

# Raptor Lake OxM Debug Tokens Guide

*Revision 0.8*

*October 2021*

**Intel Confidential**



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

\*Other names and brands may be claimed as the property of others.

Copyright © 2021, Intel Corporation. All rights reserved.



# Contents

---

## Contents

Revision History .....	5
Terminology .....	6
1 Introduction .....	7
1.1 Overall Workflow .....	7
1.2 Choice of tools .....	7
1.2.1 Case 1: If Customer implements Intel® DnX .....	8
1.2.2 Case 2: If Customer hasn't implemented Intel® DnX .....	10
1.3 Usage of Tools .....	11
2 Overview of OxM Debug Tokens .....	12
2.1 Introduction .....	12
2.2 Preparing the Platform to Accept OxM Debug Tokens .....	12
2.2.1 High Level Process .....	13
2.2.2 Detailed process (EOM) .....	14
2.3 General Signing FAQs & Recommendations .....	16
2.4 <i>Token Features</i> .....	17
2.4.1 Part ID .....	17
2.4.2 Flags .....	17
2.4.3 Debug Features .....	18
3 Creation and Signing of OxM Debug Tokens .....	22
3.1 Introduction .....	22
3.2 Token creation .....	22
3.2.1 Token Creation - Intel® PFT GUI .....	22
3.2.2 Token Creation - CLI .....	30
3.3 Signing .....	31
3.3.1 Manifest header .....	31
3.3.2 Signing token .....	34
3.3.3 Intel® PFT GUI .....	34
3.3.4 Intel® PFT CLI .....	35
4 Token Injection .....	36
4.1 Introduction .....	36
4.2 Using Intel® DnX .....	36
4.2.1 Intel® PFT - GUI .....	36
4.2.2 Intel® PFT - CLI .....	37
4.3 NO Intel® DnX .....	38
4.3.1 Intel® FPT .....	38
4.3.2 Stitching a Token into the Firmware Image .....	38
5 Erasing the Token .....	39
5.1 Using Intel® DnX .....	39
5.1.1 Intel® PFT - GUI .....	39
5.1.2 Intel® PFT - CLI .....	40
5.2 NO Intel® DnX .....	40
5.2.1 Intel® FPT .....	40



	5.2.2	Re-flash a new image .....	40
6		Reading the Token .....	41
	6.1	Using Intel® DnX .....	41
	6.1.1	Intel® PFT – GUI .....	41
	6.1.2	Intel® PFT - CLI.....	42
7		Debugging OxM Debug Token Injection .....	43
8		References.....	44



## ***REVISION HISTORY***

---

<b>Revision Number</b>	<b>Description</b>	<b>Revision Date</b>
0.5	Initial release	May 2021
0.8	Updated revision to 0.8 for Alpha	October 2021



## TERMINOLOGY

Term	Description
Intel® DnX	Intel® Download and Execute
EOM	End of Manufacturing
Intel® FIT	Intel® Flash Image Tool
IBB	Initial Boot Block
IBBL	Initial Boot Block Loader
IFWI	Integrated Firmware Image
ISH	Integrated Sensor Hub
OBB	OEM Boot Block
SUT	System Under Test
OEM KM	OEM Key Manifest
Intel® MSU	Intel® Mobile Signing Utility
GUI	Graphic User Interface
CLI	Command Line Interface
SUT	System Under Test (or Debug)
OxM	Original (x) Manufacturer where “x” can be Equipment or Design



# 1 Introduction

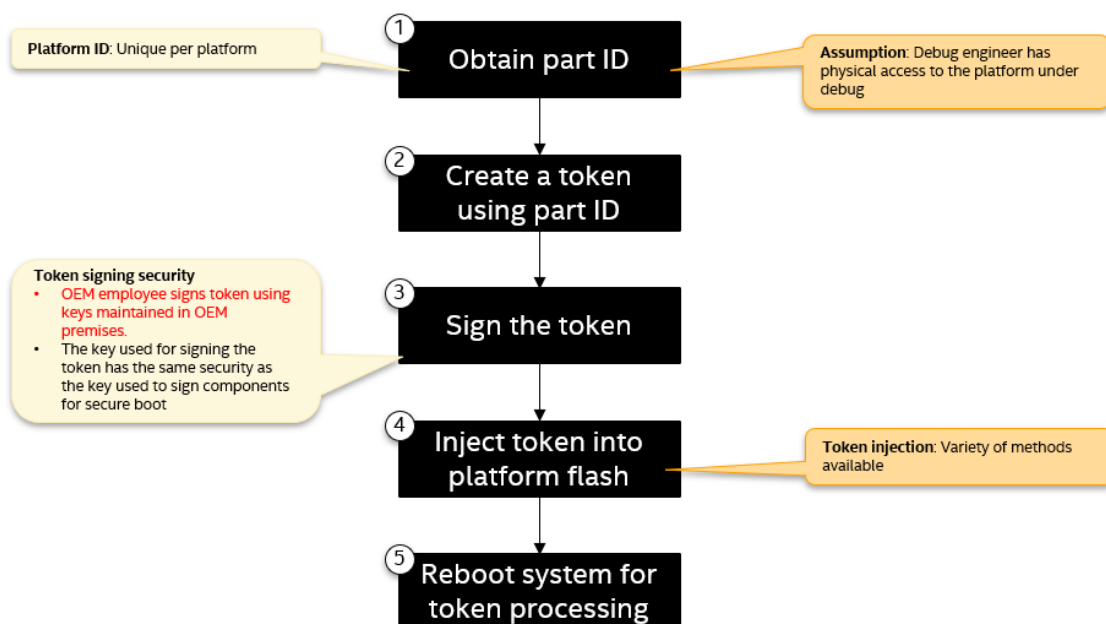
This document gives an overview of OxM Debug Tokens for Raptor Lake platforms.

The goal of this guide is to train the user to:

- Prepare the platform to work with OxM Debug Tokens
- Create OxM Debug Tokens
- Inject OxM Debug Tokens to the platform for debug
- Clear OxM Debug Token from platform after use.

## 1.1 Overall Workflow

Figure 1 shows the overall workflow to creating and injecting tokens into the platform.



**Figure 1**

Token is usually created for a specific platform to ensure the security of the platform. The details of each step in the workflow is covered in the rest of this document.

## 1.2 Choice of tools

The following tools are used within this document :

- Intel® Platform Plash Tool (Intel® PFT)
  - Read Part ID



- Create & Sign Token (sign using Intel® MSU)
- Inject or Erase Token
- Intel® Flash Programming Tool (Intel® FPT)
  - Read Part ID
  - Inject or Erase Token

Intel® PFT tool is used for creating tokens and is located in the CSME FW kit in the “Tools” -> “DnX Tools” folder.



Figure 2

Please also install the Intel® MSU for token signing on the host. All other tools mentioned above are also available in the kit.

### 1.2.1 Case 1: If Customer implements Intel® DnX

#### 1.2.1.1 Host and Target setup



Intel® Platform Flash Tool  
i.e. PFT

- Runs on Debug Host
- Read Part ID
- Create Token
- Inject token

Figure 3



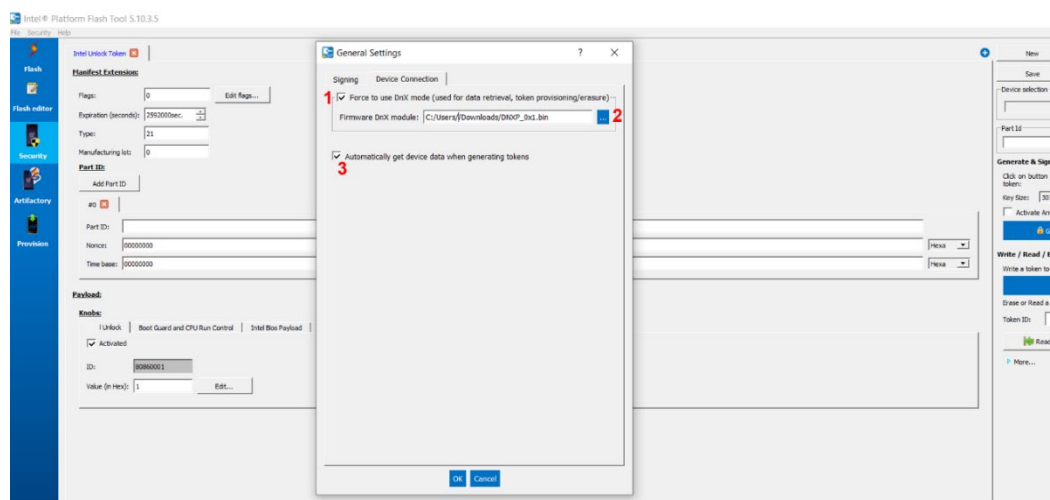


### 1.2.1.2 Intel® DnX module

Intel® DnX Module is a binary file signed by Intel. This file has the Intel® DnX logic that the Intel® CSE ROM will run. The file will be included in the Intel® CSME FW kit under the “dnx” folder.

This module should be linked to the Intel® PFT tool as shown below.

