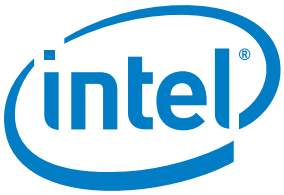


Bay Trail-M/D Platform – Intel® Trusted Execution Engine (Intel® TXE) 1.1 SKU Firmware for Windows* and UEFI based Android*

Bring-Up Guide

Revision 1.6

January 2014

Intel Confidential

Contents

Figures

Tables

Revision History

Revision Number	Description	Revision Date
1.3	Added Widevine* Provisioning support (Steps 6,7 in chapter 3.4.1, Chapter 7 and 15: Appendix A)	October 2013
1.4	New Content: - Chapters 4, 5, 6, and Chapter 16: Appendix B: Using Local Android Intel® TXE System Tools - Added ciphertext CEK in Appendix A: Google* Widevine* for Intel® TXE - Added Note in Intel® TXEI Driver section that Installation is relevant only for Windows* OS	November 2013
1.5	New Content: - Using WinPE Tools – Chapter 9 - Added DELAY_PCIE_RLS is SOC Strap 4	December 2013
1.6	Intel® TXE FW 1.1 Point Release. Added this note in introduction: Please note: Some of the chapters in this document are relevant for Android projects only (Chapter 7, Appendix A, and Appendix B)	January 2014

§

1 Introduction

This document covers the future Intel® Pentium® processor or future Intel® Celeron® processor N- & J- series based platform (formerly Bay Trail-M/D platform) firmware bring-up procedure for Intel® quad-core technology SoC (B2 silicon).

The bring-up procedure primarily involves building a FW image. Once the FW image is built, it can be programmed to the Bay Trail platform. All the paths mentioned in this guide are relative path to the root of the given kit.

Please note: Some of the chapters in this document are relevant for Android projects only (Chapter 7, Appendix A, and Appendix B)

1.1 Terminology

Term	Description
FITC	Flash Image Tool Creation
FPT	Flash Programming Tool
Intel® TXE	Intel® Trusted Execution Engine (Intel® TXE)
Intel® TXEI	Intel® Trusted Execution Engine Interface (Intel® TXEI)
ADB	Android* Debug Bridge

§

2 Quick Start Check-List

2.1 First Boot of Bay Trail-M/D

To run first basic boot of the Bay Trail-M/D platform, ensure to have the following:

- Build the SPI FW image
  - Build the image using FITC tool as described in section 3.4 and flash the image components from the FW kit
- Flash the SPI FW image
  - Flash using Dediprog or FPT method as described in section Flashing Target3.6
- If using Windows OS: Install Intel® Trusted Execution Engine Interface (Intel® TXEI) Driver as described in section 8
  - Once the platform boots, install the Intel TXEI driver found in the FW Kit
- Verify Intel® Trusted Execution Engine (Intel® TXE) information
  - Run TXEInfo tool found in the FW kit "\System Tools\ TXEInfo\" directory
- Verify Intel TXE status via TXEManuf tool
  - Run TXEManuf tool found in the FW kit "\System Tools\ TXEManuf\" directory
  - Use TXEManuf.cfg to enable/disable tests of interest

For more details on each of these steps, please refer to the appropriate chapter within the Intel TXE FW Bring up Guide.

§

3 Procedure

3.1 Prerequisites

- fitc.exe: can be found under \\System tools\Flash_Image_Tool folder
- DediProg SF100
- FPT can be found under \\System tools\Flash_Programming_Tool

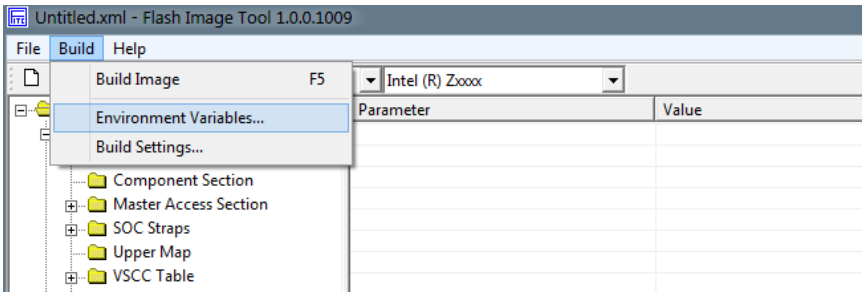
3.2 Start FITC

- Invoke Flash Image Tool by navigating to \\System tools\Flash_Image_Tool folder
- Double click fitc.exe.

3.3 Set Up Build Environment

In the main menu select Build→ Environment Variables.

Figure 1. Enviroment Variables



Edit your configuration as shown below.

- **$Source Dir:** The location where FITC will look for binary images during the image creation process
- **$DestDir:** The location where FITC will save the binary image
- **$WorkingDir:** The location where FITC.exe is running. Please keep it as ".”

Figure 2. Environment Variables

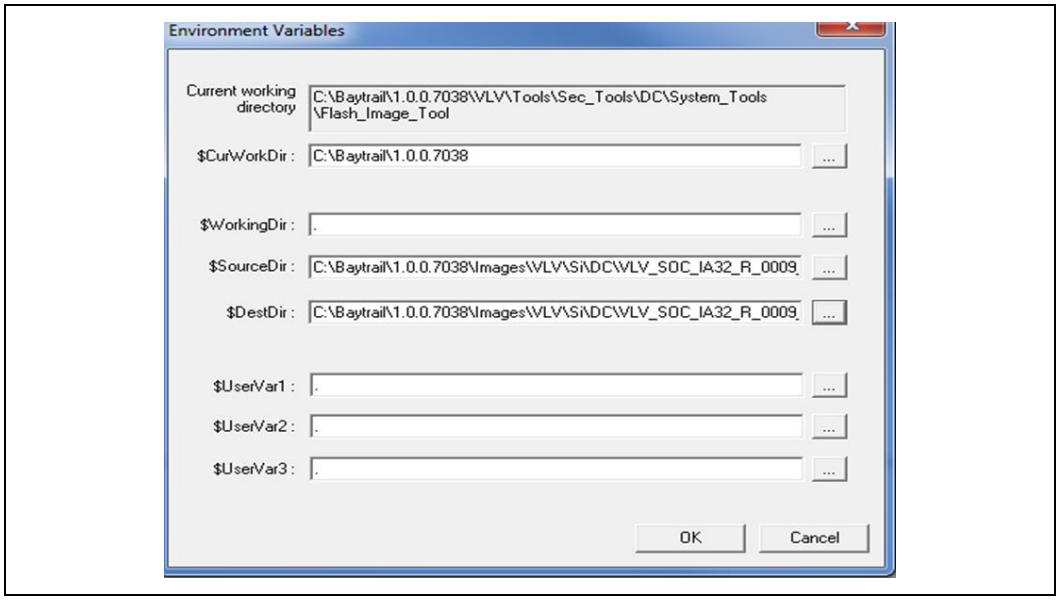
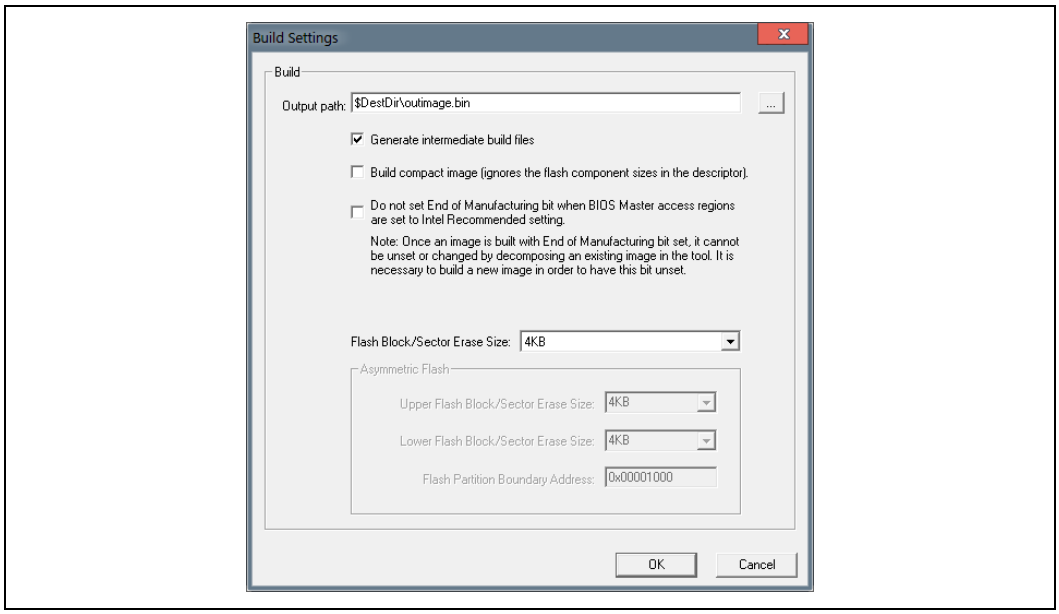


Figure 3. FITC Set Up



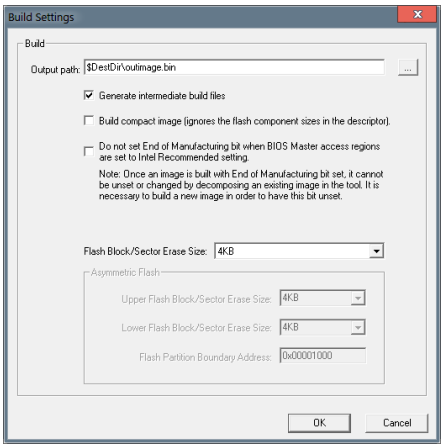
Note: Please use the environment variables when defining output path in Build → Build Settings, as shown

3.4 Create Flash Image

3.4.1 Using GUI

1. Run fitc.exe.
2. Define output image name and path
3. Build → Build Settings → Output path

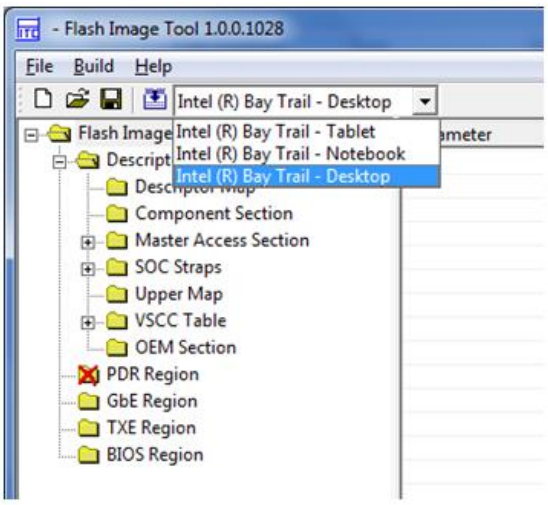
Figure 4. Define Output Path



4. Select platform type
    a. On the platform selection list, select Bay Trail – Notebook or Bay Trail – Desktop before modifying and building the flash image.

SOC straps definition will be changed upon Platform selection.

Figure 5. Select Platform Type



5. Fill in Intel TXE Region:
    a. Select "TXE Region" and double click "TXE Binary Input File"
    b. Select \\Image Components\TXE\. .*.bin

Please note: Steps 6-7 are relevant for Widevine feature only.

