `IVI_1113` directory.
2. Double-click `fitc.exe`

   The Flash Image Tool launches. Dismiss any error messages related to a `.xml` file.
3. When the tool opens, select **Baytrail-IVI** from the drop-down:

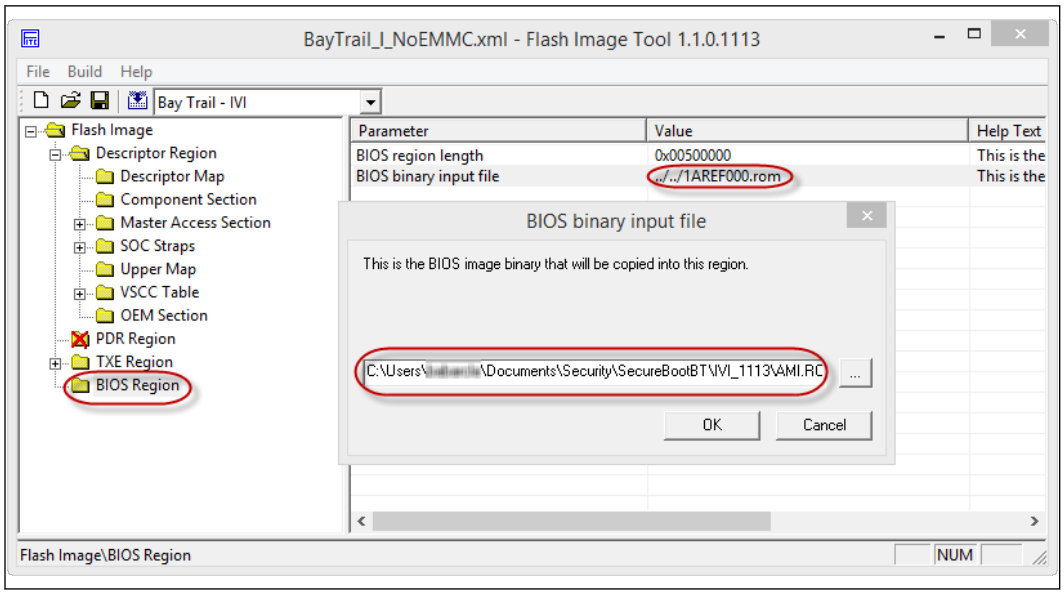


4. Click the open icon. Navigate to `C:\Users\<user>\Documents\Security \SecureBootBT\IVI_1113`.
5. Select `BayTrail_I_NoEMMC.xml` and then click **Open**:



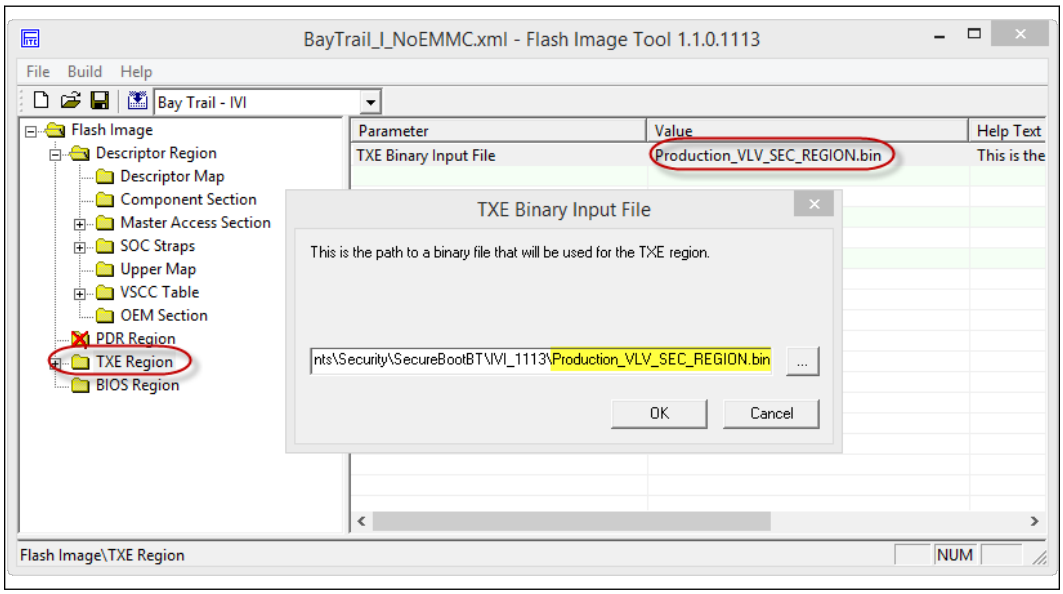
6. Select **BIOS Region** from the left pane.
7. In the **Value** column browse to select the **.rom** line. The BIOS region loads.

   Tip: If your selection list is large, you can double-click the column headings to change the sort order.
8. Click **OK** to accept `AMI.ROM` as the new BIOS input file.