#### 1.2.1.2.1 GUI



**Figure 4**

Make sure the target system is in Intel® DnX mode by checking the “Device Manager” of the Host under “Universal Serial Bus device” should show Intel® DnX device

#### 1.2.1.2.2 CLI – Intel® DnX Firmware Downloader

Intel® DnX Firmware Downloader is the name of the executable that provides command line interface for Intel® DnX to interact with Intel® CSE firmware and perform different Intel® DnX operations. This is also installed by the Intel® PFT at the path –

“C:\Program Files (x86)\Intel\Platform Flash Tool”

Share View					
		This PC > OSDisk (C:) > Program Files (x86) > Intel > Platform Flash Tool			
		Name	Date modified	Type	Size
ss		Licenses	7/10/2019 2:38 PM	File folder	
3FS		modules	7/10/2019 2:38 PM	File folder	
ds		platforms	7/10/2019 2:38 PM	File folder	
ts		7z.exe	12/20/2018 12:47 ...	Application	574 KB
		adb.exe	4/17/2019 4:59 AM	Application	1,928 KB
		AutoUpdater.exe	7/10/2019 5:52 PM	Application	242 KB
		cflasher.exe	7/10/2019 5:58 PM	Application	6,258 KB
y Folders		dfu-util.exe	7/10/2019 5:52 PM	Application	196 KB
ken		dnxFwDownloader.exe	7/10/2019 5:52 PM	Application	98 KB
I		DownloadTool.exe	7/10/2019 5:51 PM	Application	828 KB
		event-uploader.exe	7/10/2019 5:52 PM	Application	157 KB

Figure 5

### 1.2.2 Case 2: If Customer hasn't implemented Intel® DnX

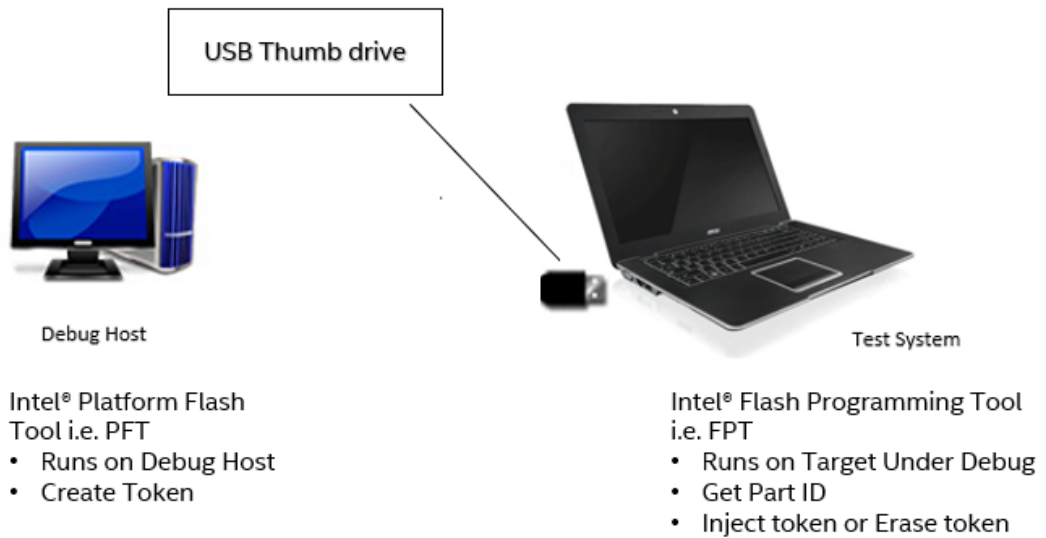


Figure 6

As seen, Intel® Platform Flash Tool is used for token creation, however Intel® Flash Programming Tool is used for Part ID and token Injection operations.

Details on Intel® DnX technology are covered extensively in the Intel® DnX User Guide. Briefly, from debug perspective, Intel® DnX technology is a closed chassis means to inject a OxM Debug Token which can re-enable debug features like the ITH logs for CSME or disabling Bootguard checks for debug.



### 1.3 Usage of Tools

OxM Debug Engineer's Host Machine	OxM's SUT	Used when?
Intel® Platform Flash Tool (PFT) Intel® Mobile Signing Utility (MSU) Intel® DnX module from CSME FW kit – Only if OxM platform is enabling DnX feature.	Intel® Flash Programming Tool (FPT)	R&D or Manufacturing debug Post EOM debug
Intel® MEU Tool	-	R&D or Manufacturing for OEM Key Manifest update R&D or Manufacturing debug Post EOM debug
-	Intel® FPT Intel® FIT	R&D or Manufacturing debug Post EOM debug



## 2 Overview of OxM Debug Tokens

---

### 2.1 Introduction

OxM Debug Tokens are used in Raptor Lake platforms to allow debug operations otherwise blocked after End-Of-Manufacturing.

The OxM Debug Token is authorized by OEMs to help them re-enable debug capabilities such as

1. **For OEM (starting Coffee Lake)**
  - OEM Unlock
    - Debug OEM signed components: ISH GDB
  - Disable OEM signed IP FW authentication during R&D
  - OEM BIOS Payload to pass BIOS settings to avoid re-flashing debug BIOS
  - Intel® BootGuard disable to debug OEM BIOS or debug non-booting systems
2. **Intel® CSME**
  - CSME ITH trace Enable/Disable for OEMs
3. **Enable Debug Interfaces (USB2.Dbc or BSSB 4 wire)**
  - Enable Closed chassis debug (MEEN) without changing IFWI
4. **CPU Run Control for debugging OS Applications**
5. **Enable DnX Flash Capabilities after EOM**

Tokens are digitally signed so that the target platform knows to accept them – thus they are secured and authorized. This chapter details the various details and is for understanding the tokens better. For tool usage, refer to Chapter 3.

### 2.2 Preparing the Platform to Accept OxM Debug Tokens

OxM Debug Tokens must be digitally signed, to ensure that the target platform will authorize them.

To “enable and use” OxM debug tokens on your platform is a 2-step process as shown below.

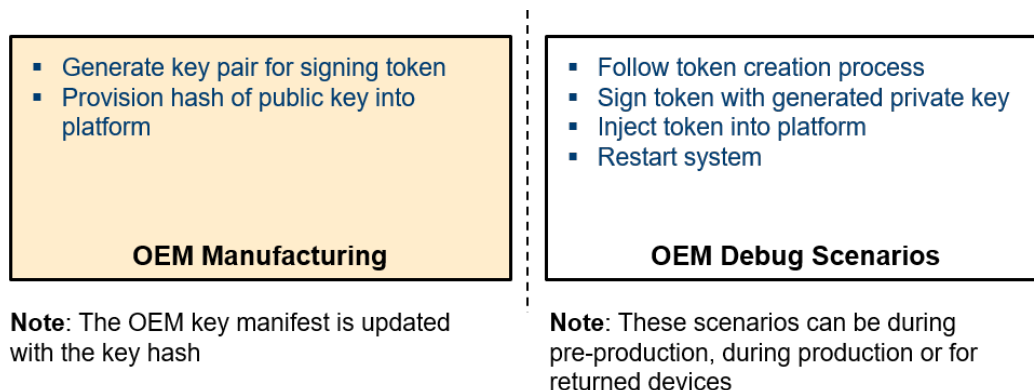


Figure 7

### 2.2.1 High Level Process

***Assumption:** At Manufacturing, the OEM is also signing other FW components like ISH or Audio and is aware of OEM Key Manifest structure.*

At Manufacturing, if the OEM does not wish to add any other components other than OxM debug token, they still need to create the OEM Key Manifest which is basically a structure to provide information about OEM signed components

High level flow is described below –

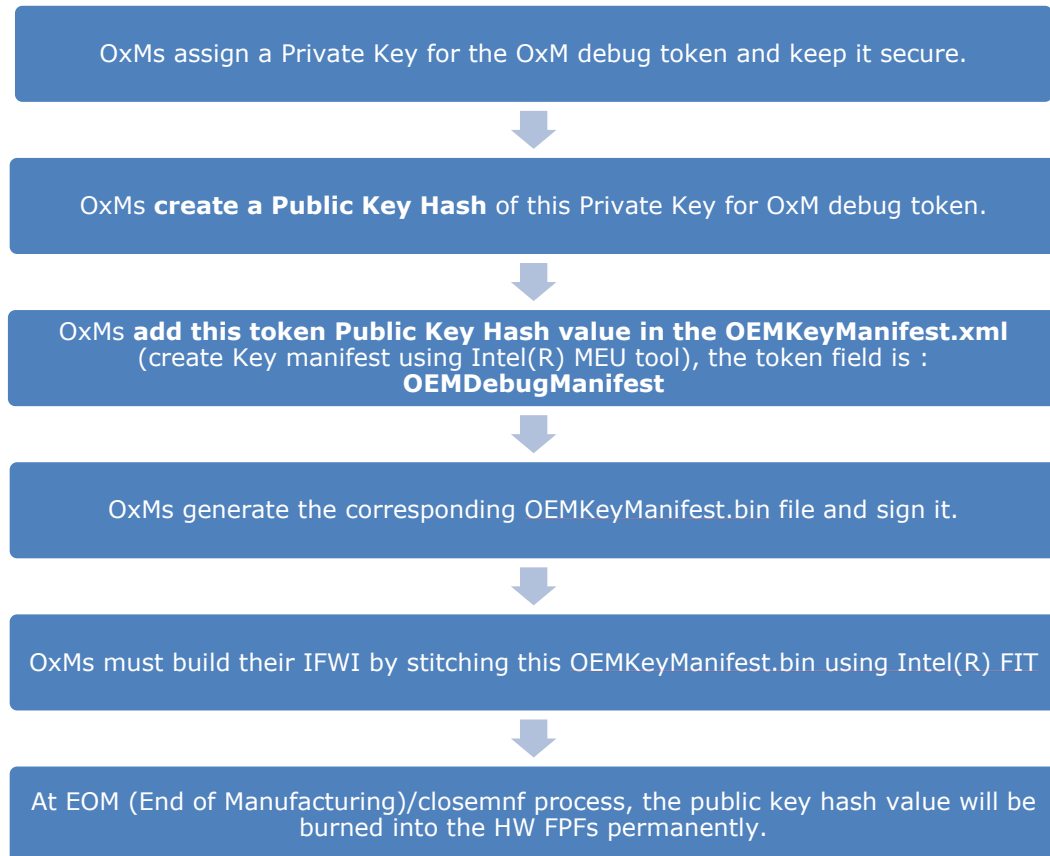


Figure 8

## 2.2.2 Detailed process (EOM)

**Note:** At the time of End-Of-Manufacturing(EOM), the Public key-hash of the Private key which will be used to sign OxM Debug tokens late is added to the OEM Key Manifest. Failure to do so will result in invalid tokens which cannot be used.