6. Construct FITC_CEK.bin following chapter 15: Appendix instructions
7. Add FITC_CEK.bin file (Flash Image -> TXE Region -> Configuration -> TXE -> CEK Configuration)

Figure 6. Add FITC_CEK

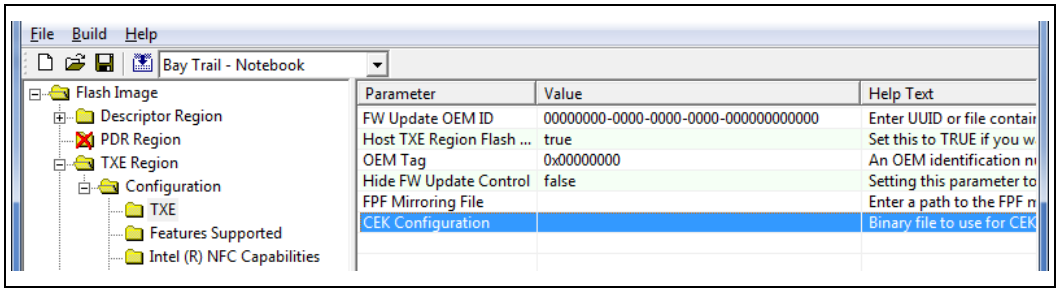
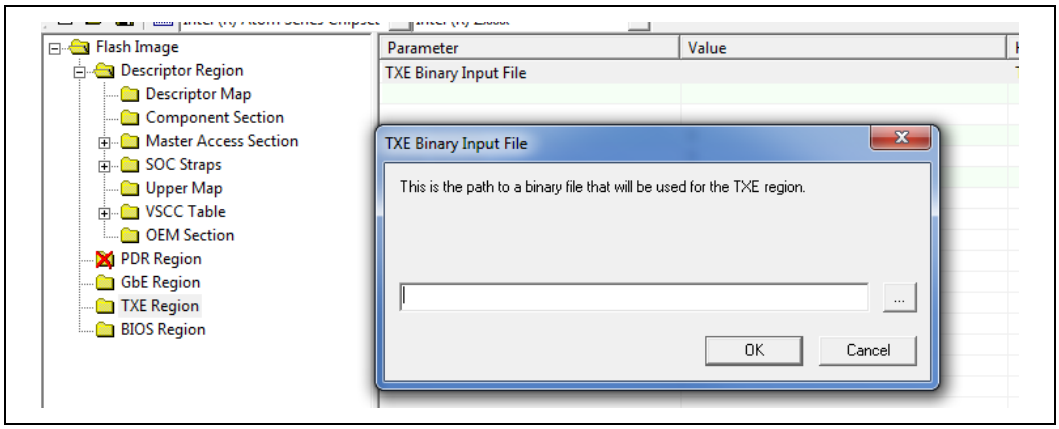
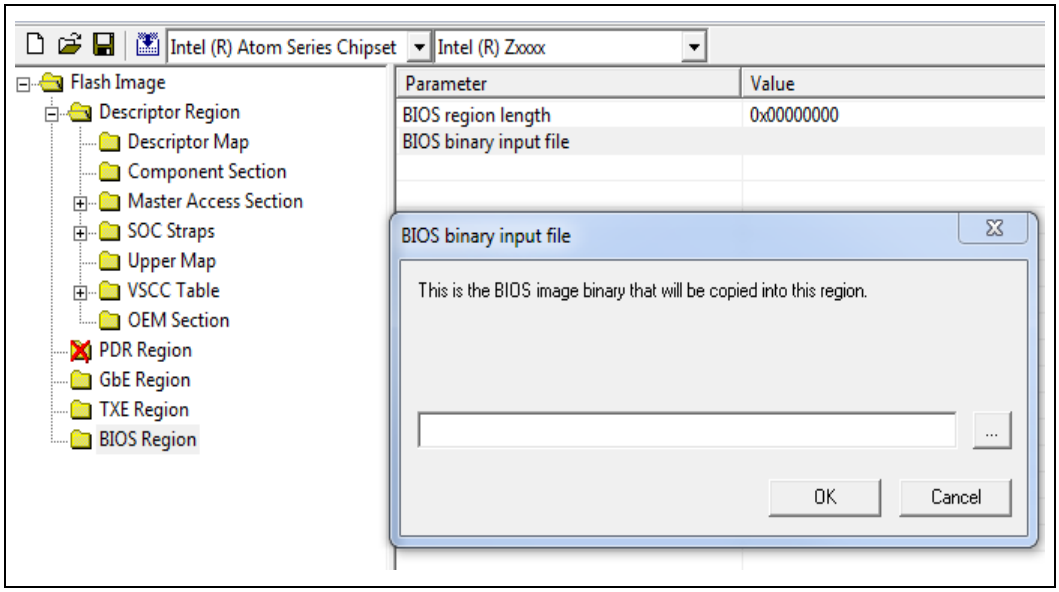


Figure 7. Intel® TXE Region



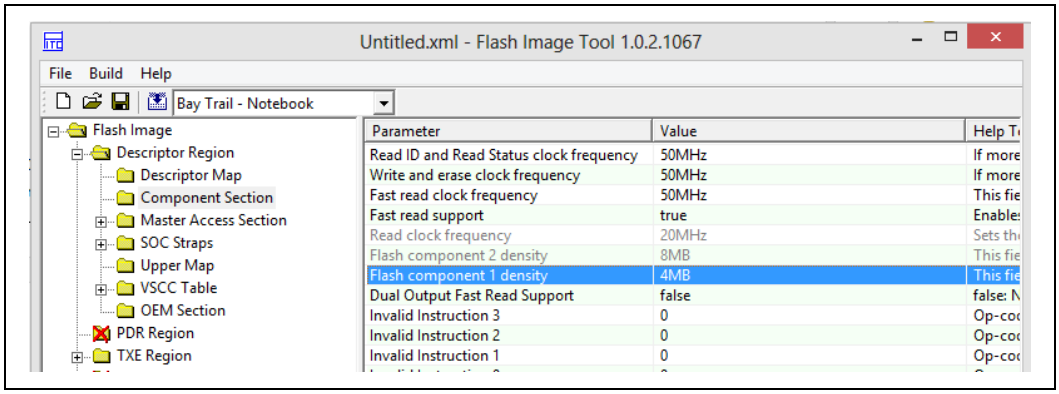
8. Fill in BIOS Region:
    a. Select "BIOS Region" and double click "BIOS binary input file"
    b. Load BIOS image (*.ROM)

Figure 8. BIOS Region



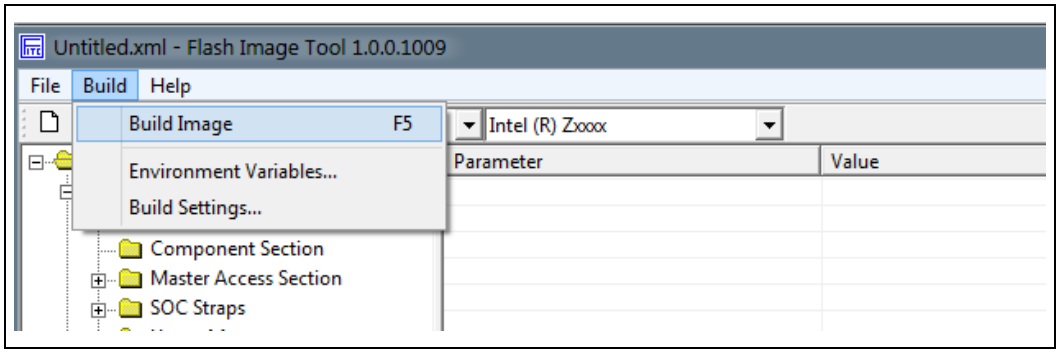
9. Configure SPI Flash image size
   a. Select Flash Image/ Component Section/ Flash component 1 density
   b. Configure the size of the SPI Flash image (e.g. 8MB, 4MB)
   c. Save XML

Figure 9. Configure Flash Image Size



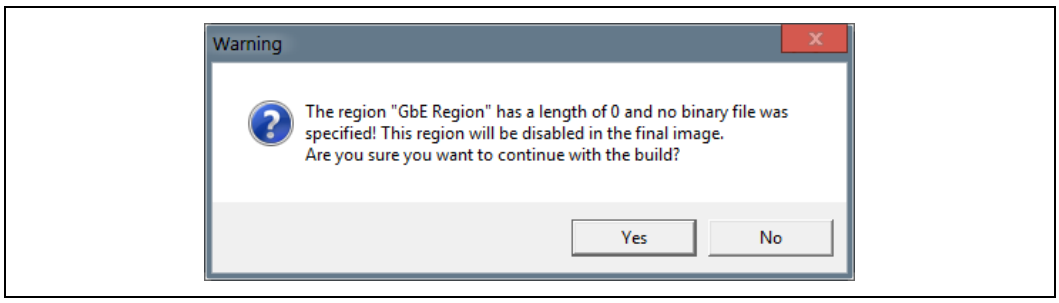
10. Build image

Figure 10. Build Image



Note: A warning will appear when building the image, Click "Yes".

Figure 11. Warning Message



The output image will be located at the path given at step 3.4.1

Figure 12. Image Output

3.4.2 Using Command Line

Please use the command line to create image through command line:

fitc.exe newfiletmpl.xml -b -txe PRODUCTION_TXE_Region.bin -bios BIOS_Region.ROM

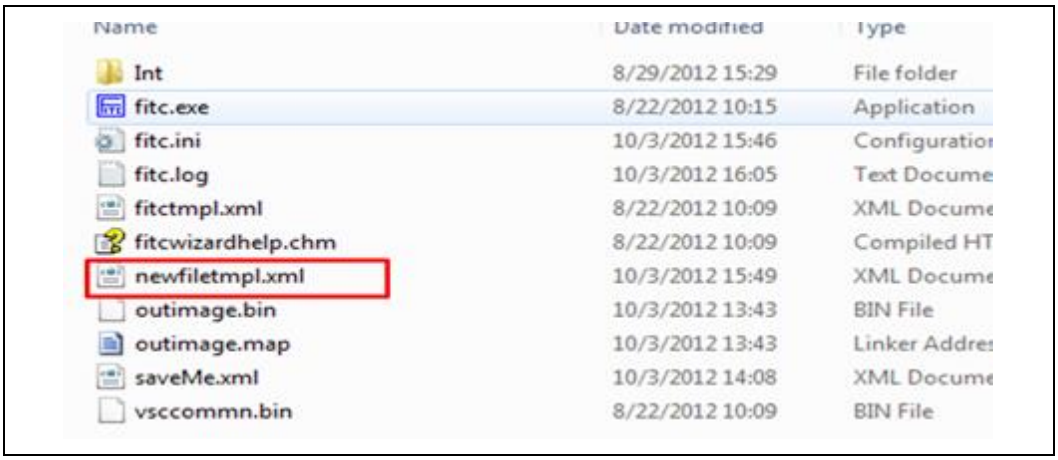
- Please note, if the Intel TXE Region or BIOS region not at the same directory as FITC tool you will need to mention the path

3.5 XML Configuration

3.5.1 Save Your Settings

11. When opening FITC.exe, it loads defaults settings defined in newfiletmpl.xml (located in the same folder as FITC.exe)

Figure 13. FITC Configuration



12. To save your custom settings, in the main menu select File→Save As. Select a name and location for the XML file that contains all the settings configured so far. It is recommended that you save this file in the same directory as FITC.exe is located for easy access.

Figure 14. Save Configuration

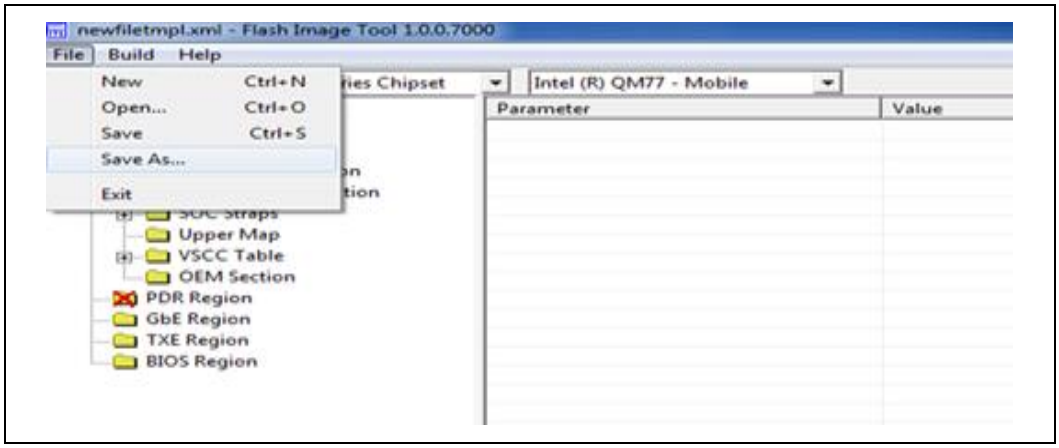
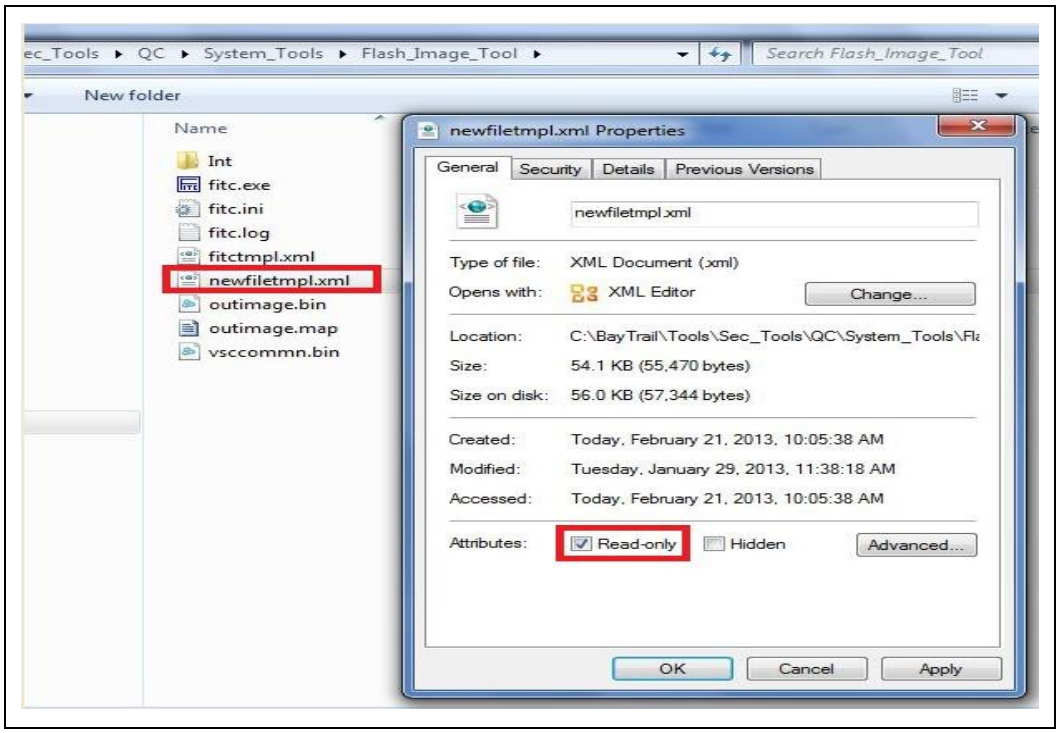