9. Select **TXE Region** from the left pane.
10. Select **TXE Binary Input File** from the right pane.

11. Double-click the **Value** column header and browse to select `C:\Users\<user>\Documents\Security\SecureBootBT\IVI_1113\Production_VLV_SEC_REGION.bin`.
12. Click **OK**. The Intel® Trusted Execution Engine region loads.



If you need to use FPF mirroring continue to Load the FPF Configuration File for FPF Mirroring.

If you do not need FPF mirroring skip to Flash the Gateway with the SPI Flash Image.

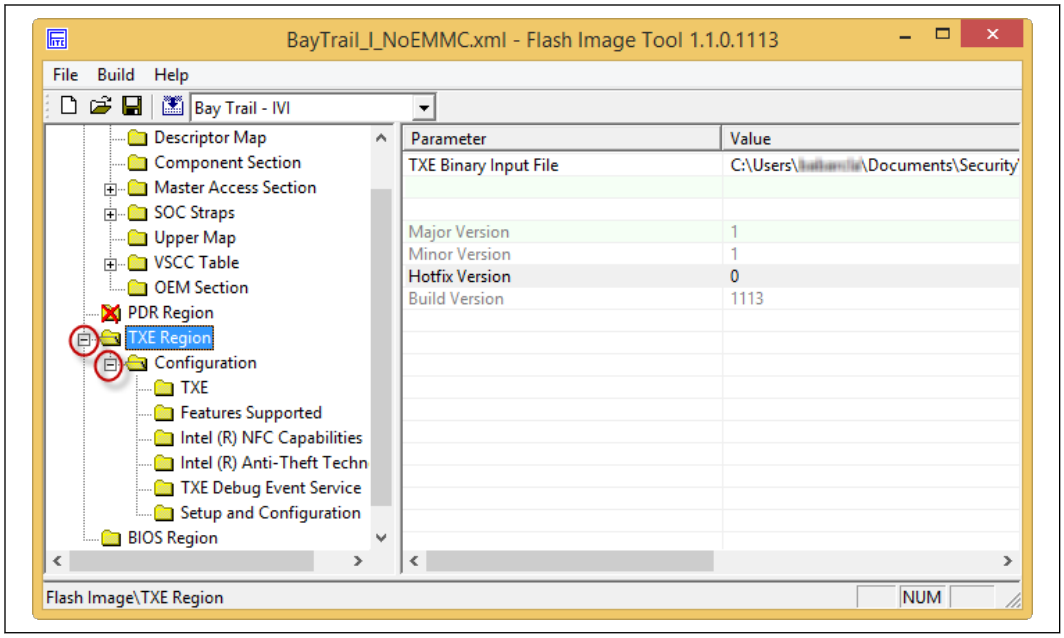
Note: In a production environment the FPF mirroring configuration file should not be included in the stitching process.

Load the FPF Configuration File for FPF Mirroring

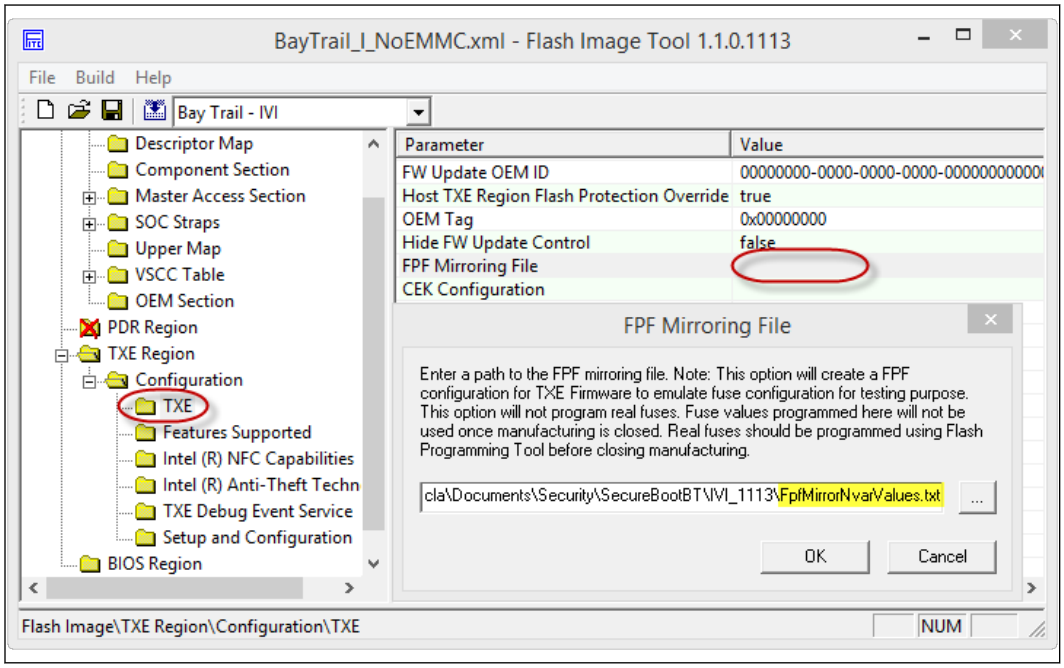
Either the end-user or the ODM can complete the steps in this section. ODM completion is recommended. An Intel CNDA is required to access the necessary tools. If you are an end user, contact your ODM for information.