### 1. Using Intel® MEU tool, create OEMKeyManifest.xml

```
<VersionBuild value="0x0000" help_text="Indicates the build number in the version numbering" />
<KeyManifestEntries>
  <KeyManifestEntry>
    <Usage value="OemDebugManifest"
    value_list="BootPolicyManifest,,iUnitBootLoaderManifest,,iUnitMainFwManifest,,cAvsImage0Manifest,,cAvsIma
    <HashBinary value="C:\Users\... \WINDOWS\pub_hash.bin"
  </KeyManifestEntry>
</KeyManifestEntries>
'OEMKeyManifest>
```

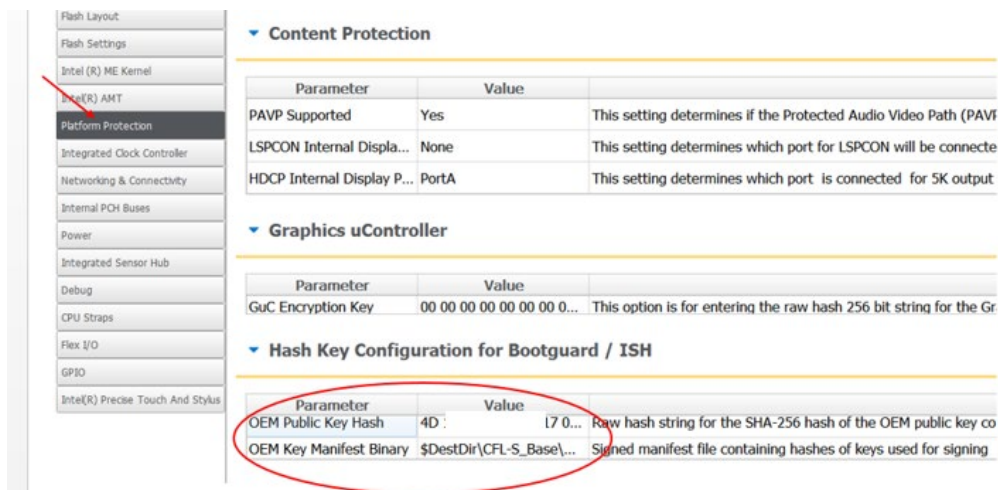
Figure 9



2. Add a KeyManifestEntry with value “**OEMDebugManifest**” for OEM token.
3. Point to the Public Key hash binary generated for signing the token.

*Note: This process is the same for adding any other OEM component to the manifest. For tokens, note the new string. The signing process in detail is captured in the Intel® Signing and Manifesting guide document in the FW kit.*

4. **Generate** the OEMKeyManifest.bin using the Intel® MEU
5. **Sign** the OEMKeyManifest.bin
6. Add this signed OEMKeyManifest.bin and its public key hash to the full image using Intel® FIT and **build the image**.



**Figure 10**

7. Finally, when EOM/closemanuf is done, the fuses will be burnt with the relevant key hashes.

*Note: As noted earlier OEM is responsible for the security of the key used to sign token. Intel recommends –*

- *Maintain security of key with same security level applied to Intel® BootGuard or secure boot keys*
- *Ensure only authorized personnel have access to key*
- *The token signing key is different from other keys used in the platform*



## 2.3 General Signing FAQs & Recommendations

### 1. Who owns the signing keys of OxM debug tokens ?

The Entity that owns burning Field Programmable Fuses also owns other OxM component keys like ISH/Audio and OxM debug tokens.

This can be the OEM or the ODM depending on the customer work model.

### 2. Can the keys used for OxM debug tokens be same as other components or OEM Key Manifest?

The key ownership is entirely with the customers and they can choose to use the same key or different keys. It is up to the customer as debug tokens are now an OxM component. Intel recommends using different keys for different components as mentioned earlier in this document.

### 3. How can ODM debug engineer request token to sign? How long would it take to get a debug token signed?

This depends on the OEM-ODM secure work model. The OxM owns having a secure and trusted mechanism for an authorized OxM debug engineer to request to sign an OxM debug token.

### 4. What if OxM doesn't put the debug tokens public key hash in the OEM Key Manifest during Manufacturing or R&D ?

The above scenarios are quite possible. OxM can add the OxM Debug public key hash later, **if and only if** an OEM Key manifest was present at the time of EOM. In short, no OEM Key Manifest means no capability to add any entry to the OEM Key Manifest later.

*Note : Detailed information on OEM signing is captured in the Intel® Signing and Manifesting guide, section 3.5*





## 2.4 Token Features

This section describes each of the Token's properties and serves as a reading material. For tool related features, go to Chapter 3.

### 2.4.1 Part ID

**After End-Of-Manufacturing**, if the OxM debug engineer wants to debug a specific platform, they need to obtain the Part ID of this system. A token created for this system cannot be used on another system. Thus, a Part ID is the basic requirement for creating an OxM debug token. Since the Part ID is read from the system under debug, the OxM debug engineer should have physical access to the SUT.

**During Manufacturing**, the OxM debug engineer can also use the OxM debug token to enable debug features. This type of token need not be tied to a specific Part ID and can be valid on multiple systems.

*Note: The token is created and signed by using OxM's secure signing mechanism by an authorized OxM debug engineer. If there is a signature mismatch, the debug token is simply invalid, it is ignored, and the platform remains in its original state before injecting the token.*

### 2.4.2 Flags

A OxM Debug Token can be "fine-tuned" with certain properties depending on the use-case of the OxM debug engineer. This section explores the main options.

#### 2.4.2.1 Expiration

An OxM debug engineer can time the token to expire after "x" number of seconds. This feature is useful when the OxM debug engineer wants to secure the amount of time an OxM debug token is active.

On the other hand, the OxM debug engineer can also create an OxM debug token that does not expire at all. This type of feature is useful for long running stress tests or validation cycles where an issue can happen after uncertain time.

If an OxM debug token with "No Expiration" and necessary debug features has been stitched in the image that's running in such a use case, then the token will be available forever.

No expiry = forever token. The OxM debug engineer should consider erasing the token as soon as debug is over.



#### 2.4.2.2 Anti-Replay

This feature allows the OxM debug engineer to protect from the re-use the same token on the same SUT.

With Anti-Replay enabled, the OxM Debug Token will be invalidated when a Part ID is read again or the system has been re-flashed, CMOS cleared or cold reset.

For best security, enable Anti-Replay and set the appropriate Expiration time.

#### 2.4.2.3 Globally valid

As mentioned earlier, during Manufacturing, an OxM debug engineer can create, sign, and inject a OxM Debug Token that is valid on multiple parts and is not tied to single platform.

By enabling the Globally valid feature of the OxM Debug Token, this can be achieved.

Note that, once End-Of-Manufacturing/closemfn is done, a globally valid token is no longer effective.

### 2.4.3 Debug Features

There are various debug features that can be enabled or disabled using an OxM debug token. This section explores those options one by one.

#### 2.4.3.1 OEM Unlock

On production systems, this feature allows the OxM to “unlock” their IPs for conducting their debug. For example, PSE IP which is customized by the OxM’s, can be unlocked for trace and run control purposes.

Since the OxM debug token is signed by an authorized OxM debug engineer using secure signing mechanism, only authorized OxM debug users will have access to this operation. If the token verification fails, the system will continue to stay in its original error state.

#### 2.4.3.2 Disable Bootguard and enable CPU probe mode

The specific feature allows the OxM debug engineers to temporarily disable Bootguard features and boot the failing system for conducting debug.

There are four fine tuning options available –

1. BootGuardDisabled
2. BootGuardNoEnforcement:
3. BootGuardNoTimeouts
4. BootGuardnoEnforcementAndTimeouts



Subsequently, it also allows enabling CPU Probe mode for run control debug purposes. This is equivalent to setting Disable CPU Debug (DCD) to 0.

Since the OxM debug token is signed by an authorized OxM debug engineer using secure signing mechanism, only authorized OxM debug users will have access to this operation. If the token verification fails, the system will continue to stay in its original error state.

#### 2.4.3.3 Enable Tracing

The OxM debug token can also enable “CSE Tracing” allowing the collection of ITH debug logs by the OxM debug engineer.

##### **Important Note:**

**Starting Tiger Lake, CSE ITH logs can ONLY be re-enabled on a production platform ( after End-Of-Manufacturing/closemanuf) using a debug token due to security hardening of the CSME IP.**

If the OxM has enabled OxM debug token feature on their platforms, this option will allow them to collect ITH traces and attach to relevant CSE sightings or do initial triage on their own.

If the OxM has NOT enabled the OxM debug token feature on their platforms, they should contact their Intel debug representative for helping with ITH log collection.

OxM ITH logs are also now improved to be more intuitive for the OxM debug engineers to do initial debug triage and help isolate issues faster. Trainings are available on Intel’s customer training portal, please check with your Intel Customer Enablement representative for more details.