Figure 15. Configuration Protection

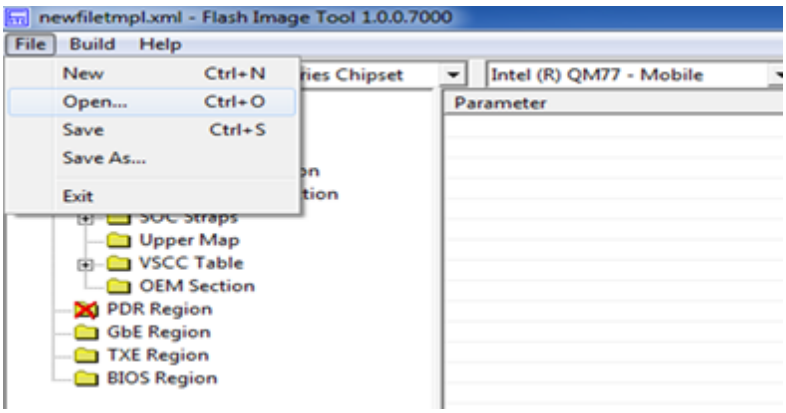


13. Protect configuration XML file from accidental changes by checking the "Read-only" attribute, as shown below.

3.5.2 Load FITC Configuration

Default or custom settings can be loaded by selecting File→Open in the main menu, and navigating to the desired xml configuration file.

Figure 16. Load Configuration



3.6 Flashing Target

3.6.1 Using DediProg

14. Run DediProg Software
15. Click "Detect" to verify SPI flash detection
16. Please make sure the voltage is set to 1.8v by clicking Config-> Miscellaneous settings
17. Click "File" button and select the FW image built in step 3.4.

Figure 17. Set Voltage

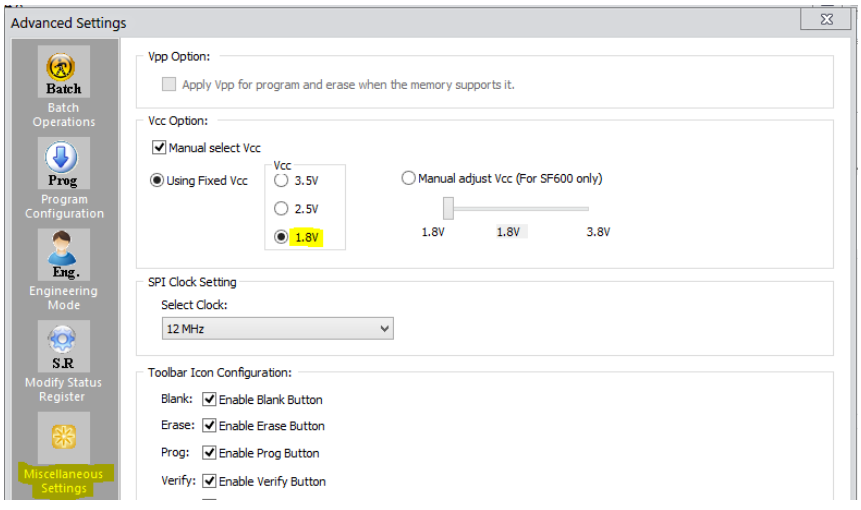
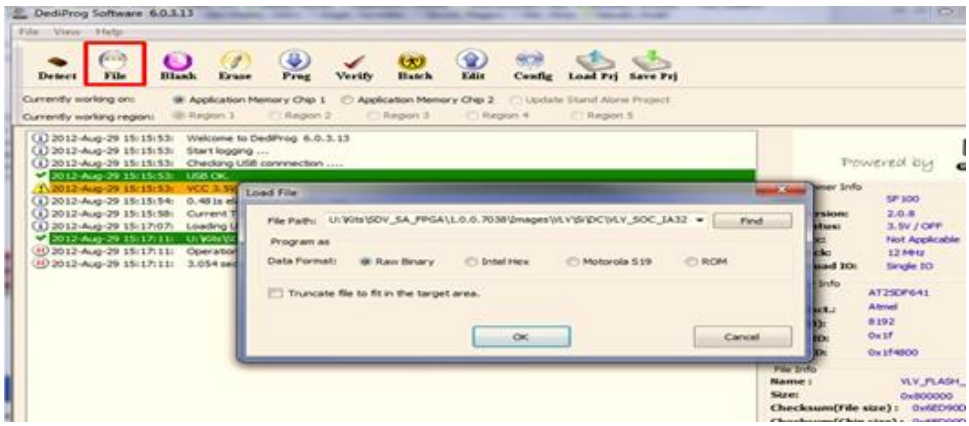
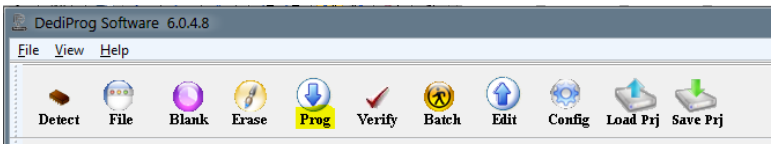


Figure 18. Select Image



18. Click "Prog" to flash the flash image to the target

Figure 19. Prog Option



19. Verify flashing was performed correctly

Figure 20. Result Expected

3.6.2 Using FPT

1. FPT is a Windows based tool aimed to program FW on the platform (FPT is running from the platform). Under \\System tools\Flash_Programming_Tool

2. Copy the FW image to the root folder of the FPT tool and rename it to outimage.bin. (For simplicity, we will use \\Flash_Programming_Tool\Windows when referring to FPT tool directory)
3. Open command line with administrative privileges, navigate to \\Flash_Programming_Tool\Windows or Windows64 and type:

```
Ftpt.exe -LIST
```

The system should respond with the number of SPI Flash devices available. For example:

```
--- Flash Devices Found ---
W25Q64BV ID:0xEF4017 Size: 8192KB (65536Kb)
W25Q64BV ID:0xEF4017 Size: 8192KB (65536Kb)
```

4. Program the SPI Flash image to the Flash device(s) by issuing the following command at the prompt:

```
fptw.exe /f outimage.bin
```

5. If the programming was successful, then the following message will be shown:

```
FPT Operation Passed
```

§

4 FPF Configuration File

- FPF configuration file is an input to FPT, FITC, and to Manifest Generation tool.
- FPF configuration file can be found under \\System tools\FpfConfigFile.txt.
- The file contains a list of fuses, where each line describes a fuse file in the following pattern: ***[ID]:[Value]:[Locked]***
  ID: Fuse file ID
  Value: Desired value of fuse file in **hex** digits, must be byte-aligned (For single bit file, should be 00 or 01)
  Locked: Boolean indicates if the file should be locked (TRUE/FALSE).
  Example: FUSE_FILE_ALT_BIOS_LIMIT:1FFF:FALSE

Table 1. Fuse Files

Name	Description
OEM_KEY_HASH	Hash of the key material used by the OEM to sign the Secure Boot Manifest
ALT_BIOS_LIMIT	Alternative BIOS limit. Used to locate the Alternative copy of the IBB and the manifests. This is actually the13 MSbits of the physical address. LSbits assumed to be 0xFFF
SB_EN	This bit indicates that SB it is enable and the other values where already configured
KEY_MANIFEST_ID	This is the ID of the of the Key Manifest (if 0 no Key Manifest is required)
FUSE_FILE_GLOBAL_VALID	FW Flag that marks that all OEM Fuses have been programed.
FUSE_FILE_TPM_DISABLE	FW SKU flag: marks if Firmware TPM is enabled in the Platform.

4.1 FPF Mirroring

4.1.1 Motivation

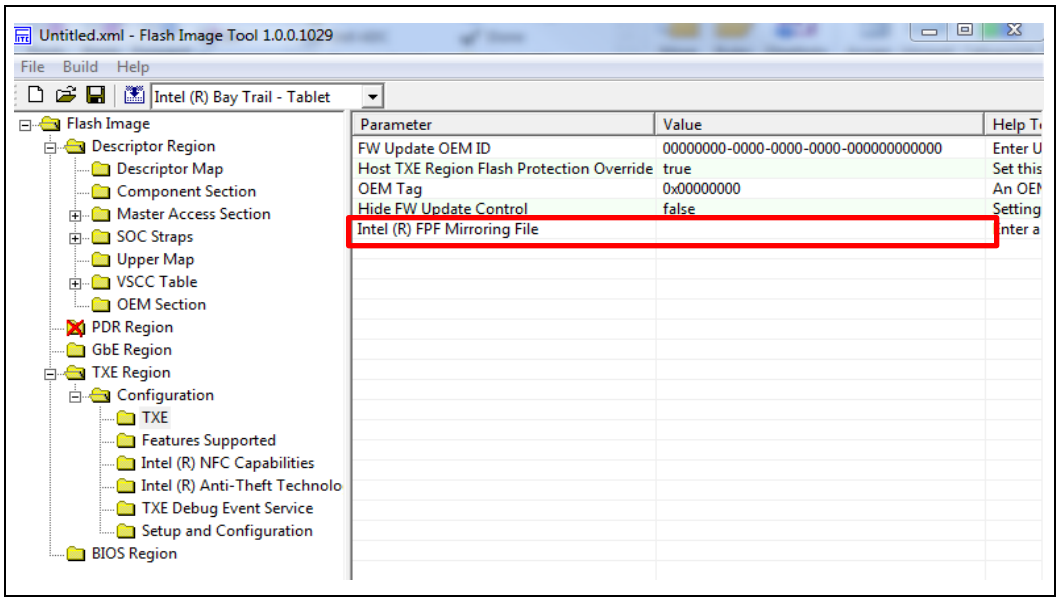
FPF mirroring allows validation\testing of FPF on the FW level, as oppose to manufacturing phase, where the FPF's are HW fused.

FPF mirroring is done via FPF configuration file.

4.1.2 FPF Mirroring in FITC

1. Edit the FPF configuration file according to the required features. (default matches platform POR)
2. Fill in the Intel TXE region in FITC, as described in section 3.4.1.
3. Double click on Intel FPF Mirroring File in TXE Region->Configuration->TXE and navigate to the edited FPF configuration file, as depicted.

Figure 21. FPF Mirroring in FITC



5 Intel® TXE Secure Boot Manifest Generation Tool (FLAMInGO Tool)

Intel® TXE Secure Boot Manifest is used to authenticate the BIOS Initial Boot Block (IBB) and KEY MANIFEST

Key Manifest authenticates the key sued to sign the Intel TXE Secure Boot Manifest. Key Manifest is optional.