Use this section if you need to use FPM mirroring. In a production environment the FPF Mirroring configuration file should not be included in the stitching process.

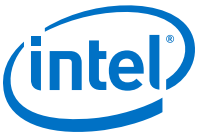
1. Click **+** to expand **TXE Region** in the left pane.
2. Click **+** to expand **Configuration** in the left pane.



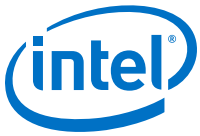
3. Select **TXE** from the left pane.
4. Select **FPF Mirroring File** from the right pane.
5. Double click the **Value** column heading and browse to select `C:\Users\<user>\Documents\Security\SecureBootBT\IVI_1113\FpfMirrorNvarValues.txt`. Click **OK**.



6. Build the binary image: Click **Build** from the top menu and then **Build Image**. This produces the 8M SPI binary image called `outimage.bin`.



Continue to Flash the Gateway with the SPI Flash Image.



5.0 Flash the Gateway with the SPI Flash Image

To complete these steps, you will use a Linux computer, Windows computer, and your Gateway. Before flashing your Gateway, you will prepare a USB flash drive. **The contents of this flash drive will be erased.**

Prepare USB Flash Drive

Either the end-user or the ODM can complete the steps in this section. An Intel CNDA is required to access the necessary tools. Contact your ODM for information.

Begin these steps on a Linux computer with your Gateway powered off.

1. Insert a 1 GB or larger USB flash drive into your Linux computer. **The contents of this flash drive will be erased.**
2. Format the USB flash drive:

```
sudo mkdosfs -n 'bios' -I /dev/sd?
```

3. Remove the USB flash drive from the Linux computer and insert it into your Windows computer. The Windows computer automatically reads the flash drive as `bios`.
4. Copy `fparts.txt` and `fpt64.efi` from the `F2A_BIOS` directory to the USB flash drive.
5. Copy `outimage.bin` from the `IVI_1113` directory to the USB flash drive.
6. Remove the USB flash drive from your Windows computer.

Choose how you want to flash the SPI image onto your Gateway and follow the appropriate steps:

- UEFI shell: Continue with Flashing the SPI Image Onto the Gateway Using UEFI Shell.
- Dediprog: Continue with Flashing the SPI Image Onto the Gateway Using Dediprog.

Note: The UEFI shell method requires a working BIOS on the Gateway. If your Gateway does not have a working BIOS use the Dediprog steps.

Flashing the SPI Image Onto the Gateway Using UEFI Shell

Either the end-user or the ODM can complete the steps in this section. An Intel CNDA is required to access the necessary tools. Contact your ODM for information.

1. Insert the USB flash drive into the Gateway.
2. Power on the Gateway and immediately press the `DEL` key to load BIOS Setup.
3. Click the **Boot** tab.
4. Choose **UEFI** for **Boot Option #1**.

5. Save your changes and exit BIOS Setup. The Gateway boots into EFI.
6. At the prompt, enter the `fp0` partition on the USB flash drive:

```
fp0
```

7. Confirm the BIOS files are present:

```
ls
```

   Make sure your output includes:

   - `fparts.txt`
   - `fpt64.efi`
   - `outimage.bin`

   The presence of these files indicates you are in the correct partition. If the files are not present, repeat steps 6 and 7, using `fp1` in step 6.

   Important: Do not continue before confirming that you are in the partition that contains these files.

8. Complete the BIOS upgrade:

```
fpt64.efi -F outimage.bin
```

   Watch for errors while this command executes.