#### 2.4.3.4 OEM BIOS payload

The OxM debug token binary can carry a payload to the OEM BIOS allowing the OxM debug engineer to configure these settings without having to re-build and re-flash a debug BIOS.

For example, if the OxM debug engineer wants to unhide an usually hidden OxM BIOS menu, the OxM debug engineer can select that in the OxM debug token.

Another example is enabling debug transports like DCI.OOB for DCI.OOB.USB2/3.Dbc or DCI.OOB.BSSB-2 wire without re-building and re-flashing of the image on the SUT.

At the time of writing this document, the below options are supported using the 32-bit OEM BIOS payload using various bit combinations. Detailed bit-wise description can be found in the Intel® BIOS Writer’s Guide.



Option name	Description
ExposeDebugMenu [Bit 0]	Expose debug options in production BIOS menu. If debug options menu is exposed, this setting does nothing.
StreamingTraceSink [Bit 1:4]	Enable Debug Transports specifying Power Controls  Directly maps to BIOS Debug Menu choices in Intel Reference Code
NpkPolicy [Bit 5]	Enable NPK
TraceEnable [Bit 6]	Enable BIOS traces  Note : At the time of writing this document, this bit is not supported. Please check an updated version of this document.
JTAGC10PGDisable [Bit 7]	Disable JTAG C10 power gate
USBOverCurrentOverride [Bit 8]	Override USB OC configuration to allow VISA
Intel Reserved [Bit 9:15]	Future Intel Use
OEM Available [Bit 16:31]	OEM's can use these bits as desired

#### 2.4.3.5 Enable Debug Interface

This feature of the OxM debug token allows the authorized OxM debug engineer to re-enable DCI.OOB.USB2.Dbc on the production locked platform without having to re-build and re-flash the image on the SUT.

The assumption is that the OxM will not clear the USB2DbcEn Bit in their BIOS per Intel's recommendation.

#### 2.4.3.6 Cancel OEM Authentication

This feature of the OxM debug token allows the OxM debug engineer to use "debug signed" components on the platform instead of "production signed" only for debug purposes (except BootGuard Key manifest and OEM Key manifest).



#### 2.4.3.7 DnX capabilities

If the OxM chooses to enable the Intel® DnX feature on their production (End-Of-Manufacturing/closemanuf) platforms for Image recovery and re-flashing purpose. Note, only OxM debug token operations are available after End-Of-Manufacturing.

Please refer to Intel® DnX User guide from the CSME kit for full details.

#### 2.4.3.8 ISH GDB debug

This feature allows an authorized OxM debug engineer to do source level debug for the OxM's ISH component. The possibilities of this option are currently being explored.

*Note: If the OxM has specific requirements they would like to request, please contact your Intel® Customer enablement representative.*



## 3 Creation and Signing of OxM Debug Tokens

---

### 3.1 Introduction

In this section we discuss the other side of the Token Process Lifecycle, that is, actual debug by the OxM's debug Engineer.

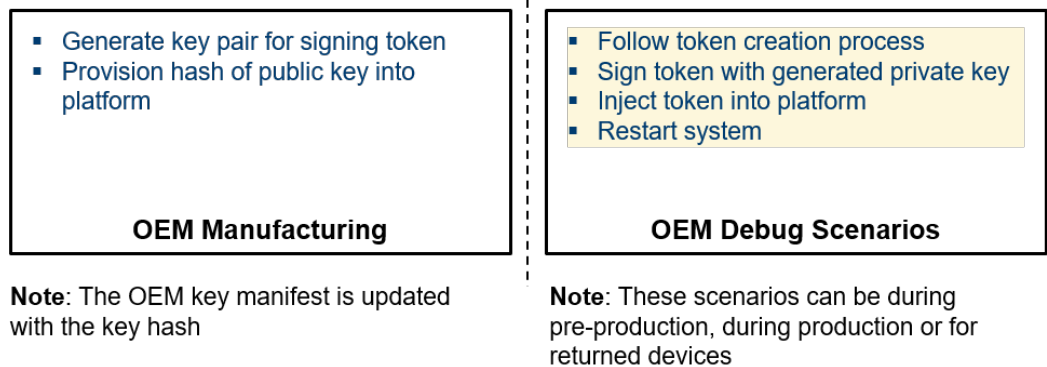


Figure 11

#### Assumptions:

- The OxM debug engineer has physical access to the SUT.
- The OxM debug engineer is authorized by the OxM to sign the OxM debug token OR is authorized by the OxM to request to sign the OxM debug token.

### 3.2 Token creation

Intel® PFT allows to generate and sign the token on the click of a button which includes the use of template for token creation.

#### 3.2.1 Token Creation - Intel® PFT GUI

As shown in Figure 12, using the Intel® PFT, Click on the “**New**” button, and then select the target platform, and OEM Unlock Token as the token template for an OEM Unlock Token.

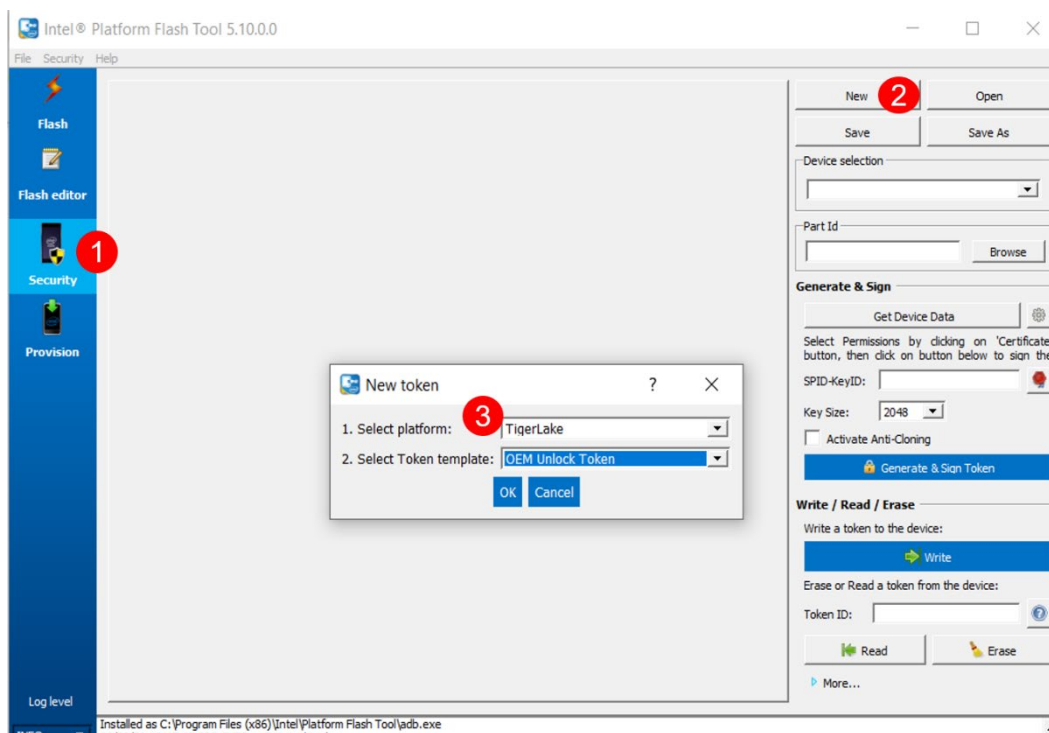


Figure 12

### 3.2.1.1 Configuring the settings

There are multiple options that can now be set for the token. Leave all of them with defaults, **except** for the following:

#### 3.2.1.1.1 Expiration

Set the time (in Seconds) you'd like the token to be valid. Default in the tool is 3 days.

### 3.2.1.1.2 Flags

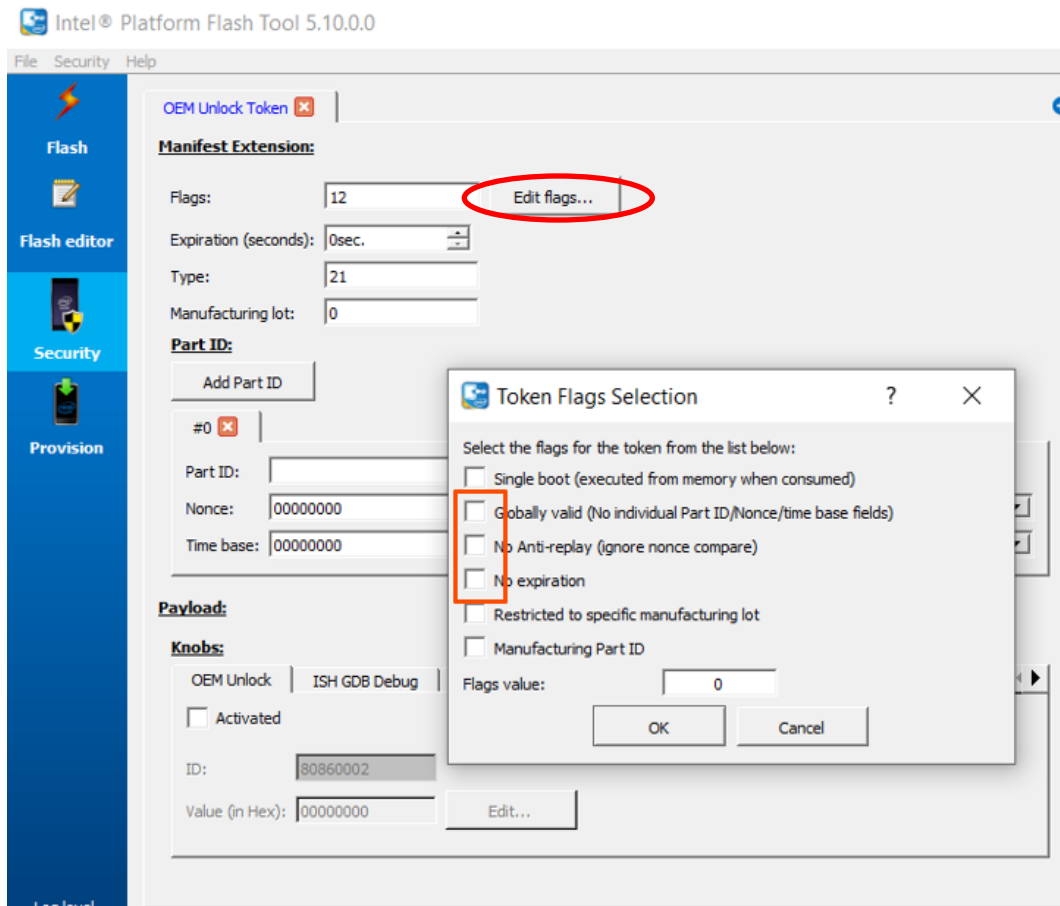


Figure 13

In the “Flags” section you can select either -

- **Globally valid (pre-EOM only):** This means that the token can be used on any platform whose Public key hash matches that of the Private key of the token and is not tied to a particular platform ID. Useful for pre-EOM debugging only.