Intel recommended method is Key Manifest

5.1 FLAMInGo Tool Parameters

Table 2. Parameters

Parameter Name	Description
PublicKeyFile	Public key file to calculate Sha256 from
HashFileout	Name of the file to place the SHA256 digest of the public key
FuseConfigFile	Name of the file that contains the fuses configuration
ManifestName	String that identifies the manifest, same name must be used when completing the manifest generation
IBBFile	Name of the file that contains IBB data (maximum 127kb)
SVN	Security Version Number
SigningKey	Name of the file that contains the public key of the key that is used to sign the manifest
OEMDataFile	Name of the file that contains OEM data (maximum 400 bytes)
KeyManifestFile	Name of the file that contains a valid key manifest generated by this
SignatureFile	Name of the file that contains an RSA signature of the hash file generated when creating a manifest
Unsigned	Creates the blob of the manifest without generating a hash (optional)

- For further information type “FLAMInGO.exe ?”

5.2 Creating Secure Boot Manifest & Signing IBB

This section will cover creating and signing of the SB manifest which is required to enable verified boot.

Prerequisites:

- Public and private key to be used for signing the BIOS
- FLAMinGo tool (can be found in the FW kit)
- SampleSigner (can be found in the FW kit)
- FPFconfigFile.txt (can be found the FW Kit)
- IBB (127KB) – Initial Boot Block

Creation and Signing Procedure

1. Open CMD and use Flamingo to hash the certificate:
   FLAMInGO.exe HashKey pubkey.cer PubKeyHash.txt
   the output will be the PubKeyHash.txt which will include the hash of the public key.

2. Insert the hash of the public key from the previous step into the FPFconfigFile.txt under FUSE_FILE_OEM_KEY_HASH_1,
   and enable secure boot by setting the FUSE_FILE_SECURE_BOOT_EN value to 01.
   the result should look like this (with your own hash key inside):

Figure 22. FPFconfigFile.txt Example

```
#This Fuse bit is for enabling Verified Boot. Change value to "01" to enable Secure/verified boot
FUSE_FILE_SECURE_BOOT_EN:01:FALSE

#Hash of the public part of the OEM signing key obtained with the Flamingo tool
FUSE_FILE_OEM_KEY_HASH_1:12343211232135432135432135431887351351351346213643543541356434 13:FALSE

#The 13 Most Significant Bits of address of alternate copy of IBB within BIOS region
#Alt_bios_limit file is 16 bits wide; applicable values are up to 0x1FFF (13 bits effective).
FUSE_FILE_ALT_BIOS_LIMIT:0000:FALSE

#This is the ID of the of the Key Manifest ('0' indicates no key manifest is required)
FUSE_FILE_KEY_MANIFEST_ID:00:FALSE
```

3. Use Flamingo to create the Secure Boot manifest
   FLAMInGo.exe SBManCreate [FuseConfigFile] [Manifest Name] [IBBFile] [SVN][Signing Key] [-OEMDataFile <OEMDataFile>]
   The output will be the hash of the Secure Boot manifest and an .xml structure file with the name [Manifest Name]

4. Sign the Hash of the Secure Boot Manifest using sampleSigner:
   SampleSigner.exe [HashFileToSign] [PrivateKeyFile] [OutSignatureFile]
   The output will be the signature file.

5. Complete the process by integrating the Secure Boot manifest with the IBB
   FLAMInGo.exe SBManComplete [FuseConfigFile] [Manifest Name] [Signature File]
   The output here will be a 128KB .bin file which will includes the Secure Boot manifest (1KB) and the IBB (rest of the 127KB)

6. Copy the 128KB file to the end of the BIOS or the full SPI image.

Sample run of the tools:
FLAMInGO.exe HashKey pubkey.cer PubKeyHash.txt
FLAMInGO.exe SBManCreate FuseConfigFile.txt manifest IBB.bin 2 pubkey.cer -oemdatafile oemdata.bin
SampleSigner.exe manifest_SB_hash.bin pubkey.cer manifest_sig.bin
FLAMInGO.exe SBManComplete FuseConfigFile.txt manifest manifest_sig.bin

5.3 Creating Key Manifest & Signing IBB

This section will cover creating and signing of the Key manifest which is required to perform verified boot using Key Manifest.

Prerequisites:

- Create two Public and private key sets and a certificate from each pair, one for the Key manifest (e.g KMpubkey) and one for the secure boot manifest (e.g SBpubkey).
- FLAMinGo tool (can be found in the FW kit)
- SampleSigner (can be found in the FW kit)
- FPFconfigFile.txt (can be found the FW Kit)
- IBB (127KB)

Creation and Signing Procedure

1. Use Flamingo to hash the Key Manifest certificate:
   FLAMInGO.exe HashKey KMpubkey.cer KMPubKeyHash.txt
   the output will be the KMPubKeyHash.txt which will include the hash of the public key of the Key Manifest.

2. In the FPFconfigFile.txt file:
   - Insert the hash of the Key Manifest public key from the previous section under FUSE_FILE_OEM_KEY_HASH_1
   - Enable secure boot by setting the FUSE_FILE_SECURE_BOOT_EN value to 01
   - Set the Key Manifest ID under FUSE_FILE_KEY_MANIFEST_ID, note that the Key Manifest ID should be different form 00.
   - The result should look like this (with your own hash key and Key Manifest ID inside):

Figure 23. FPFconfigFile.txt Example

```
#This Fuse bit is for enabling Verified Boot. Change value to "01" to enable Secure/verified boot
FUSE_FILE_SECURE_BOOT_EN:01:FALSE

#Hash of the public part of the OEM signing key obtained with the Flamingo tool
FUSE_FILE_OEM_KEY_HASH_1:12343211232135432135432135431887351351351346213643543541356434 13:FALSE

#The 13 Most Significant Bits of address of alternate copy of IBB within BIOS region
#Alt_bios_limit file is 16 bits wide; applicable values are up to 0x1FFF (13 bits effective).
FUSE_FILE_ALT_BIOS_LIMIT:0000:FALSE

#This is the ID of the of the Key Manifest ('0' indicates no key manifest is required)
FUSE_FILE_KEY_MANIFEST_ID:01:FALSE
```

3. Use Flamingo to generate the hash of the Key Manifest:
   FLAMInGo.exe KeyManCreate [FuseConfigFile] [ManifestName][KeyToCerify] [SVN] [SigningKey]
   - KeyToCertify is the the certificate of the Secure Boot Manifest to be used (e.g SBpubkey)
   - SVN will be set to 0 for testing (this is relevant for Key revocation)
   - SigningKey is the certificate of the Key Manifest

4. Sign the hash using SampleSigner
   SampleSigner.exe [HashFileToSign] [KMPrivateKeyFile] [OutSignatureFile]
   the output will be the signature file.

5. Complete the creation of the Key Manifest
   FLAMInGo.exe KeyManComplete [FuseConfigFile] [ManifestName] [SignatureFile]
   - With this step giving the Key Manifest, the next step is to create the Secure Boot Manifest (using the Key Manifest in the process).

6. Create the Secure Boot Manifest (the OEM data file is optional)
   FLAMInGo.exe SBManCreate [FuseConfigFile] [Manifest Name] [IBBFile] [SVN][Signing Key] [-OEMDataFile <OEMDataFile>][-KeyManifestFile <KeyManifestFile>]
   - The FuseConfigFile (FPFconfigFile.txt file) is the original one from the Key Manifest creating
   - SVN will be set to 0 for testing
   - The KeyManifestFile is the output of step 5
   - The output of this section is the hash of the Secure Boot Manifest

7. Sign the hash using sample signer
   SampleSigner.exe [HashFileToSign] [PrivateKeyFile] [OutSignatureFile]
   - The output of this step is the signature file ([OutSignatureFile]).

8. Complete the process by integrating the Secure Boot Manifest, the Key Manifest and the IBB

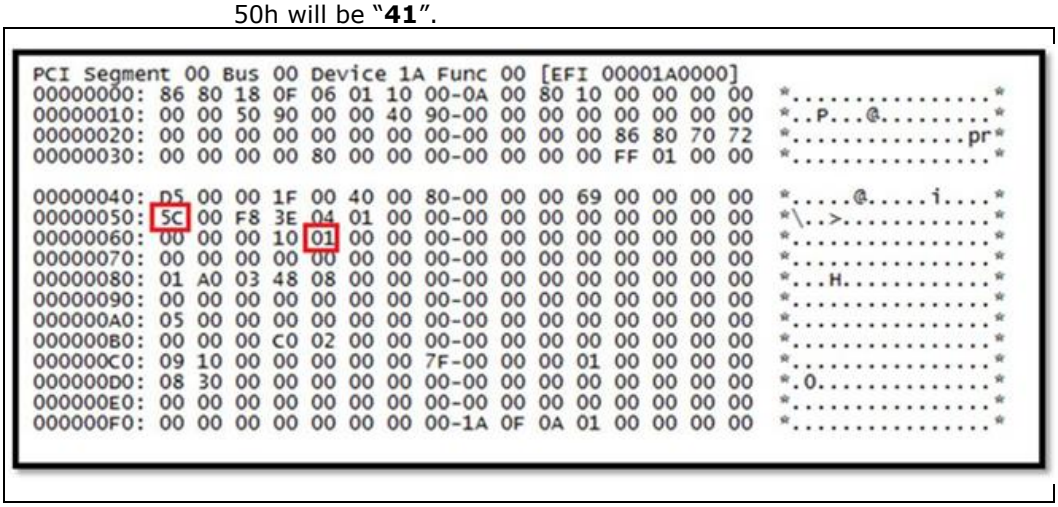
FLAMInGo.exe SBManComplete [FuseConfigFile] [Manifest Name] [Signature File]

- The FuseConfigFile (FPFconfigFile.txt file) here is the original one used in the creation of the Key Manifest.
- The output from this step is a 132KB file that includes the Key Manifest [4KB], the Secure Boot Manifest [1KB] and the IBB [127KB].

9. Copy the 132KB file to the end of the BIOS or the full SPI image.

5.4 How to Confirm Verified Boot Executed Successfully

1. From the EFI shell run "PCI 1A 0"
   a. Verify offset 50h shows "**5C**": verified boot executed using mirroring, if not executed the value will be "**00**".
      - When enabling verified boot using fused silicon, the value of offset 50h will be "**41**".



   b. Run: "MEM FFFE0000 –b"
      - Verify at this offset in memory that IBB is shielded from the user (had been copied to the SRAM) as shown below.

      If verified boot had not been executed, the IBB will be exposed in this offset.

Figure 24. Verified Boot Enabled – IBB is shielded from User

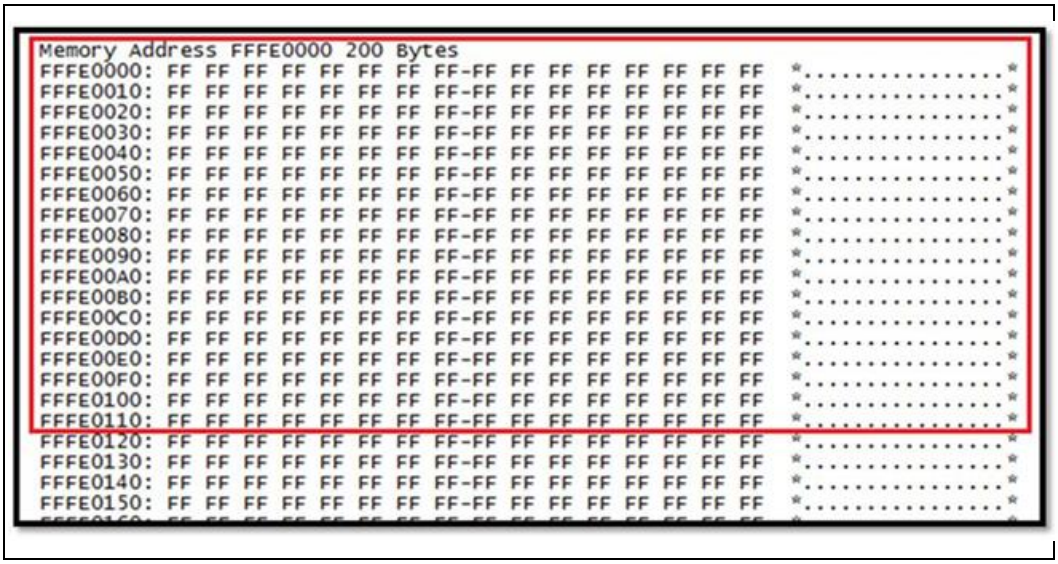
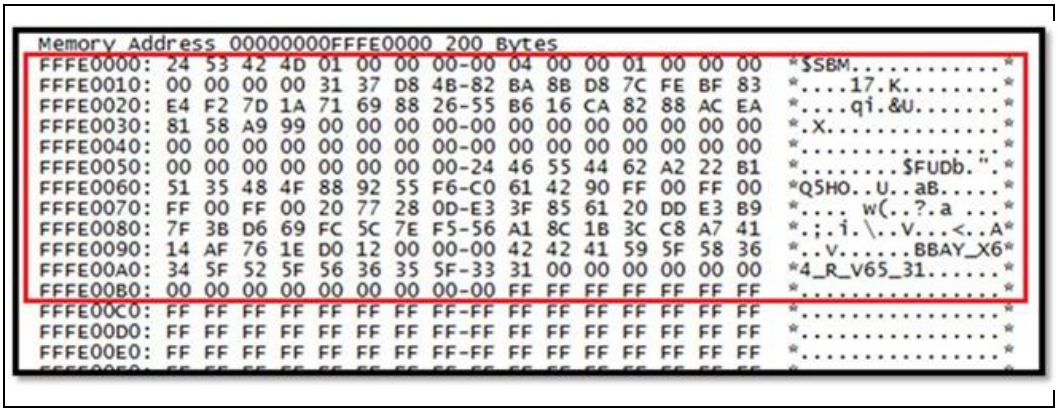