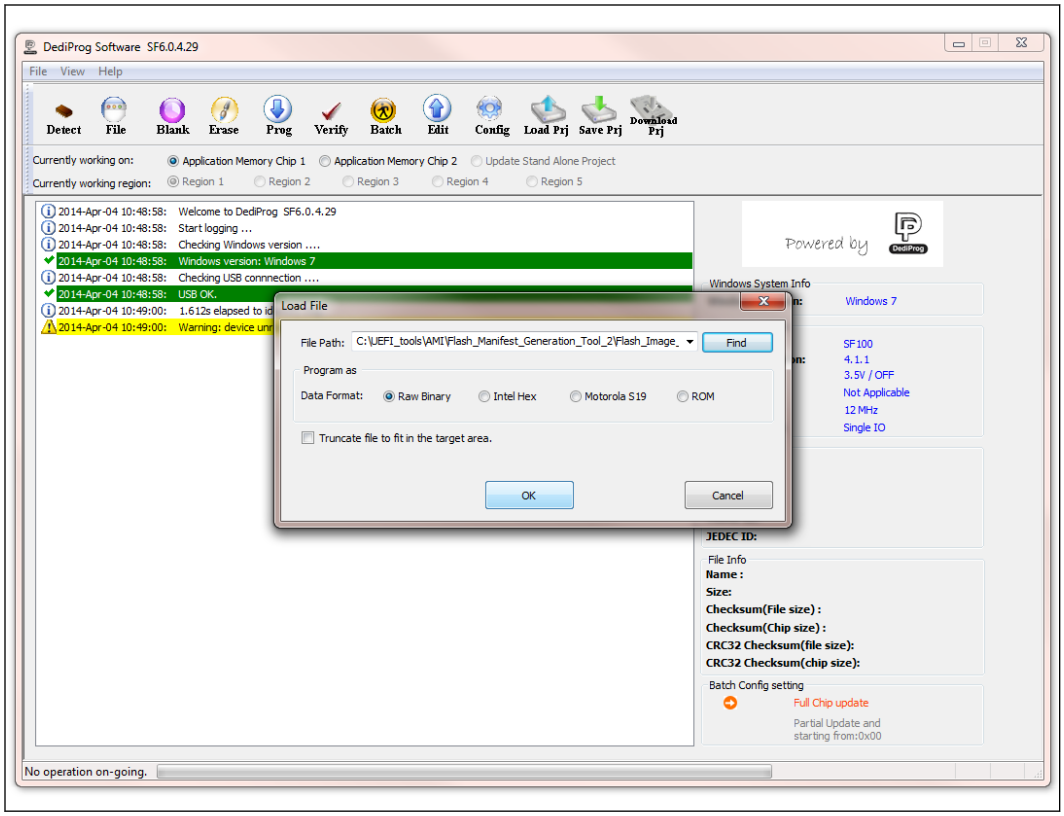
Flashing the SPI Image Onto the Gateway Using Dediprog

Either the end-user or the ODM can complete the steps in this section. An Intel CNDA is required to access the necessary tools. If you are an end user, contact your ODM for information.

Disregard this section if you completed the steps in Flashing the SPI Image Onto the Gateway Using UEFI Shell.

Begin these steps onto a computer running Linux.

1. Power down your Gateway and leave it powered down until the end of these steps.
2. Connect the Dediprog* SF100 programmer into your Linux computer's USB port.
3. Connect the 8-pin header to the Gateway.
4. Launch the DediProg engineering software on your Linux computer.
5. Click **File** and browse to select `outimage.bin`:



6. Click **Batch** to begin programming the SPI.
7. After programming completes, click **Verify** to verify the checksum of the written image in SPI and in the file buffer.
8. If checksum verification is successful, disconnect the Dediprog programmer header from the Gateway.
9. Power up your Gateway.

6.0 Program the Field-Programmable Fuses

Warning: **The process in this section is irreversible and can be completed only one time. Errors will render your Gateway permanently inoperable. Proceed with caution.**

Either the end-user or the ODM can complete the steps in this section. **ODM completion is strongly recommended.** An Intel CNDA is required to access the necessary tools. Contact your ODM for information.

Perform these steps on your Gateway with your Gateway powered off.

1. Insert the USB flash drive from Flash the Gateway with the SPI Flash Image into your Gateway.
2. Power on your Gateway and boot it to the EFI shell.
3. Map the USB flash drive to the EFI shell:

```
map -r
```

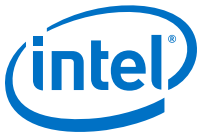
4. Change the path to the USB flash drive, for example, `Shell>fs0:`
5. Read the default values in FPF:

```
fpt64.efi -READFPF OEM_KEY_HASH_1
fpt64.efi -READFPF SECURE_BOOT_EN
fpt64.efi -READFPF Global_Valid
```

6. Program and read back the OEM KEY HASH values, replacing <OEM hash value> with the actual hash value. The hash value looks similar to `0x2D8CE4A658C6A45D2B64C7FB3D73A31893EB58274BB383274BBAFDD355E55016`.

   Warning: Do not use this sample hash value. **You must use your actual hash value.**

```
fpt64.efi -WRITEFPF OEM_KEY_HASH_1 -v <OEM hash value>
fpt64.efi -READFPF OEM_KEY_HASH_1
```



7.0 Implement UEFI Secure Boot Stage 3

These steps are for the end user.

UEFI Secure Boot is the technology name that the UEFI Forum uses to describe UEFI verification of UEFI applications. References to UEFI Secure Boot in this document refer to the process of the UEFI Stage 2 verification of the UEFI application called `grub.efi`. UEFI Secure Boot is used to verify the authenticity of the `grub.efi` boot loader. Once `grub.efi` has been authenticated, it verifies the authenticity of the kernel.

Note: This section assumes you have built your Wind River® Intelligent Device Platform XT 3 Gateway operating system. For help with it see https://software.intel.com/en-us/Setup-IDP-DevelopmentTools.