*Note: Once platform is post End of Manufacturing only part-specific token is accepted i.e., only token with Part ID (PID) will work.*

- **No Anti-replay:** Anti-Replay protection stops a token being re-used on the same device after token has been cleared either by clearing RTC or re-flashing Image. Selecting “No Anti Replay” allows the debugger to ignore nonce and replay the token.  
This option is only relevant for tokens tied to a particular platform ID.
- **No expiration:** This means that the token has no time limit.

*Note: It is recommended to use to token with time expiration and Anti-replay flag.*





### 3.2.1.1.3 Knobs

The Intel® Platform Flash Tool provides debug features as “Knobs” for the token. These define what the token allows/disables on the platform.

You can check/uncheck the checkbox inside each tab to add the knob to the token, and then edit the value (automatically) of the token by clicking the Edit button and selecting from the radio buttons inside. The knobs available vary depending on the token being created.

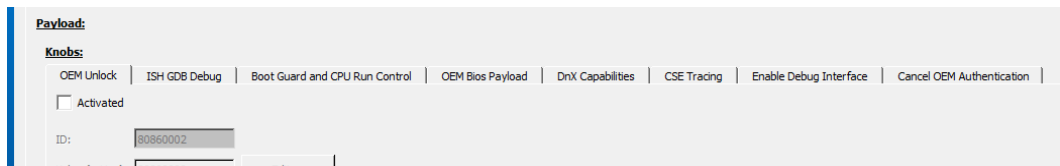


Figure 14

Knob	Meaning
<b>OEM Unlock</b>	<ol style="list-style-type: none"> <li>1. Allow an OEM to</li> <li>2. Enable/Disable Unlock of OEM IPs</li> <li>3. Enable DAM</li> </ol> <p>See Figure 15</p>
<b>ISH GDB Debug</b>	Enable ISH GDB support. Please check with you Intel® ISH Customer Enablement Representative for this
<b>BootGuard and CPU Run Control</b>	<p>Used for platforms that have Intel® Boot Guard enabled in IFP fuse at the EOM. Allows to disable Intel® BootGuard for debug purposes and has the following options -</p> <p><b>BootGuardDisabled:</b> Disable boot block verification process completely and allow platform to boot.</p> <p><b>BootGuardNoEnforcement:</b> Disable Enforcement policy of boot block verification process completely and allow platform to boot</p> <p><b>BootGuardNoTimeouts:</b> Disable Timeouts due to Intel® Boot Guard failures allowing to debug :</p> <ol style="list-style-type: none"> <li>1. ACM or BIOS related flows.</li> <li>2. If the problem is the timer duration being too short or a problem with ACM or BIOS itself</li> </ol> <p><b>BootGuardnoEnforcementAndTimeouts:</b> This is a combination of the two features above.</p>



Knob	Meaning
	This option also clears Disable CPU Debug (DCD override to 0) allowing for CPU Run control.
<b>OEMBiosPayload</b>	<p>OEM can pass 32-bit data to their BIOS. The tool will allow you to enable these features via checkboxes to enable BIOS features like – Display hidden menu, Enable ITH, Enable Debug interface and so on. Refer to the Intel® BIOS Writers Guide for full information.</p> <p>See Figure 16.</p> <p><i>Note : Bit 6 is not supported at the time of writing this document. Please check the newer version of the document, when available.</i></p>
<b>DnXCapabilities</b>	Re-enable Intel® DnX image recovery capabilities after EOM. Please refer to the Intel® DnX User Guide for full information.
<b>Cancel OEMAuthentication</b>	Cancel OEM IP signing verification to conduct more easy R&D of their FW When this option is enabled in an OxM signed debug token, it disables the FW authentication for ISH, CAVs and IPU FW.

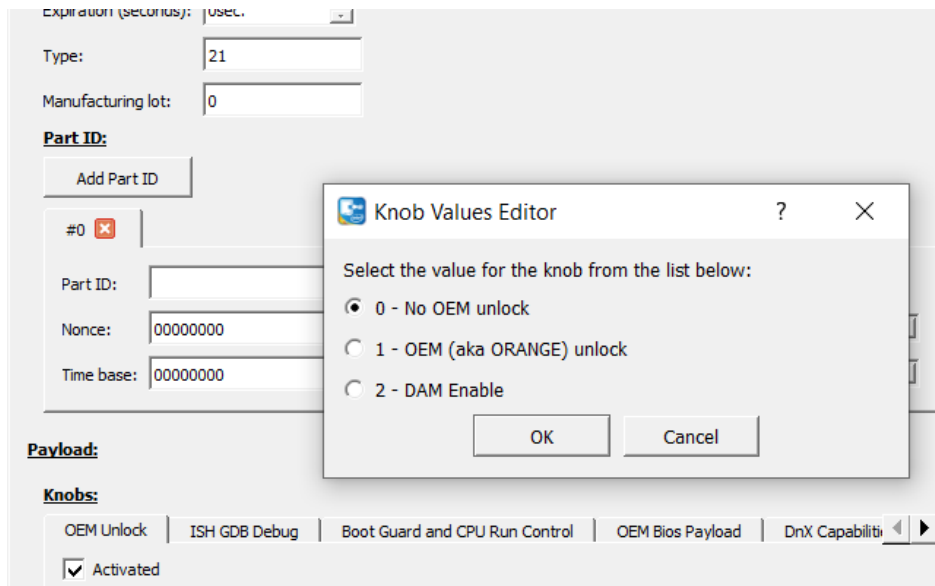


Figure 15

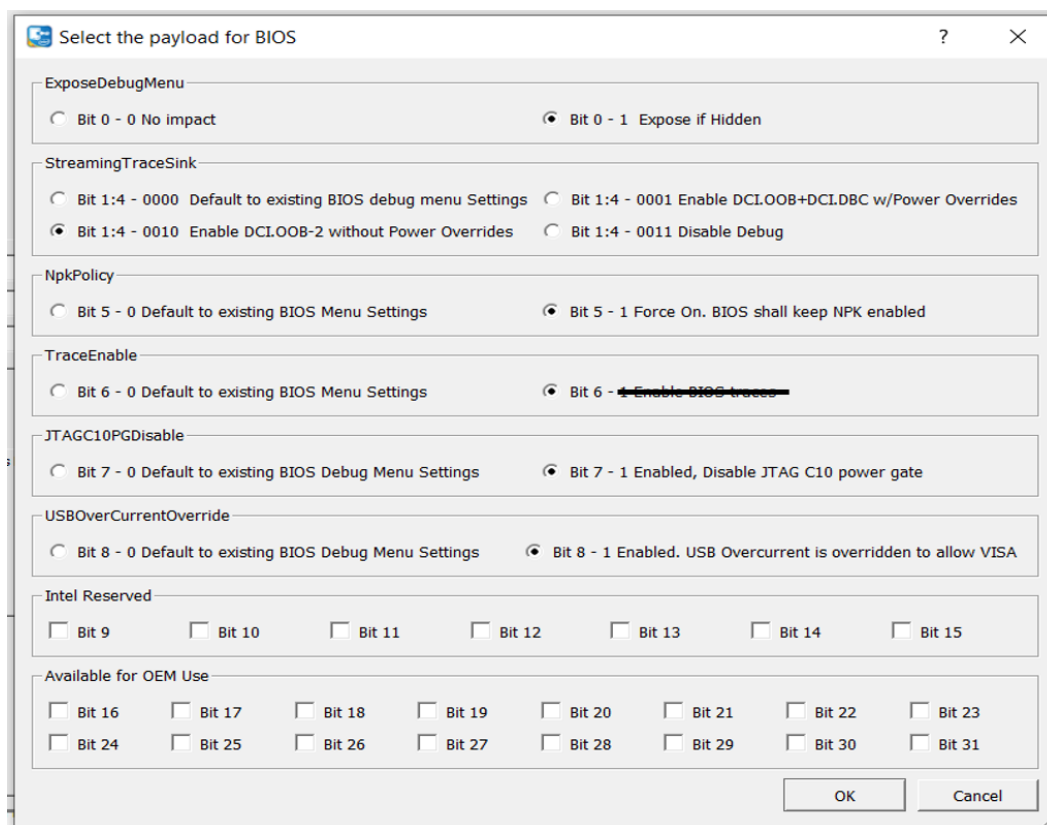


Figure 16

### 3.2.1.2 Getting the Target's Part ID

Part ID is only relevant on Post-EOM systems.