Figure 25. Verified Boot had Not Been Executed – IBB is visible



§

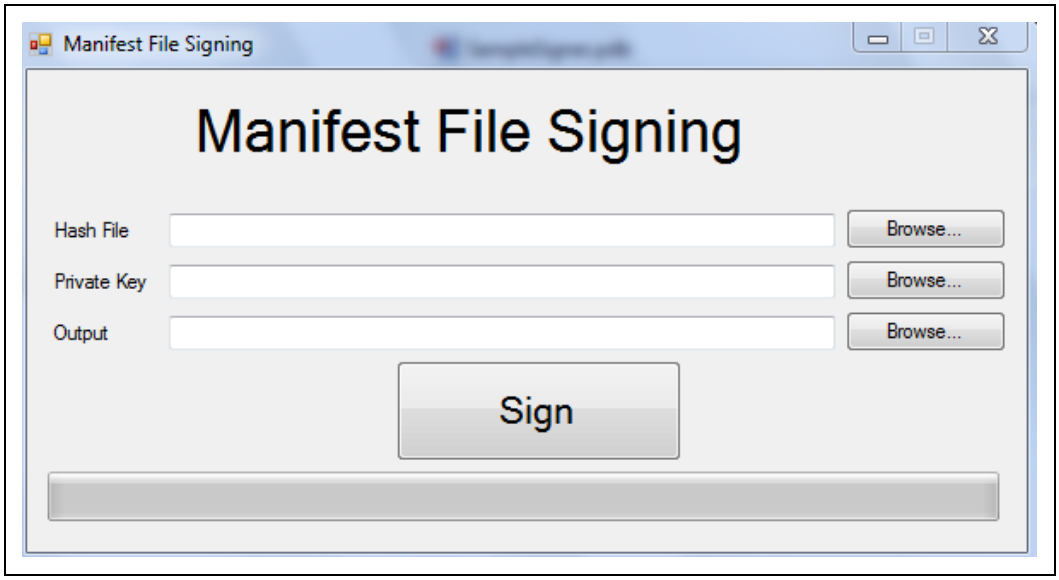
6 Sample Signer –Verified Boot Manifest Signing Reference Tool

Signing reference tool can be found under \\Flash Manifest Generation Tool \SampleSigner.exe.

Parameters:

1. Hash file to sign
2. Private key
3. The output location of the signed file.

Figure 26. Signing Tool



7 Widevine* KeyBox Provisioning Procedure

For full manufacturing flow details, please refer to "Bay Trail M/D Platform, Manufacturing Recommendation for Intel® Trusted Execution Engine (Intel® TXE) 1.1 SKU Firmware for SR'14 UEFI Android Platform " Document (IBL# 541110)

1. Provision Widevine using IV (Initialization Vector) and encrypted KeyBox file (refer to 15 for files creation procedure)
   - Run FPT –provkb <iv_and_keybox.bin>

```
...\Flash_Programming_Tool\Windows>fptw.exe -provkb iv_and_keybox.bin

Intel (R) Flash Programming Tool. Version:  1.0.2.1071
Copyright (c) 2007 - 2013, Intel Corporation. All rights reserved.

Platform: Intel(R) Mainstream Express Chipset
Reading HSFSTS register... Flash Descriptor: Valid

--- Flash Devices Found ---
    W25Q64BV    ID:0xEF4017    Size: 8192KB (65536Kb)


Keybox Provisioning Operation: Successful
```

2. Optional: Verify that the Widevine device has been properly provisioned
   - Run TXEInfo –feat "keybox"

```
...\TXEInfo\Windows>TXEInfoWin.exe -feat “keybox”

Intel(R) TXEInfo Version: 1.0.2.1071
Copyright(C) 2005 - 2013, Intel Corporation. All rights reserved.

Keybox:                                 Provisioned
```

3. After properly closing manufacturing (using FPT-closemnf), run TXEManuf EOL Testing.
   Edit TXEManuf.cfg file in EOL section
   - Uncomment "SubTestName "Validate Keybox Provisioning"" test in order to include WV Provisioning Test check

```
SubTestName="TXE Manufacturing Mode status"
SubTestName="Flash Region Access Permissions"
SubTestName="CF9GR lock check"
SubTestName="FPF Global Valid bit check"
// SubTestName="Security Descriptor Override (SDO) check"
SubTestName="Validate Keybox Provisioning"
```

- Run TXEManuf –EOL

```
...\TXEManuf\Windows>TXEManufWin.exe -EOL

Intel(R) TXEManuf Version: 1.0.2.1071
Copyright(C) 2005 - 2013, Intel Corporation. All rights reserved.


TXEManuf End-Of-Line Test Passed
```

§

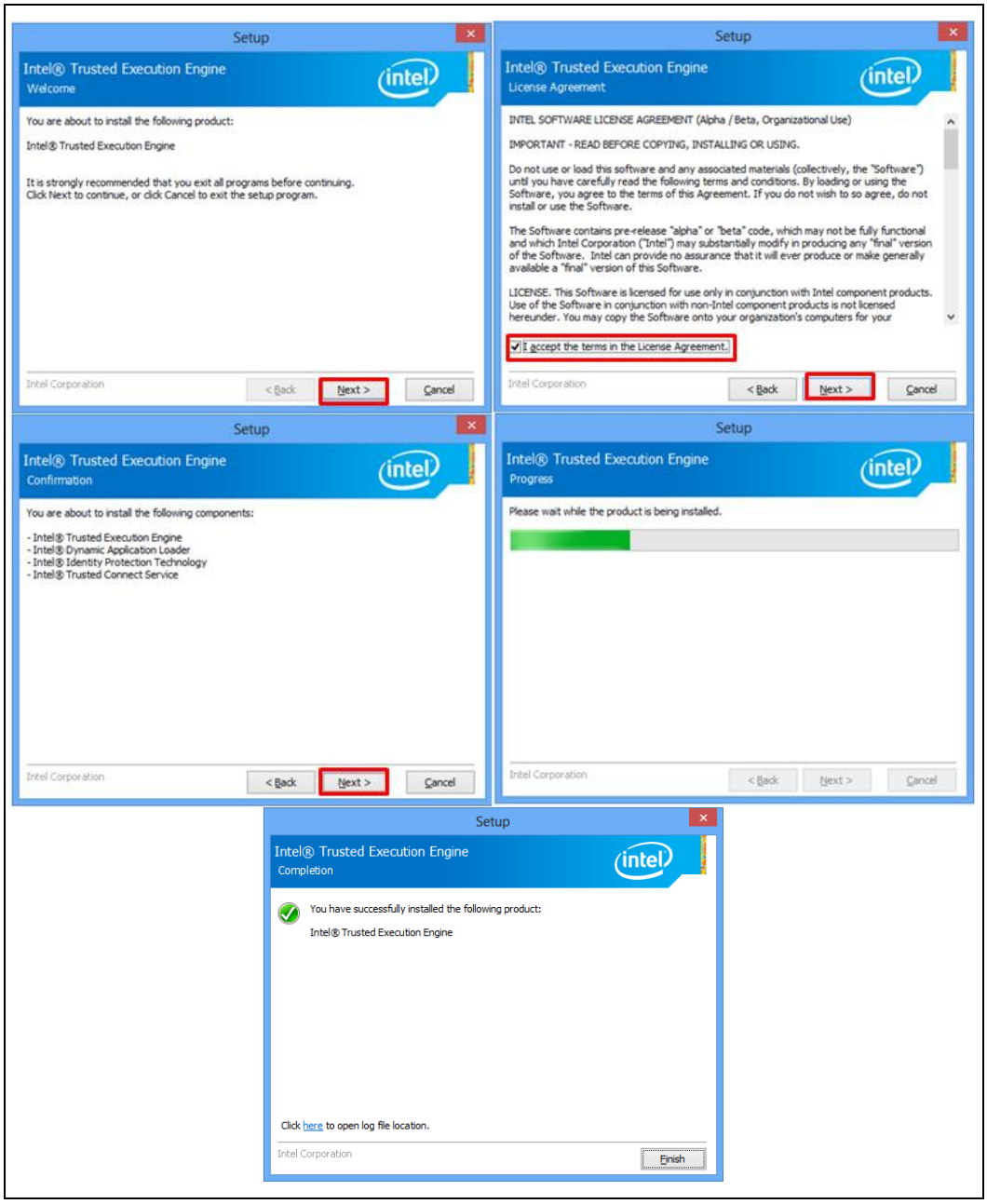
8 Intel® Trusted Execution Engine Interface (Intel® TXEI) Driver

Note: Installing Intel® TXEI driver is relevant only if you use Windows OS. For Android OS, TXEI driver is already part of OS image.

8.1 Install the Intel® TXEI Driver using Installer

1. Navigate to the root folder of the Intel TXE Installer (\\Installers)
2. Double click "SetupTXE.exe"
3. Follow the installation procedure as shown in Figure 27. Intel® TXE Installation Steps.

Figure 27. Intel® TXE Installation Steps



8.2 Install the Intel® TXEI Driver using Command Line

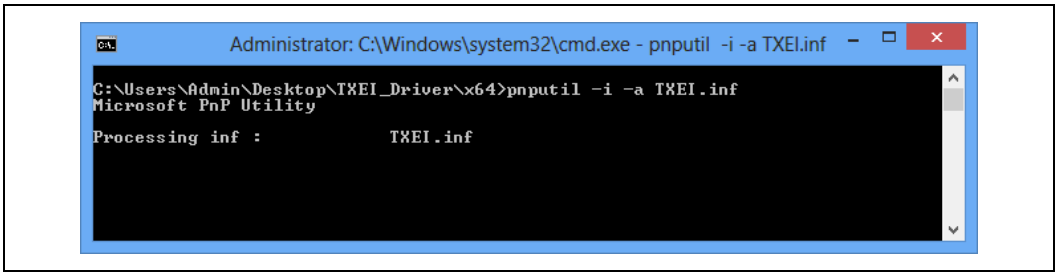
Please use this option as an alternative method. Installing the driver via Installer is the recommended method.

1. 1. Verify Intel TXEI driver is not installed on your system by:
   a. Open Device Manager (right click on My Computer -> Manage)
   b. Go to Device Manager subcategory
   c. Open System devices subcategory
   d. Look for device called "PCI Encryption/ Decryption Controller".

If "Intel® Trusted Execution Engine Interface" is already installed, uninstall it by right click->uninstall and check the "*Delete the driver software for this device*" checkbox.