To complete the steps you will need these files that were created when you built your Gateway OS:

File	Description	Location
`bzImage--3.14-r0.2-intel-baytrail-64-<build_date>.bin`	Kernel image file.	`$Projdir/export/images/`
`bzImage-initramfs-intel-baytrail-64.bin`	Kernel image file with initramfs.	`$Projdir/export/images/`
`grub.efi`	Bootloader image file.	`$Projdir/build/grub-efi-0.97-r4/image/boot/grub/`
`grub.conf`	Bootloader configure file.	`$Projdir/build/grub-efi-0.97-r4/image/boot/grub/`
`wrlinux-image-idp-intel-baytrail-64.tar.bz2`	Intelligent Device Platform XT 3 root file system tar ball.	`$Projdir/export/images/`
`wrlinux-image-idp-intel-baytrail-64-dist-srm.tar.bz2`	Intelligent Device Platform XT 3 root file system signed by SST tar ball.	`$Projdir/export/images/`
`wrlinux-image-idp-intel-baytrail-64.ext3`	Intelligent Device Platform XT 3 root file system block file.	`$Projdir/export/images/`
`wrlinux-image-glibc-idp-intel-atom-baytrail-srm.ext3`	Intelligent Device Platform XT 3 root file system signed by SST block file.	`$Projdir/export/image/`

In addition, you will need the SST tool for the UEFI Secure Boot implementation process. This tool was provided with Wind River® Intelligent Device Platform XT 3 and is used to sign:

- grub.efi
- kernel

- RootFS
- RPM

Create Owner Public and Private Keys

These steps are for the end user.

In this section you will create a chain of certificates and keys. The root of this chain is an RSA pair called Owner-Cert and Owner-Private. The owner can bring a private key and use SST to generate a new public cert, based on the existing private key.

These are the SST options:

Option	Description	Current Value	Default Value
`verbose`	Open or close the signing trace. Value can be `yes` or `no`.	`no`	`no`
`name`	User defined name for the role.	`owner`	Role name.
`output-dir`	The output directory for your private key and CA certificate.	`./outputE`	SST directory (`"."`)
`role`	Trust role in SRM. Can be `vendor` or `owner`.	`owner`	N/A

The following example shows the generation of the Owner x509v3 certificate and private key.

1. Use the table above to modify and execute:

```
./SST create-key --role=owner --name=ownerE \ --output-dir=./outputE
```

   The result is two files in the `./outputE` directory:

   - `owner-cert.pem`
   - `owner-private.pem`

2. If the owner has a private key, then generate the owner certificate, where `owner-key.pem` is the existing private key:

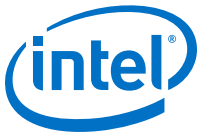
```
./SST create-key --role=owner --name=owner --output-dir=./outputE --priv-
key=owner-key.pem
```

   The resulting owner certificate is placed in the `./outputE` directory as `owner-cert.pem`

Create Vendor Public and Private Keys

These steps are for the end user.

In this section you will create a vendor RSA key pair that is signed by the owner's private key. The vendor is an authorized entity by the owner. The vendor can provide a private key and use SST to generate a new public cert based on the existing private key.



These are the SST options:

Option	Description	Current Value	Default Value
`lverbose`	Open or close the signing trace. Value can be `yes` or `no`.	`no`	`no`
`name`	Vendor name.	`vendorE`	Role name.
`output-dir`	The output directory where you can find your private key and CA certificate.	`./outputE`	SST directory (".")
`role`	Trust role in SRM. Can be `vendor` or `owner`.	`vendor`	N/A
`issuer`	The name of the issuer who delegates to this vendor.	`owner`	`owner`

The following example shows the generation of the Vendor x509v3 certificate and private key.

1. Use the table above to modify and execute the command.

```
./SST create-key --role=vendor --name=vendor --output-dir=./outputE --issuer=owner
```

   The result is two files in the `./outputE` directory:

   - `vendor.pem`
   - `vendor-private.pem`

2. If the vendor has a private key, then generate the vendor certificate, where `vendor-key.pem` is the existing private key:

```
./SST create-key --role=vendor --name=vendor --output-dir=./outputE --priv-
key=vendor-key.pem
```

   The resulting owner certificate is placed in the `./outputE` directory as `vendor-cert.pem`

Automatically Signing Bootloader, Kernel, RootFS, and RPM Packages

These steps are for the end user.

If SRM is enabled on your Gateway, you can sign the bootloader, kernel, RootFS, and RPM packages at once using the steps in this section.

If SRM is not enabled on your Gateway, disregard this section and continue with Sign the Bootloader.

1. Run the configure command.
2. Copy these files from the `outputE` directory to `$HOME/Project/layers/wr-idp/wr-srm/files/keys/` :
   - `owner-cert.pem`
   - `owner-private.pem`

- `vendor.pem`
- `vendor-private.pem`

3. Run `make`:

```
make fs
```

All items are signed. Skip ahead to Configure UEFI for Secure Boot.

Sign the Bootloader

These steps are for the end user.

If you successfully followed the steps in Automatically Signing Bootloader, Kernel, RootFS, and RPM Packages disregard this section and skip ahead to Configure UEFI for Secure Boot.

The SST is used to sign and update the `grub.efi` bootloader with the appropriate signatures and keys. The update process includes hashing and encrypting `grub.efi` with the vendor private key. `grub.efi` is also updated to include the `grub.efi` signature, vendor certificate, and owner certificate.