#### 3.2.1.2.1 Using Intel® DnX - GUI

Check section 1.2 for Intel® DnX setup. For Part ID, click on “Get Device Data” as shown below.

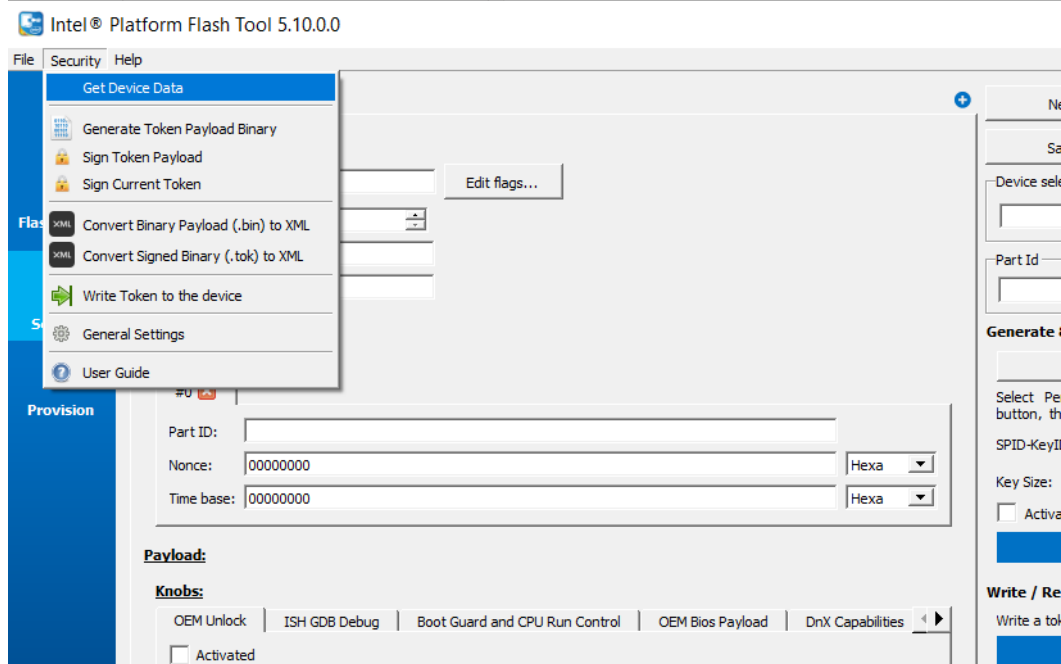


Figure 17

### 3.2.1.2.2 Using Intel® DnX - CLI

Check section 1.2 for Intel® DnX setup. Launch a Command prompt to execute “dnxFwDownloader.exe”

```
dnxFwDownloader.exe --command gettokenpid --fw_dnx  
DNXP_0x1.bin --flags 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary DnXP_0x1.bin from CSME Firmware Kit
--flags	Slot number for anti-replay protection of corresponding token:  0: No AR protection needed. Nonce is stored in the temp storage in SRAM  1: Nonce generated is stored in first Nonce slot  These are described in section 3.3.1.3



### 3.2.1.2.3 NO Intel® DnX

Check Section 1.2 for the proper set up.

Launch the Intel® FPT tool runs on the target being debugged.

Operation	Command Line
Get Part ID of the target platform	FPT.exe -GETPID <file>

This will retrieve the part ID into a file.

Next, Copy this file to a storage device. Launch Intel® PFT on the Host and click on “Browse” button as shown

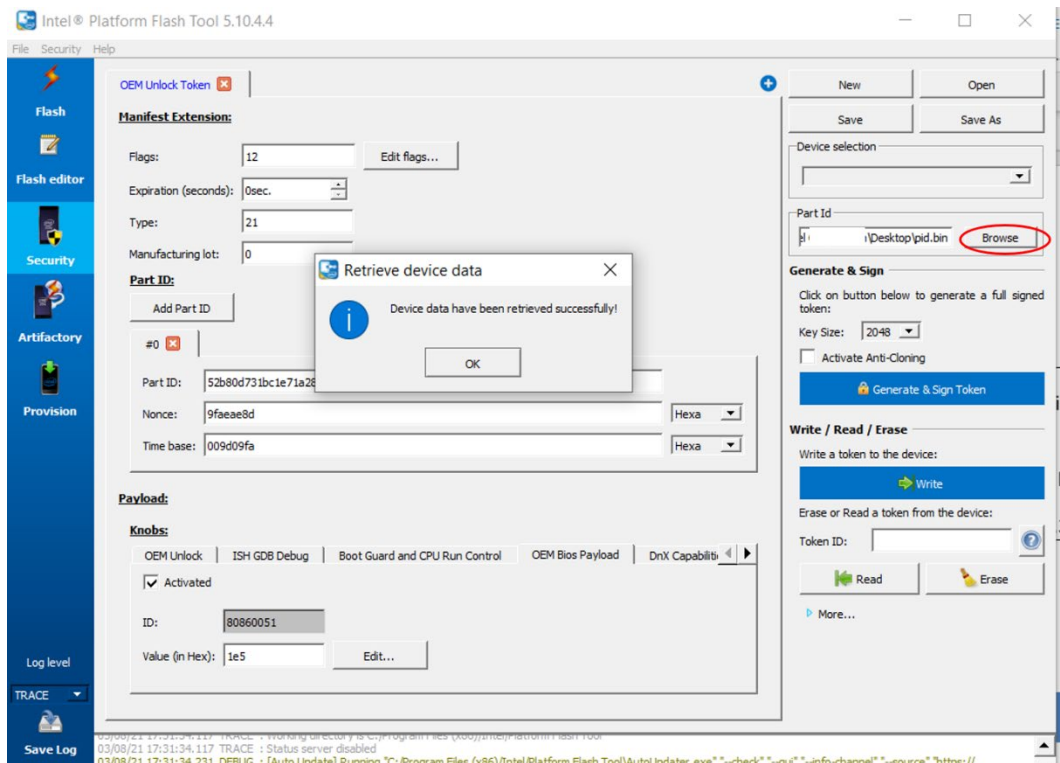


Figure 18

If there are multiple Part IDs, you can use the “Add Part ID” button as shown



Figure 19

### 3.2.2 Token Creation - CLI

To create the token, browse to the Intel® PFT's installation folder and launch the file `tokens_list_<project_name>.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<tokens version="1" format="1" view_filter="simple">
  <token name="OEM Unlock Token" platform="Lakefield" token_id="3">
    <!--Set the desired expiration time for token to be valid-->
    <manifest_extension type="21" length="0" version="1" number_id="1" payload_version="1" flags="0" expiration_seconds="3600" manufacturing_lot="0">
      <!-- Enter part_id, nonce and time_base of your device-->
      <part_id nonce="00000000" part_id="0x" time_base="00000000"/>
    </manifest_extension>
    <payload nb_knobs="4">
      <knob name="OEM Unlock" id="0x80860002" data="0x00000001" activated="1"/>
      <knob name="Bootguard" id="0x80860030" data="0x00000002" activated="1"/>
      <knob name="OEM Bios debug" id="0x80860051" data="0x00000001" activated="1"/>
      <!--"activated" points to which knob is activated>
      <"data" depends on the list below. The data is in hex>
      <Bitmap options:>
        <EnableGetNvmProperties - bit 0>
        <EnableNvmConfiguration - bit 1>
        <EnableClearPlatformConfiguration - bit 2>
        <EnableWritingNvmContent - bit 3>
        <EnableReadingNvmContent - bit 4-->
      <knob name="DnX Capabilities" id="0x80860101" data="0x00000001f" activated="1"/>
    </payload>
  </token>
</tokens>
```

Figure 20

As shown above, set the token Flags as mentioned in section 3.3.1.1 like *expiration*, *anti-replay* etc under “manifest extension” tab.

Note that the **flags** should be a hex value as described in the comment.

Also, choose the **knobs** that should be enabled for your debug purpose. Discussed in section 3.3.1.1.3

As shown in **example** above, OEM Unlock, Bootguard and OEM BIOS debug are all activated.

The “part id” can also be copied into this xml file after reading it using either of the methods provided in Section 3.3.1.2

Intel® DnX capabilities knob is also filled in with data and is activated in this example. For more information on this, please refer to the Intel® DnX User Guide.

Once the knobs are set, run the command



```
# token-manager-tool-cli.exe -c tmt_cli_config_file.xml -g
```

to generate the token. See snippets of sample output.

```
C:\Program Files (x86)\Intel\Platform Flash Tool> token-
manager-tool-cli.exe -c tmt_cli_config_file.xml -g
INFO   : Intel(R) Token Manager Tool CLI v1.7.0-0 (built on
Friday February 22nd 2019, 08:25:57 UTC)
INFO   : option 'c' used: config file for TMT CLI
INFO   : XML config file to use: 'tmt_cli_config_file.xml'
INFO   : Checking config file...
INFO   : Checking 'commands' section...
INFO   : Checking 'globalSettings' section...
INFO   : Checking Local signing parameters...
WARNING: Signing keys local passphrase info not set in XML
config file!
. . .
. . .
INFO   : Done!
INFO   : Config file looks ok :-)
INFO   : option 'g' used: generate the token binary
. . .
. . .
INFO   : Output token payload file:
c:\TEMP\tmt_tmp_token_PAYLOAD.tok
INFO   : Broxton token detected
INFO   : [TmtApi] Token has been successfully generated to
c:\TEMP\tmt_tmp_token_PAYLOAD.tok
INFO   : Done!
INFO   : Token payload done:
'c:\TEMP\tmt_tmp_token_PAYLOAD.tok'
INFO   : Bye!
```

## 3.3 Signing

### 3.3.1 Manifest header

Please also install the Intel® Mobile Signing Utility released in the kit for signing. Intel® PFT works with this tool for signing, once installed.