2. Open command line with admin privileges and navigate to the root folder of TXEi.inf (\\TXEI_Driver\x64 or x86)
3. Type "pnputil.exe –i –a TXEI.inf

Figure 28. Intel® TXEI Installation



4. Select "*Install*"

Figure 29. Windows Security Prompt

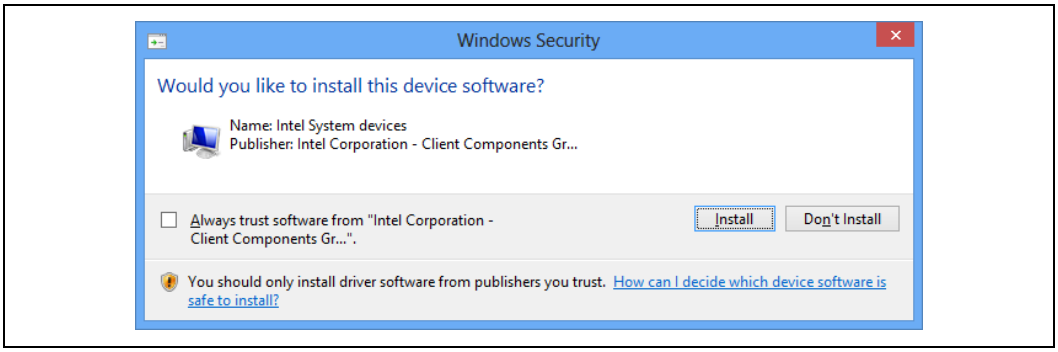
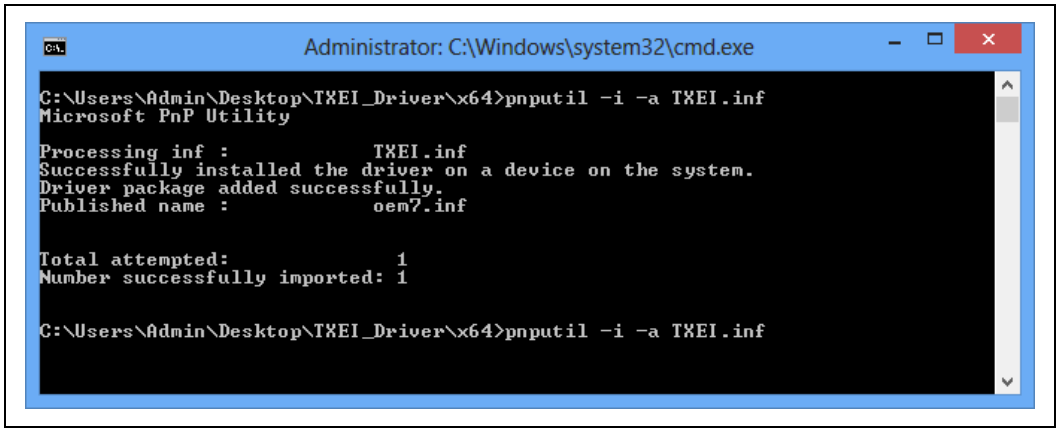
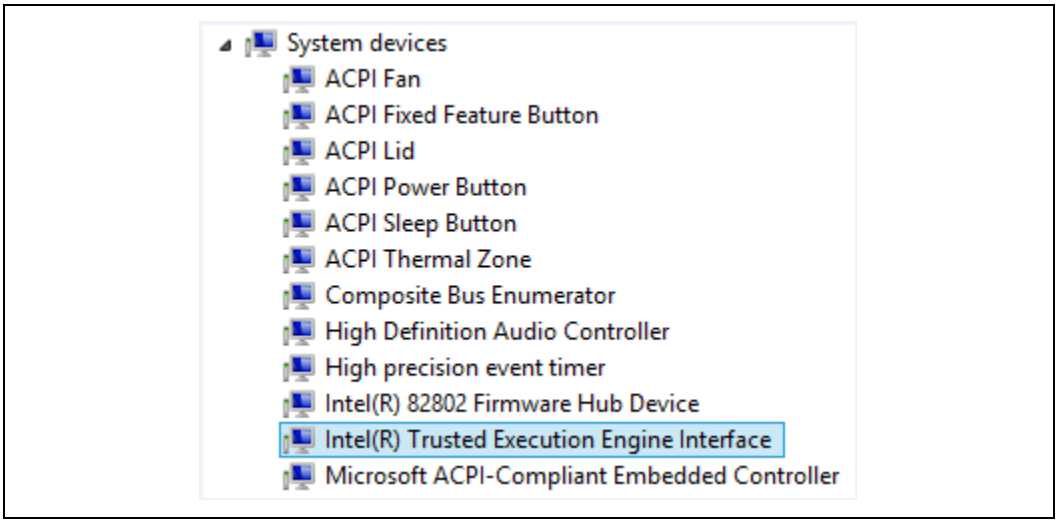


Figure 30. Finishing Intel® TXEI Installation



5. Verify Intel® TXEI is installed by referring to device manager→System devices:

Figure 31. Verify Intel® TXEI Installation in Device Manager



§

9 Using WinPE Tools

When using Windows FW tools in WinPE, please remember to load the TXEI driver at every boot.
This can be done by: *X:\Windows\System32>drvload.exe <path>\TXEI.inf.*
TXEI.inf can be found in every FW kit release.

10 Using EFI System Tools in UEFI Shell with UEFI Secure Boot Enabled Option

Due to Microsoft's mandatory UEFI Shells and related applications requirement (System.Fundamentals.Firmware.UEFISecureBoot), when running Intel or customer manufacturing utilities in UEFI shell, customer is required to disable UEFI Secure boot via BIOS setup menu or UEFI variable. If OEM/ODM wants to run specific EFI tool that need to run with UEFI secure boot, OEM/ODM will sign that EFI tool with their OEM key.

11 Intel TXEManuf

Intel TXEManuf tool will auto-detect the hardware/firmware SKU, and automatically runs tests to check functionality of their related features on the manufacturing line.

11.1 Prerequisites

Intel® TXEI driver must be installed. (Please refer to section Note: for instructions on how to install Intel® TXEI driver).

11.2 TXEManuf Usage

For detailed instructions please refer to “Intel TXEManuf” section in “System Tools User Guide” document, located at the System Tools folder.

§

12 Intel® TXE FW Update

Intel FWUpdate tool allows an end user, such as an IT administrator, to update Intel TXE FW without having to reprogram the entire flash device. It then verifies that the update was successful.

12.1 Prerequisites

Intel® TXEI driver must be installed. (Please refer to section 8 for instructions on how to install Intel® TXEI driver).

12.2 FWUpdate Usage

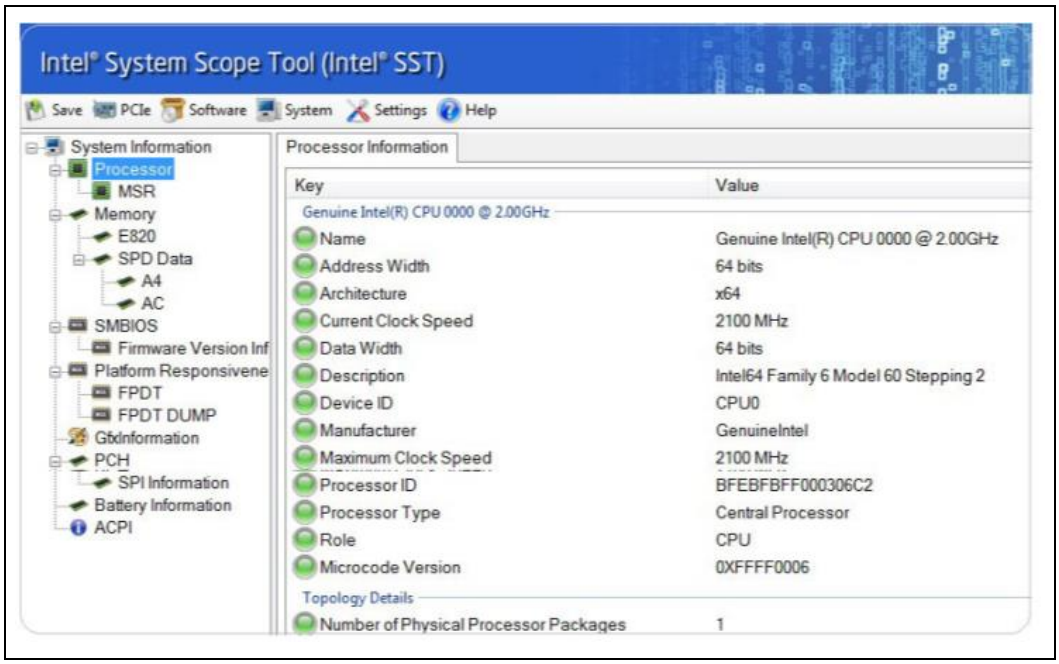
For detailed instructions please refer to “Intel TXE FW Update” section in “System Tools User Guide” document, located at the root folder.

§

13 Intel® System Scope Tool (Intel® SST)

The Intel® System Scope Tool (Intel® SST) is a tool which gives the complete snapshot of the system including both hardware and the software details.

Figure 32. Intel® System Scope Tool Screen Shot



- This tool is useful for providing full platform information for debugging purposes.
- An output file can be saved in .html format and attached to Bay Trail sightings
- This tool can be found on Intel® VIP inside the Bay Trail-M/D Compliance Kits (PV VIP kit # 54724).

14 FITC Soft Straps

Table 3. SOC Strap 0

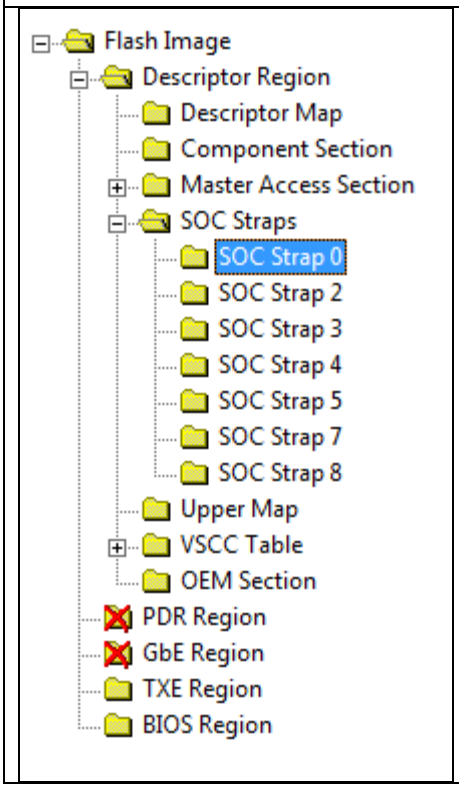
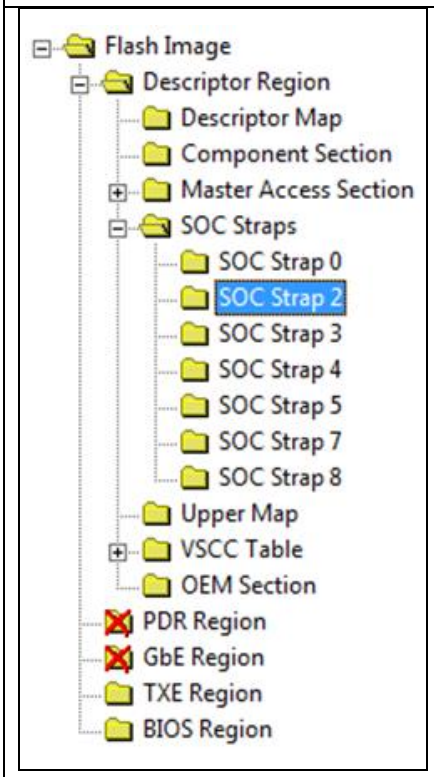
Location	Parameter	Values	Description
	BIOS Protected Range 4 Base	0x0000	Specifies the lower base of the BIOS protected range number 4. Address bits [11:0] are assumed to be 12'h000 for the base comparison. (goes to bits [12:0] at register: [Protected_Range_4] PR4 (@0x84)).
	BIOS Protected Range 4 Limit	0x0000	Specifies the upper limit of the BIOS protected range number 4. Address bits [11:0] are assumed to be 12'hFFF for the limit comparison. (Goes to bits [28:16] at register: [Protected_Range_4] PR4 (@0x84)).
	BIOS Protected Range 4 Write Protection Enable	True False (default)	When set, this bit indicates that the Base and Limit fields are valid and that writes directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared. Disabling this protected range could be done also by the security override pin strap. (this soft strap and the security override pin strap are reflected into bit 31 at register: [Protected_Range_4] PR4 (@0x84)).