The commands used to sign the bootloader are:

Option	Description	Current Value	Default Value
`verbose`	Open or close the signing trace. Value can be `yes` or `no`.	`no`	`no`
`owner-cert`	The root certificate of the trust chain.	`owner-cert.pem`	The `owner-cert.pem` file in the SST directory.
`vendor-cert`	The device vendor certificate.	`vendor-cert.pem`	The `vendor-cert.pem` file in the SST directory.
`priv-key`	The device vendor private key.	`vendor-private.pem`	The `vendor-private.pem` file in the SST directory.

Sign the `grub.efi` bootloader using the table above for modifications as necessary:

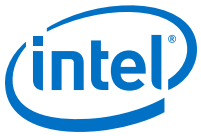
```
./SST sign-bootloader --owner-cert=./owner-cert.pem --vendor-cert=./vendor-
cert.pem --priv-key=./vendor-private.pem ./grub.efi
```

Continue to Sign the Kernel.

Sign the Kernel

These steps are for the end user.

If you successfully followed the steps in Automatically Signing Bootloader, Kernel, RootFS, and RPM Packages disregard this section and skip ahead to Configure UEFI for Secure Boot.



The SST is used to sign and update the kernel with the appropriate signatures and keys. The update process includes hashing the kernel and encrypting the kernel with the vendor private key. The kernel also is updated to include the kernel signature and Vendor certificate.

The commands used to sign the kernel are:

Option	Description	Current Value	Default Value
`verbose`	Open or close the signing trace. Value can be `yes` or `no`	`no`	`no`
`vendor-cert`	The device vendor certificate.	`vendor-cert.pem`	The `vendor-cert.pem` file in the SST directory
`priv-key`	The device vendor private key.	`vendor-private.pem`	The `vendor-private.pem` file in the SST directory

Sign the kernel using the table above to modify the command as necessary:

```
./SST sign-kernel --vendor-cert=./vendor-cert.pem --priv-key=./vendor-
private.pem ./bzImage-initramfs-intel-baytrail-64.bin
```

Continue to Sign RootFS.

Sign RootFS

These steps are for the end user.

If you successfully followed the steps in Automatically Signing Bootloader, Kernel, RootFS, and RPM Packages disregard this section and skip ahead to Configure UEFI for Secure Boot.

A signature on RootFS ensures the binary and library in RootFS can be verified successfully after RootFS is deployed to the Gateway. The kernel and bootloader image in RootFS can also be signed when using SST's sign-all command to sign RootFS.

The commands used to sign RootFS are:

Option	Description	Current Value	Default Value
`verbose`	Open or close the signing trace. Value can be `yes` or `no`.	`no`	`no`
`vendor-cert`	The device vendor certificate.	`vendor-cert.pem`	The `vendor-cert.pem` file in the SST directory.
`priv-key`	The device vendor private key.	`vendor-private.pem`	The `vendor-private.pem` file in the SST directory.

continued...

Option	Description	Current Value	Default Value
`Owner-cert`	The root certificate of the trust chain.	`owner-cert.pem`	The `owner-cert.pem` file in the SST directory.
`Output`	The signed RootFS output.	`./signed-images.tar.bz2`	The `srm-enabled-images.tar.bz2` file in the directory.
`Mode`	RootFS type. Value can be `tarball` or `blockfile`.	`tarball`	`tarball`

Sign the RootFS tar ball (`wrlinux-image-glibc-idp-intel-atom-baytrail.tar.bz2`) using the table above to modify the command as necessary:

```
./SST sign-all --mode=tarball --vendor-cert=./vendor-cert.pem --priv-key=./vendor-private.pem --owner-cert=./owner-cert.pem --output=./signed-images.tar.bz2 ./ wrlinux-image-idp-intel-baytrail-64.tar.bz2
```

Continue to Sign RPM Packages.

Sign RPM Packages

These steps are for the end user.

If you successfully followed the steps in Automatically Signing Bootloader, Kernel, RootFS, and RPM Packages disregard this section and skip ahead to Configure UEFI for Secure Boot.

When the Intelligent Device Platform XT 3 SRM feature is enabled, an RPM package must be signed by the SST before it can be installed on the Gateway.

The commands used to sign the RPM Packages are:

Option	Description	Current Value	Default Value
`priv-key`	The device vendor private key.	`vendor-private.pem`	The `vendor-private.pem` file in the SST current directory.
`Mode`	Sign single RPM or multiple RPM packages. Options are `rpm` or `dir`.	`rpm`	`rpm`

Sign the RPM package (`example.atom.rpm`) using the table above to modify the command as necessary:

```
./SST sign-rpm --mode=rpm --priv-key=./vendor-private.pem ./ example.atom.rpm
```

Optional: sign multiple RPM packages:

```
./SST sign-rpm --mode=dir --priv-key=./vendor-private.pem ./ rpm-dir
```

Upon command completion, all RPM packages in the `rpm-dir` directory will be signed.

Continue to Configure UEFI for Secure Boot.