For customers who don't want to use Intel tools for signing, this step is used for test-signing the token binary with the required Manifest header, which the customer can sign later with actual private key that they own in their environment using their own preferred signing method, similar to other OEM signed components.



*Note: Please contact your Intel Customer representative if you have any questions regarding this step. Any mismatch of keys will cause the token to be discarded and hence this step is critical.*

### **3.3.1.1 Intel® PFT GUI**

In-order to sign the tokens, the key and signing information should be entered into the Intel® PFT tool under the “General settings” tab.

To create a password protected private key, using OpenSSL, using for example ‘foobar’ as the password, run the following command from the CLI:

```
# openssl.exe genrsa -passout pass:foobar -out  
privkey_pwd.pem 2048
```

*Note: A token with key hash that doesn’t match value in OEM Key Manifest will not work. Before you continue with token creation be sure the OEM key manifest is updated. See section 2.2 “Preparing the Platform to Accept OxM Debug Tokens”*

*OpenSSL tool mentioned above is one of the tools that can be used to create a private key and corresponding public key pair. Customers can have their own tool or method to cover this.*

As shown in Figure 21 below, Customers can enter the signing key information in the PFT under “Security”-> Security Option -> General Settings

The “Key” field consists of the key file location and the “Password” is the password if it was used to create the key as mentioned above.



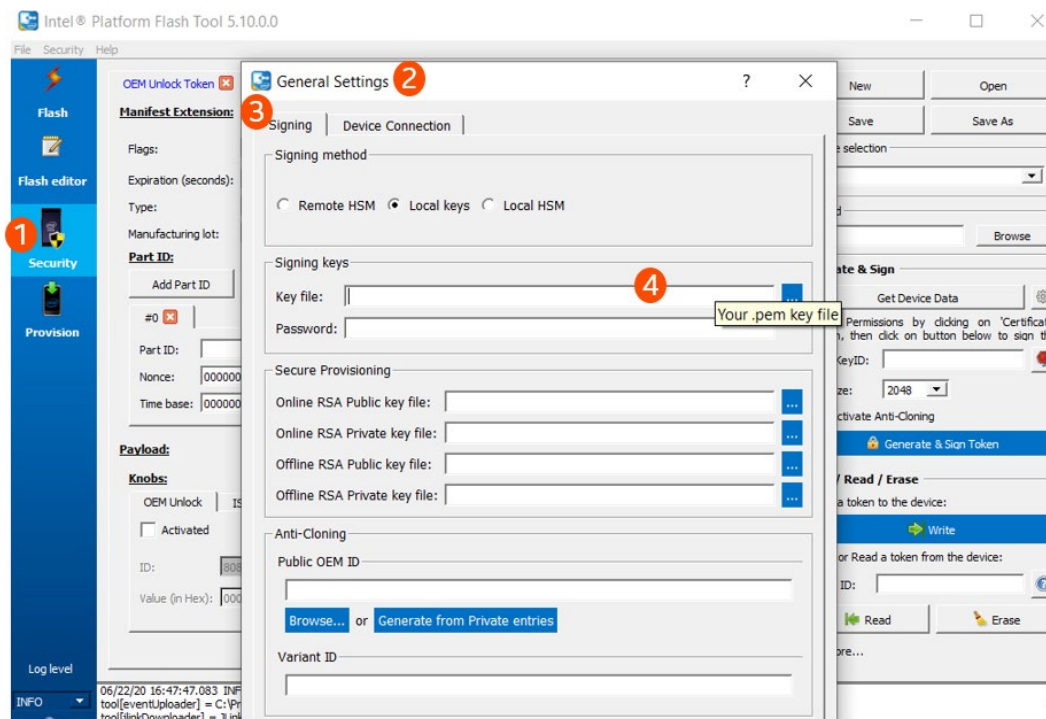


Figure 21

### 3.3.1.2 Intel® PFT CLI

Signing key information can be updated in the `tmt_cli_config.xml` by browsing to the Intel® PFT installation directory.

```
<globalSettings>
  <!-- connection types are: 0 = DnX, 1 = ADB, 2 = fastboot, 3 = FW DnX
  > "dnx_fw" is the DnX firmware module that is used to interact with the device in DnX mode -->
  <connection type="3" dnx_fw="" />
  <!-- signing types are: 0 = local keys, 1 = local HSM, 2 = remote HSM
  > for local keys (0), please specify .pem file as "local_key" and passphrase file as "local_passphrase" ("spid" and "keyid" are NOT used in this case)
  > for local HSM (1), please specify key ID as "local_key" and engine type (chil, pkcs11, etc.) as "local_passphrase" ("spid" and "keyid" are NOT used in this case)
  > for remote HSM (2), please specify spid and keyid ("local_key" and "local_passphrase" are NOT used in this case) - user rights (spid and keyid) can be retrieved using
  <signing keys type="0" local_key="C:\Users\...\.OpenSSL-Win32\bin\privkey.pem" local_passphrase="" spid="" keyid="" />
</globalSettings>
```

Figure 22

Under “signing\_keys”:

**type** is *local* and can be set by entering 2

**local\_key** is the path to the private key

**local\_passphrase** is the password used for creating this key.



Once the token is configured or the token payload binary created, the next step is to sign the token with the signing keys

### **3.3.2 Signing token**

If the information was already set per Section 3.3.1, this step will sign the token using the saved Private key information.

If OEM doesn't want to use Intel tool to sign the token with actual private key, this step can be used to sign token with debug/test key. Signing gives full structure that is acceptable by the FW.

***Note:** Do not skip this step. The token payload must have the correct Manifest header. If you have any questions, please contact your Intel® Customer Enablement representative.*

The customer can then replace the debug key with actual private key using their own tools. This step is similar to how Intel® MEU signs debug FW and then replaces debug key with production actual key.

### **3.3.3 Intel® PFT GUI**

After configuring the token settings and knobs from the template, generate and sign the token using the "Generate and sign Token" button. Refer to Figure 23

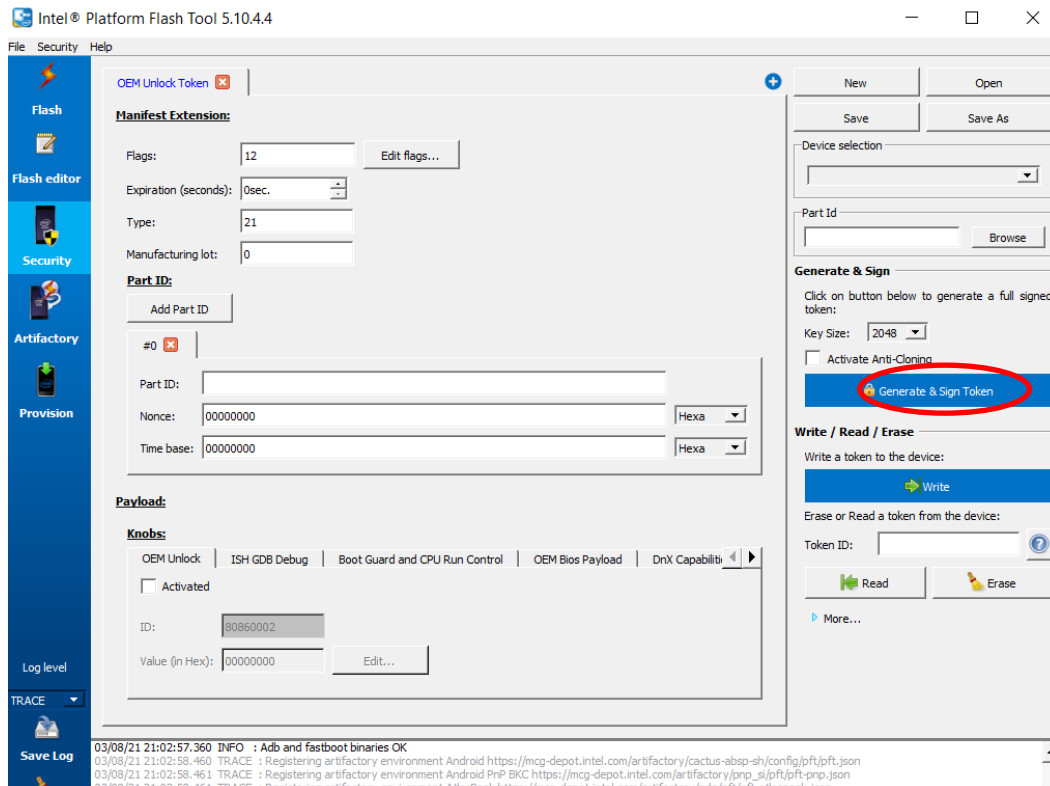


Figure 23

### 3.3.4 Intel® PFT CLI

Under the Intel® PFT's installation directory launch the terminal to use the command

```
# token-manager-tool-cli -c "tmt_cli_config_file.xml" -s
```

## 4 Token Injection

### 4.1 Introduction

Tokens can be injected into a platform using tools such as Intel® FPT or using Intel® DnX. Some tokens can also be compiled into the firmware image, using Intel® FIT. The Intel® PFT, used for creating tokens, can also be directly used to inject the token using Intel® DnX, via a UI button.