Table 4. SOC Strap 2

Location	Parameter	Values	Description
	SPI Boot Block Size	00: 64KB (Default): Invert A16 if Top Swap is enabled. 01: 128KB: Invert A17 if Top Swap is enabled. 10: 256KB: Invert A18 if Top Swap is enabled	Sets SPI Boot Block Size
	SIO1_F0_Disable	False (default) True	Disable LPSS1 function 0 (DMA). false = enable. true = disable
	SIO1_F1_Disable	False (default) True	Disable LPSS1 function 1 (PWM#1). false = enable. true = disable
	SIO1_F2_Disable	False (default) True	Disable LPSS1 function 2 (PWM#2). false = enable. true = disable
	SIO1_F3_Disable	False (default) True	Disable LPSS1 function 3 (HSUART#1). false = enable. true = disable
	SIO1_F4_Disable	False (default) True	Disable LPSS1 function 4 (HSUART#2). false = enable. true = disable
	SIO1_F5_Disable	False (default) True	Disable LPSS1 function 5 (SPI). false = enable. true = disable
	SCC SDIO Disable	False (default) True	Disable SDIO. false = enable. true = disable
	SCC SDCARD Disable	False (default) True	Disable SDCARD. false = enable. true = disable
	HAD Disable	False (default) True	Disable HD Audio. false = enable. true = disable
	LPE Disable	False (default) True	Disable LPE. false = enable. true = disable
	XHCI Disable	False (default) True	Disable USH. false = enable. true = disable
	LAN Disable	True (default) False	Disable GbE. False = enable. true = disable
	SATA Disable	False (default) True	Disable SATA. False = enable. true = disable
	EHCI Disable	False (default) True	Disable USB: false = enable, true = disable

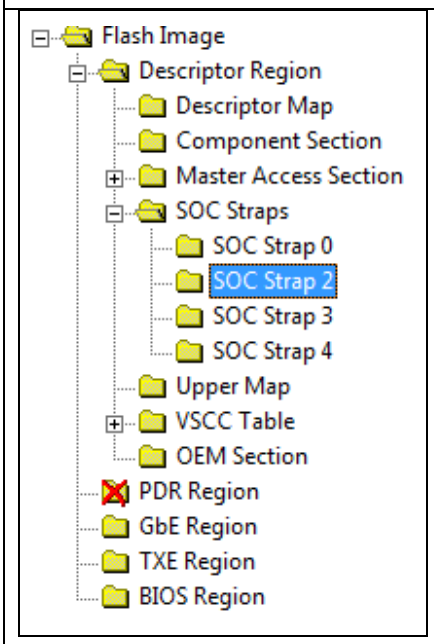
Location	Parameter	Values	Description
	PCIe 0 Disable	False (default) True	Disable PCIe port 0. False = enable. true = disable
	PCIe 1 Disable	False (default) True	Disable PCIe port 1. False = enable. true = disable
	PCIe 2 Disable	False (default) True	Disable PCIe port 2. False = enable. true = disable
	PCIe 3 Disable	False (default) True	Disable PCIe port 0. False = enable. true = disable
	SIO2 F0 Disable	False (default) True	Disable LPSS2 function 0 (I2C#0). false = enable. true = disable
	SIO2 F1 Disable	False (default) True	Disable LPSS2 function 1 (I2C#1). false = enable. true = disable
	SIO2 F2 Disable	False (default) True	Disable LPSS2 function 2 (I2C#2). false = enable. true = disable
	SIO2 F3 Disable	False (default) True	Disable LPSS2 function 3 (I2C#3). false = enable. true = disable
	SIO2 F4 Disable	False (default) True	Disable LPSS2 function 4 (I2C#4). false = enable. true = disable
	SIO2 F5 Disable	False (default) True	Disable LPSS2 function 5 (I2C#5). false = enable. true = disable

Table 5. SOC Strap 3

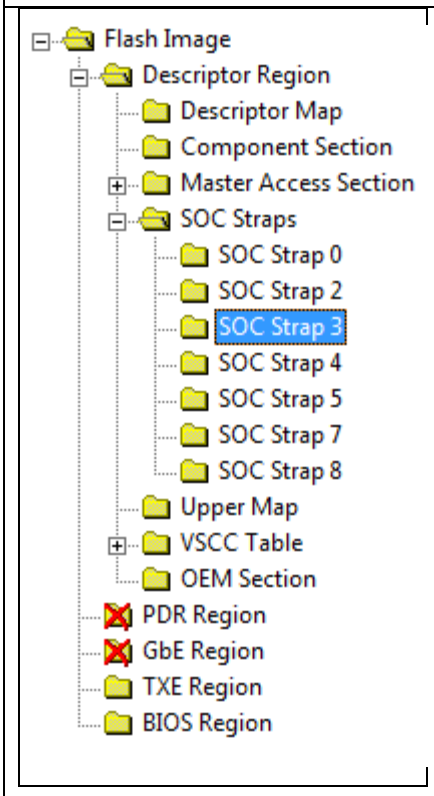
Location	Parameter	Values	Description
	SIO2 F6 Disable	False (default) True	Disable LPSS2 function 6 (I2C#6). false = enable. true = disable
	SIO2 F7 Disable	False (default) True	Disable LPSS2 function 7 (I2C#7). false = enable. true = disable
	DISBENDCLK_SSEN	False (default) True	Enable spread spectrum to DISPLAY BEND: false = disable, true = enable
	DISPSSCLK_SSEN	False (default) True	Enable spread spectrum to DISPLAY SS: false = disable, true = enable
	HFHPLLCLK_SSEN	False (default) True	Enable spread spectrum to HFHPLL: false = disable, true = enable
	PCIECLK_SSEN	False (default) True	Enable spread spectrum to PCIe: false = disable, true = enable
	SATACLK_SSEN	False (default) True	Enable spread spectrum to SATA: false = disable, true = enable
	OTG Super Speed PHY Disable	False (default) True	Disable OTG Super Speed PHY. false = enable. true = disable
	XHCI Super Speed PHY Disable	False (default) True	Disable USH Super Speed PHY. false = enable. true = disable
	Spread Percentage	0.456 (default) 0.439 0.423 0.391 and more…	Select Spread Spectrum Percentage (%). 0.456 is the Intel recommended value. Other values are for testing purposes only.

Table 6. SOC Strap 4

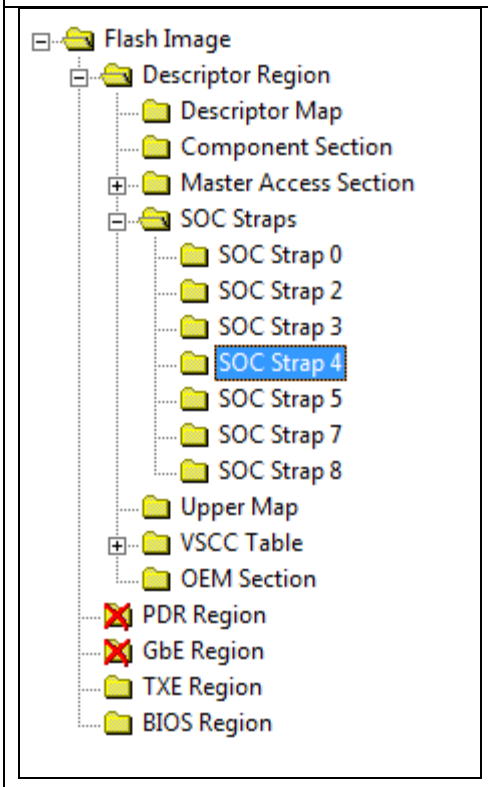
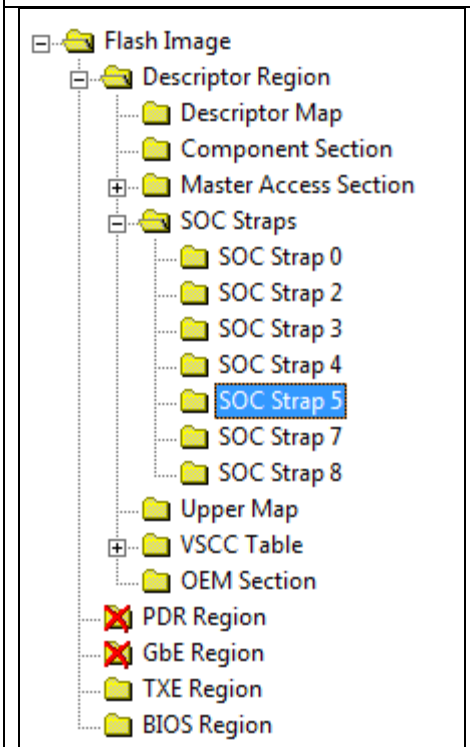
Location	Parameter	Values	Description
	LPC GPIO Select	1'b0: LPC (default) 1'b1: GPIO	Select the usage of LPC pins
	LPCCLK1_enb	True (default) False	False = disable. true = enable
	LPCCLK_SLC	1'b0 - iLPCCLK0 (default) 1'b1 - iLPCCLK1	Select LPC return clock source. This soft-strap is reflected to LPCC.LPCCLK_SLC register
	DELAY_PCIE_RLS	False (default) True Note: Not configurable strap. Value automatically set according to PCIECLK_SEEN value.	When set, PCIe reset release will be delayed after PMC patch load. Note: if PCIECLK_SEEN is enabled, DELAY_PCIE_RLS must be set as 1

Table 7. SOC Strap 5

Location	Parameter	Values	Description
	STGREN Register	1: Port Staggering Enabled (default) 0: No Port Staggering	Set the default for the PCI. Port Staggering Enabled: 0 = No Port Staggering, 1= Port Staggering Enabled. This strap sets the default value of the PCIe PORTSTAGEN register.
	Lane Reversal	0: No Lane Reversal (default) 1: Lane Reversal	Lane Reversal: 0 = No Lane Reversal, 1= Lane Reversal.
	Root Port Configuration	00: 4x1s Port 1 (x1), Port 2 (x1), Port 3 (x1), Port 4 (x1) (default) 01: 1x2, 2x1s Port 1 (x2), Port 3 (x1), Port 4 (x1) 10: 2x2 Port 1 (x2), Port 3 (x2) 11: 1x4 Port 1 (x4) **Note [1]**	Set the default value of root Port Configuration.

Note [1]: If the 'Root Port Configuration' default value is been changed in SOC Strap 5 the following changes need to be performed as well:

If the Value of "Root Port Configuration" been changed to "01: 1x2, 2 x1s Port 1 (x2), Port 3 (x1), Port 4 (x1)"
Require change of "PCIe 1 Disable" value in PCH Strap 2 from "false" to "true".

If the Value of "Root Port Configuration" been changed to "10: 2x2 Port 1 (x2), Port 3 (x2)",
Require change of "PCIe 1 Disable" and "PCIe 3 Disable" in PCH Strap 2 from "false" to "true".

If the Value of "Root Port Configuration" been changed to "11: 1x4 Port 1 (x4))",
Require change of "PCIe 1 Disable" and "PCIe 2 Disable" as well as "PCIe 3 Disable" in PCH Strap 2 from "false" to "true".

Table 8. SOC Strap 7

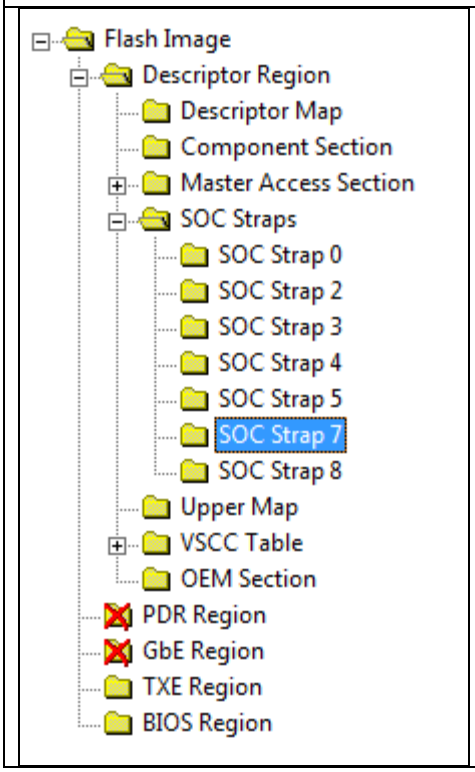
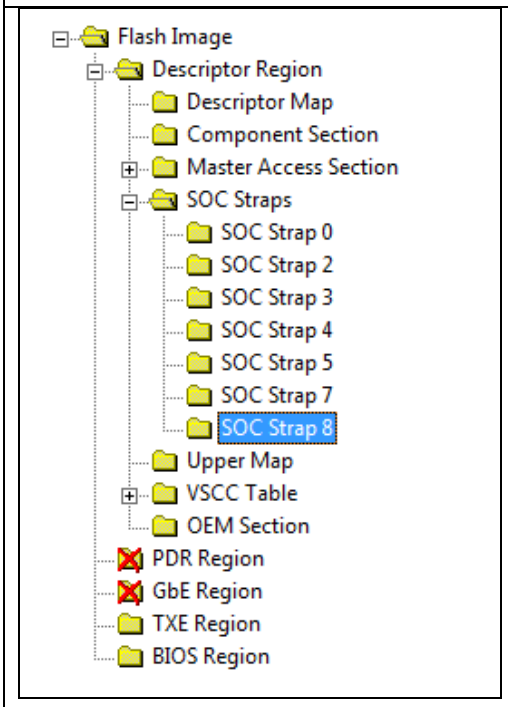
Location	Parameter	Values	Description
	PCIECMNLNPW Enable	True (default) False	False = disable, true = enable. **NOTE:** Integrated clock usage and specs must be signed off by SEG architects.
	PCIELANE0PWREN	True (default) False	PCIE Enables power for digital (synthesized) logic for the Common Lane logic and PLL1Core uPAR: false = disable, true = enable.
	PCIELANE1PWREN	True (default) False	PCIE Enables power for digital (synthesized) logic in the data lane to reduce leakage.
	PCIELANE2PWREN	True (default) False	PCIE Enables power for digital (synthesized) logic in the data lane to reduce leakage.
	PCIELANE3PWREN	True (default) False	PCIE Enables power for digital (synthesized) logic in the data lane to reduce leakage.