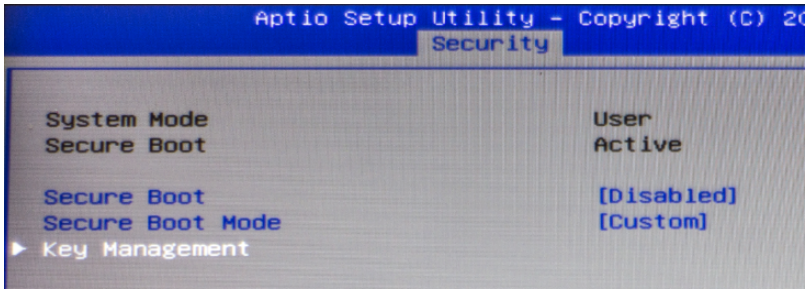
8.0 Configure UEFI for Secure Boot

This section provides a synopsis of the instructions to enroll the grub.efi and kernel keys into the UEFI secure key database to complete the Stage 3 secure boot setup. For the complete process of deploying the RootFS image see the Wind River® Intelligent Device Platform XT Programmer's Guide at https://knowledge.windriver.com/Content_Lookup?id=045671.

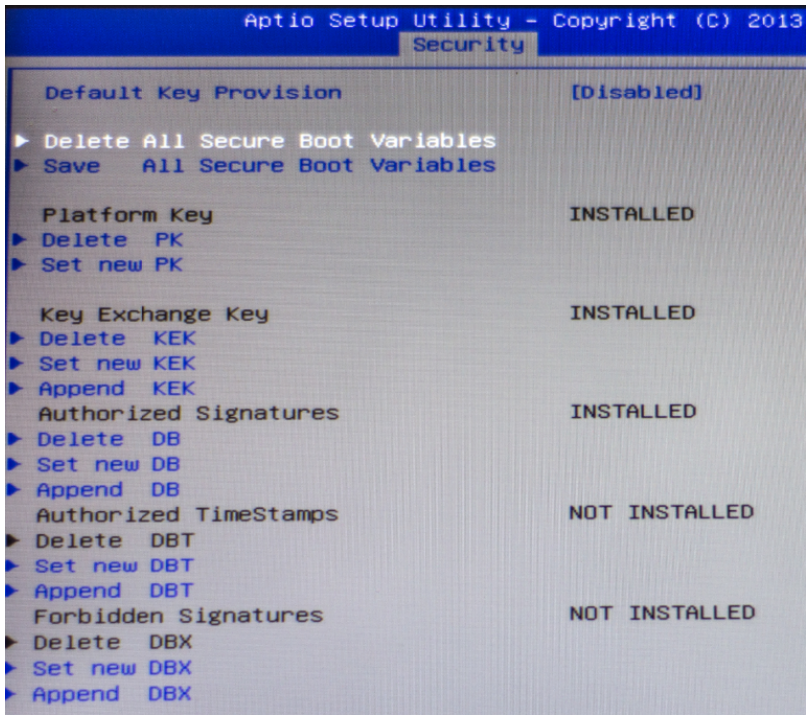
Perform these steps on your Gateway with your Gateway powered down.

Change the BIOS Settings

1. Insert the USB flash drive that contains the SRM image into the Gateway.
2. Power on your Gateway and boot into BIOS Setup.
3. Select the **Security** tab.
4. Disable **Secure Boot**.
5. Select **Key Management**.