Table 9. SOC Strap 8

Location	Parameter	Values	Description
	SATACMNLNPW Enable	True (default) False	False = disable, true = enable. **NOTE:** Integrated clock usage and specs must be signed off by SEG architects.
	SATALANE0PWREN	True (default) False	SATA Enables power for digital (synthesized) logic for the Common Lane logic and PLL1Core uPAR: false = disable, true = enable.
	SATALANE1PWREN	True (default) False	SATA Enables power for digital (synthesized) logic in the data lane to reduce leakage.
	Satastatus2PMC	True (default) False	False = SATA to indicate D3 status to PMC. True = SATA to indicate DEVSLP status to PMC.

§

15 Appendix A: Google* Widevine* for Intel® TXE

15.1 Creating Widevine* CEK (Customer Encryption Key)

The CEK is responsible for encrypting Widevine Keybox in Android devices and not accessible by the host. The CEK is a global key used among the same models of devices for a single Customer.

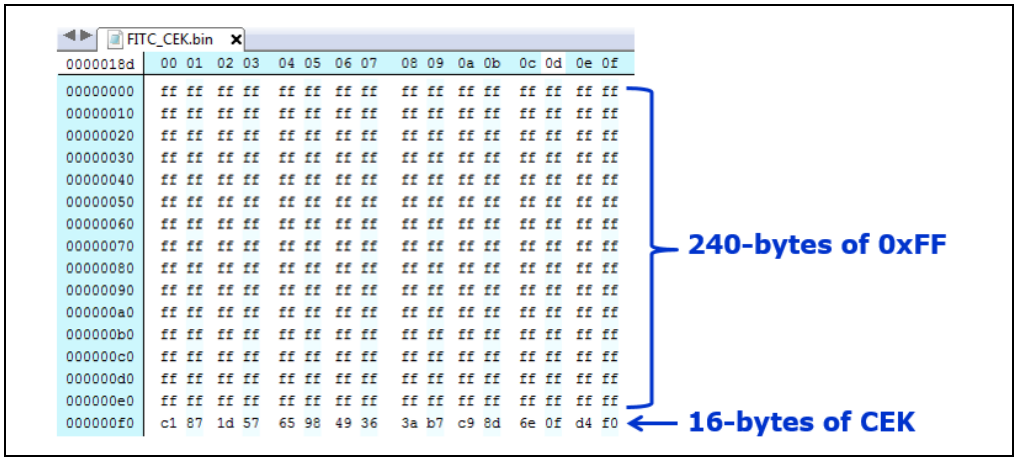
Note: The below key files are for demonstration purposes only and are not actual keys.

15.1.1 FITC CEK File Creation Procedure

15.1.1.1 Cleartext CEK

1. Generate a 16-byte Hex random number (unique per OEM) which is called CEK.
2. Combine 240 bytes of 0xFF (upper) with 16-byte CEK (lower) into 256 bytes FITC_CEK.bin
3. Insert FITC_CEK.bin into flash image with FITC tool (refer to section 3.4.1 steps 6 and 7).

Figure 33. FITC CEK File Map Example

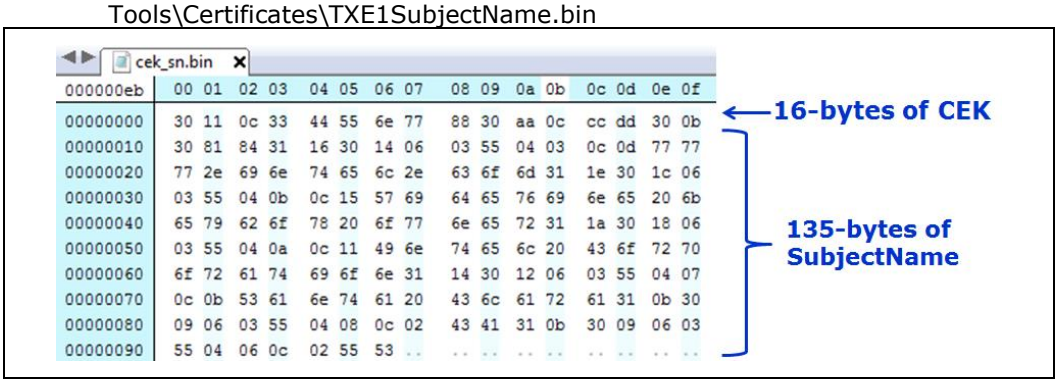


When building the image, add FITC_CEK.bin file (Flash Image -> TXE Region -> Configuration -> TXE -> CEK Configuration)

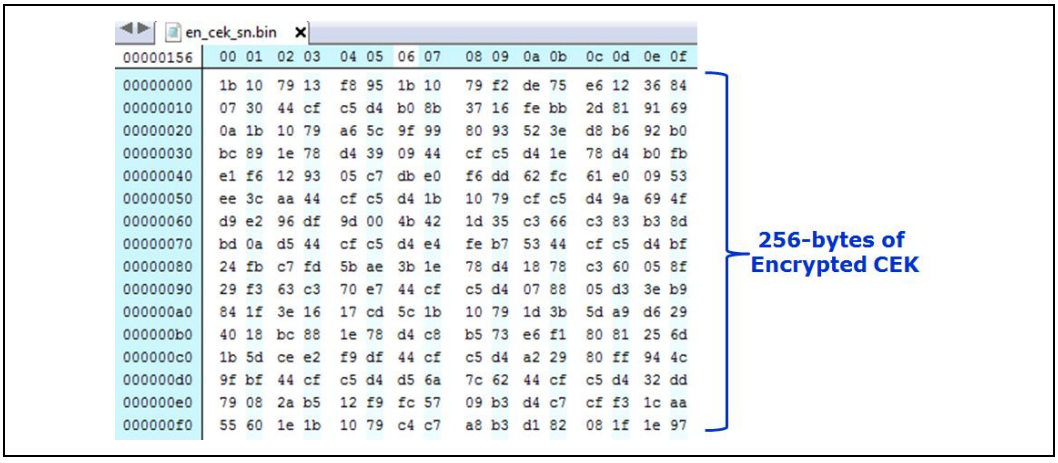
15.1.1.2 Ciphertext CEK

Note: Customer should scope relevant Google* documentation and decide for CEK insertion method. Intel recommended method is ciphertext

1. Generate a 16-byte Hex random number (unique per OEM) which is called CEK.
2. Create concatenated CEK||subjectname into one file (cek_sn.bin)
   a. SubjectName can be found under \\System Tools\Certificates\TXE1SubjectName.bin



3. Use openssl tool (open source tool) to convert the CEK certificate to *.pem format
   a. CEK certificate can be found under \\System Tools\Certificates\ TXE1DrmCekKeyProvPreProduction.cer
   b. Run openssl.exe x509 -inTXE1DrmCekKeyProvPreProduction.cer -inform DER -out <TXE1DrmCekKeyProvPreProduction_CEK.pem> -outform PEM
4. Encrypt CEK:
   a. Run: openssl.exe rsautl -encrypt -inkey <TXE1DrmCekKeyProvPreProduction_CEK.pem> -certin -pkcs -in cek_sn.bin -out en_cek_sn.bin



5. Insert en_cek_sn.bin into flash image with FITC tool (refer to section 3.4.1 steps 6 and 7)

15.2 Constructing Widevine* Provisioning KeyBox File

To support **Security Level 1** playback of protected content on Android devices, Widevine Keybox must be provisioned by Customer in factory. This keybox contains a device ID that is **unique** for each Android device and is the license that establishes a root of trust between Widevine DRM servers and the Android device.

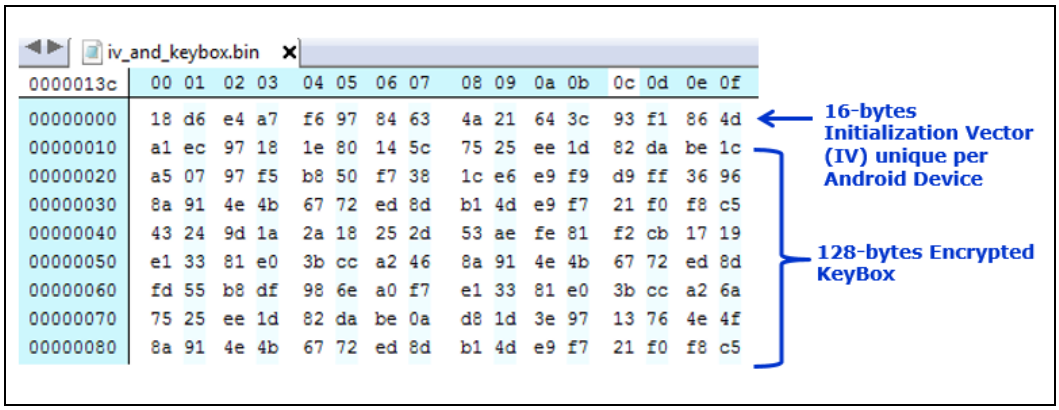
15.2.1 KeyBox Creation Procedure

1. Request KeyBox directly from Google.
2. Create appropriate unique 16-byte IV (Initialization Vector) for respective Android device.
3. Encrypt keybox by global CEK and IV via AES-CBC encryption.

   Example of encrypting KB with IV & CEK using AES-CBC:

   openssl aes-128-cbc -nopad -K <16-byte-CEK> -iv <16-byte-IV> -in GoogleKeyBox.bin -out EncryptedKB.bin
4. Write 16 bytes of IV, 128 bytes of encrypted keybox using FPT (refer to Widevine* KeyBox Provisioning Procedure, Chapter 7)

Figure 34. FPT KeyBox Provisioning File Map Example



§

16 Appendix B: Using Local Android Intel® TXE System Tools

16.1 Using Android System Tools

In order to use TXE System tools locally on SUT, you must push the tools using ADB (Android Debug Bridge) to a directory that can be accessed using Terminal Emulator OR using ABD itself.

16.2 Setup & Install ADB

Obtain ADB tool (can be attained from Android site). There are Linux and Windows versions of the tool. Usage of this tool will be the same with respect to TXE System tools.

16.3 How to Push & Use the Intel® TXE System Tools

1. Connect SUT to console
   a. Place both SUT & Console on same IP network
   b. Use "ADB Connect <IP_Address_Of_SUT>"
2. Push the TXE FW tools & their components, example below:
   a. \platform-tools>adb.exe push FPT /data/local
      1862 KB/s (357043 bytes in 0.187s)
   b. \platform-tools>adb.exe push TXEInfo /data/local
      1986 KB/s (222110 bytes in 0.109s)
   c. \platform-tools>adb.exe push TXEManuf /data/local
      1914 KB/s (305893 bytes in 0.156s)
   d. \platform-tools>adb.exe push TXEManuf.cfg /data/local
      377 KB/s (6023 bytes in 0.015s)
   e. \platform-tools>adb.exe push fparts.txt /data/local
      7 KB/s (8057 bytes in 1.000s)
   f. \platform-tools>adb.exe push vsccommn.bin /data/local
      133 KB/s (2132 bytes in 0.015s)
   g. \platform-tools>adb.exe push FpfConfigFile.txt /data/local
      26 KB/s (431 bytes in 0.015s)
3. Run the TXE System tools using ADB or local Terminal Emulator
   a. Using ABD Example:

- \platform-tools>adb.exe shell
- root@android:/ # su
- 127|root@android:/ # cd data/local
- root@android:/data/local # chmod 777 FPT TXEInfo TXEManuf
- Verify execution rights have been given via "ls -l"
- While in "data/local" directory run: "./TXEInfo" / "./FPT" "./TXEManuf"

b. Running local Terminal Emulator or Serial connection will be the same usage of the TXE System tools. For Serial connection on Intel CRB:

- Connect microUSB to Console COM port.
- Using Terminal client (i.e. PuTTY) configure connection to be serial with speed of 115200.
- When connection is successful, change to Android directory where tools have been pushed (i.e. /data/local per above example).

§