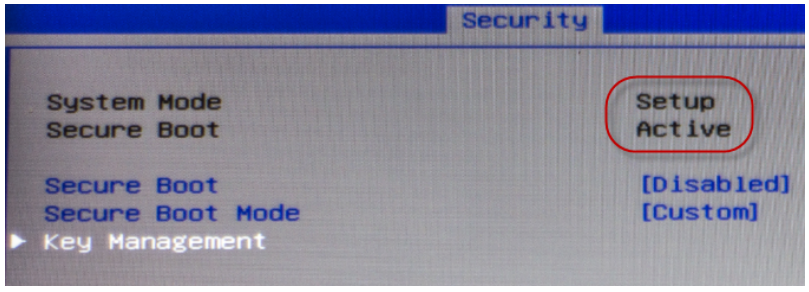


6. Select **Delete All Secure Boot Variables**. Select **Yes** at the prompt to proceed.

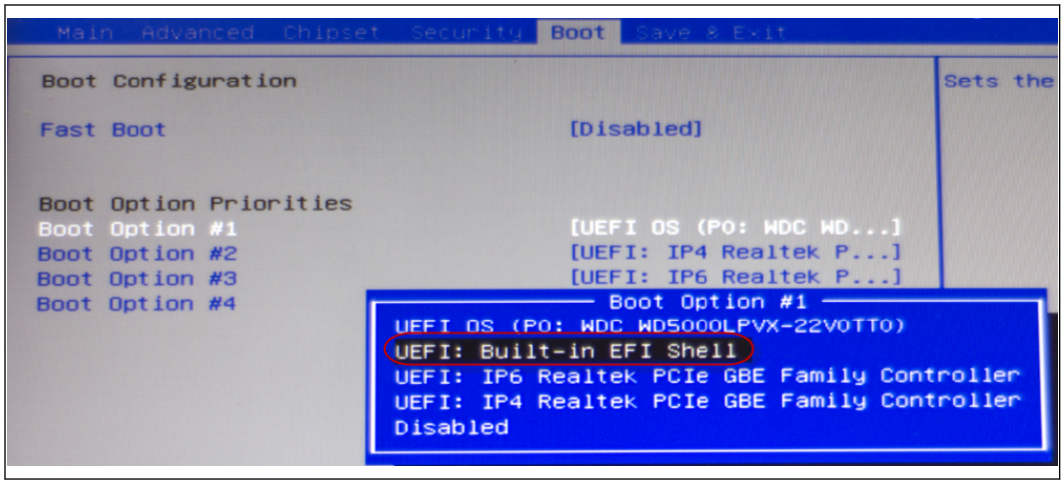


Your screen will display **NOT INSTALLED** for each variable.

7. Press **Esc** to return to the **Security** screen. Confirm you are in **Setup** mode:



8. Press **Esc** to return to the **Main** menu.
9. Select the **Advanced** tab and then **CSM Configuration**.
10. If the screen does not indicate **CSM configuration is disabled**, then disable it. CSM must be disabled for Secure Boot.
11. Press **Esc** to return to the **Main** menu.
12. Select the **Boot** tab and then **Boot Option #1**.
13. Select **UEFI: Built-in EFI Shell**:



14. Save your changes, exit BIOS Setup, and restart your Gateway. It boots into the UEFI shell.

Continue with Enable Secure Boot.

Enable Secure Boot

1. On the boot screen look for the line that includes **Removable HardDisk** and note the `fsX` at the beginning of the line, where `X` is either `0` or `1`.
2. At the `Shell>` prompt, go to the boot partition on the USB flash drive. Replace `X` with `0` or `1`:

```
fsX
```

3. Go to the grub directory:

```
cd EFI\BOOT
```

4. List the contents of the directory to make sure it includes `BOOTIA32.efi`

```
dir
```

5. Run the grub to enroll the keys:

```
.\BOOTX64.efi
```

   The keys are enrolled in the UEFI database.

6. Restart your Gateway and boot into BIOS Setup.
7. Select the to the **Security** tab.
8. Enable **Secure Boot**.
9. With the USB flash drive still inserted into the Gateway, save your changes, exit BIOS Setup, restart your Gateway and return to BIOS Setup.
10. Select the **Boot** tab and then **Boot Option #1**.
11. Select **UEFI: <the USB flash drive>**

12. Save your changes and restart your Gateway. In addition to other information, your boot screen displays:
    - UEFI Secure Boot: Enabled
    - GRUB Verified Boot: Enabled
13. Transfer the Wind River® Intelligent Device Platform XT 3 image to the hard drive on the Gateway:

```
tgt=/dev/sda /sbin/reset_media
```

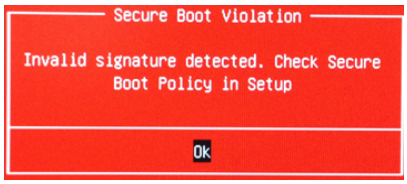
14. Remove the USB flash drive from the Gateway and power down the Gateway.

Continue to Test Secure Boot.

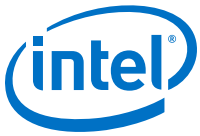
Test Secure Boot

Use the following steps to make sure that an unsigned `grub.efi` denies kernel boot.

1. Insert the USB flash drive into a Linux computer.
2. Copy an unsigned `grub.efi` to `/EFI/BOOT` in the VFAT partition of the USB flash drive. Name it `UNSIGNED_BOOTIA32.efi`
3. Plug the USB flash drive into the Gateway and power up the Gateway.
4. Press the **<Del>** key and select the **UEFI: Built-in EFI Shell** option from the **Boot Device Menu**.
5. Run `UNSIGNED_BOOTIA32.efi`. You will receive a warning message stating that your access is denied. This indicates that the Secure Boot policy is being enforced.
6. Return the USB flash drive to the Linux computer to create an Intelligent Device Platform XT 3 image with a signature that is different from the signature in the UEFI database.
7. Transfer the image to a USB flash drive.
8. In Enable Secure Boot you changed your boot device priority so your Gateway boots from the USB flash drive. If your Gateway is not booting from the USB flash drive, return to the earlier section to change the boot priority.
9. Boot from the USB flash drive. You will see the following message indicating that the secure boot policy is being enforced:



You have successfully tested Secure Boot. If desired, you can change your BIOS settings to boot directly from the Gateway.



9.0 Conclusion

In this guide, you accomplished these tasks:

- Installed software on a Microsoft Windows* computer and set up Win32OpenSSL.
- Created a Secure Boot image and installed it on your Gateway.
- Programmed the field-programmable fuses.
- Implemented Secure Boot Stage 3.
- Enabled Secure Boot on your Gateway and tested it.