### 4.2 Using Intel® DnX

Intel® PFT only supports Token injection within the tool using Intel® DnX technology. Please check section 1.2 for more details. The target machine must be in Intel® DnX mode.

#### 4.2.1 Intel® PFT - GUI

The tool has a “Write” button as shown below to write the token to the system.

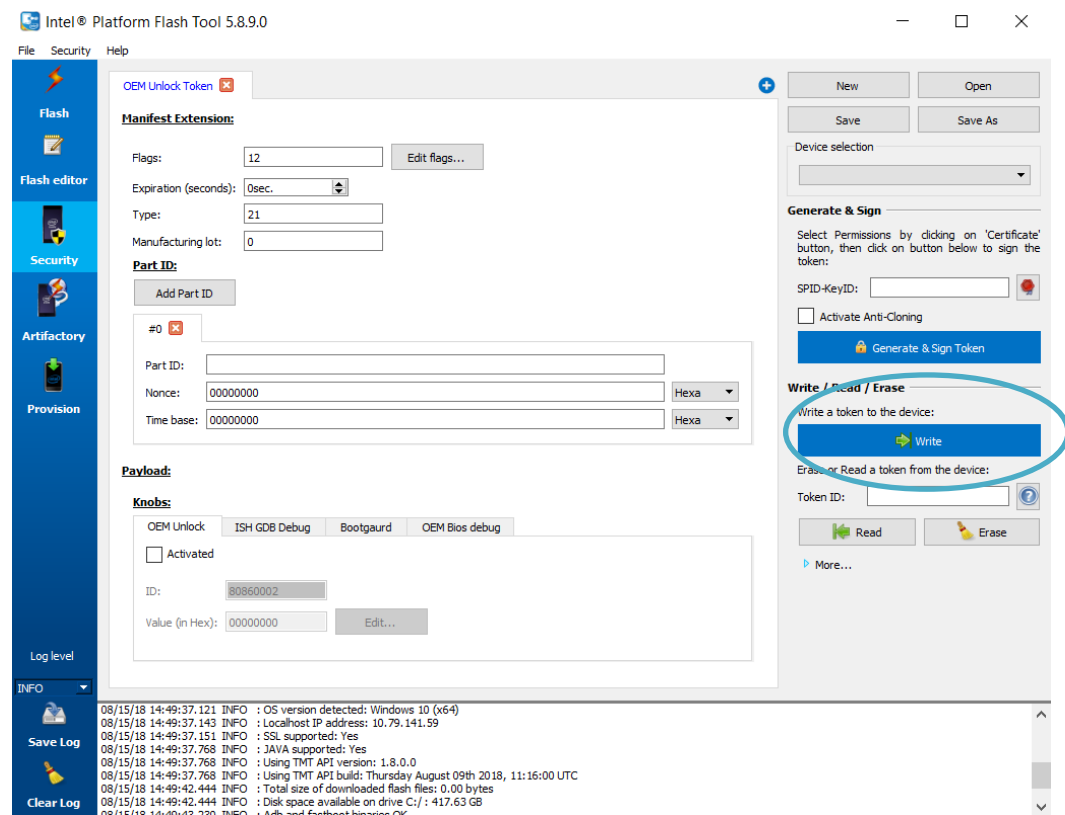


Figure 24

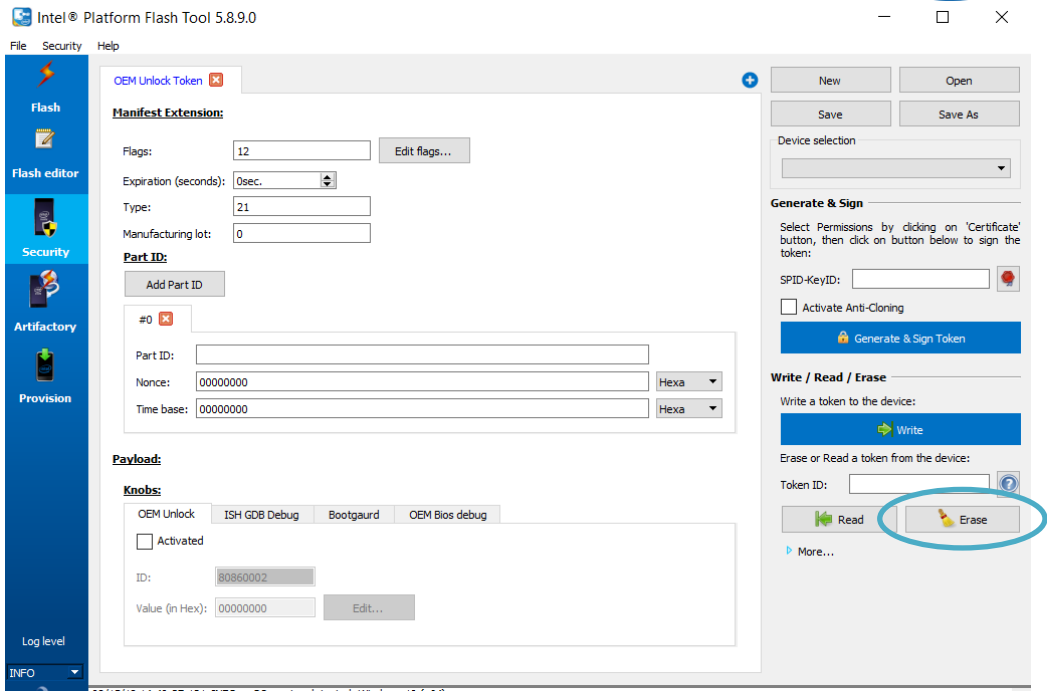


Figure 25

The token will be consumed and validated by the firmware on the next platform reset, so the machine should be rebooted after injection. It will remain there until it is erased, or the firmware is re-flashed, erasing the token.

4.2.2 Intel® PFT - CLI

Check section 1.2 for Intel® DnX setup. Launch a Command prompt to execute “dnxFWDownloader.exe” and use ‘*writetoken*’ command

Sample:

```
dnxFwDownloader.exe --command writetoken --fw_dnx
DNXP_0x1.bin --token token_to_write.bin --slot 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary DNXP_0x1.bin from CSME Firmware Kit
--token	path to the token
--slot	Slot Index of the token



## 4.3 NO Intel® DnX

### 4.3.1 Intel® FPT

If customer doesn't use Intel® DnX technology, the OEM Unlock Token can be injected into a platform using Intel® FPT. This should be followed by a global reset.

The token will remain there until it is erased, or the firmware is re-flashed, erasing the token.

Operation	Command Line
<b>Writes the token where the filename is the token name</b>	Fpt.exe -WRITETOKEN<file>  Fpt.exe -greset

Note that these APIs are unable to give any indication if the token passed validation or not. However, this information can be collected via Intel® System Trace.

### 4.3.2 Stitching a Token into the Firmware Image

The OEM Unlock Token can be compiled directly into the firmware image when it is built, using Intel® FIT. Use this step if you can re-flash the IFWI. Otherwise consider either Intel® FPT or Intel® DnX

As shown in below Figure, this information is entered under the Debug tab, in the Unlock Token field. An image prepared this way can be used for debug purposes but should **never** be burned on production systems.

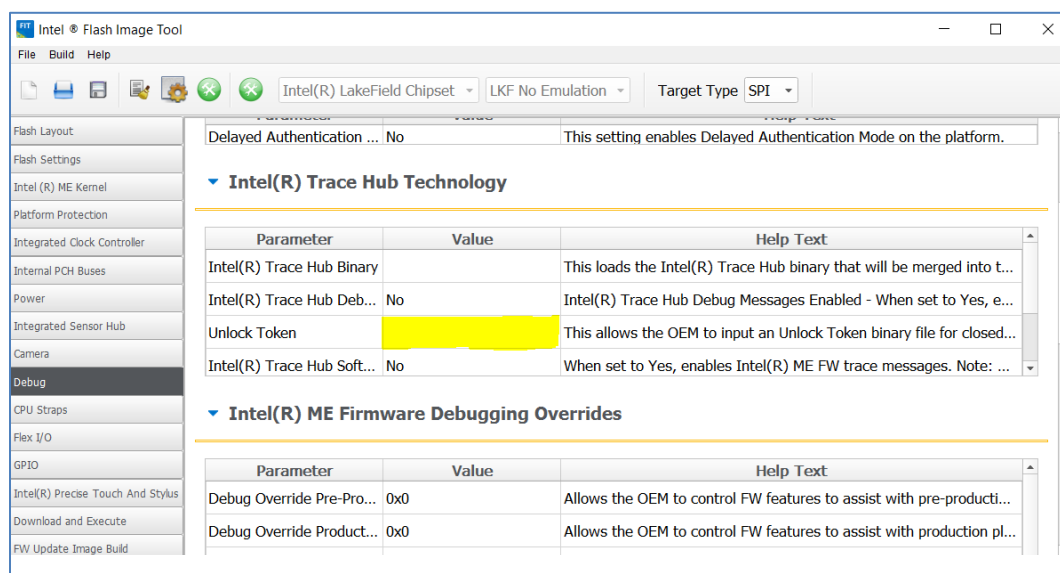


Figure 26

## 5 Erasing the Token

Tokens can be erased from the platform using the tools such as Intel® FPT or using Intel® DnX.

### 5.1 Using Intel® DnX

Please check section 1.2 for details on the setup. The target machine must be in Intel® DnX mode.

#### 5.1.1 Intel® PFT - GUI

Using the “Erase” button on the UI, this tool can clear the token from the device.

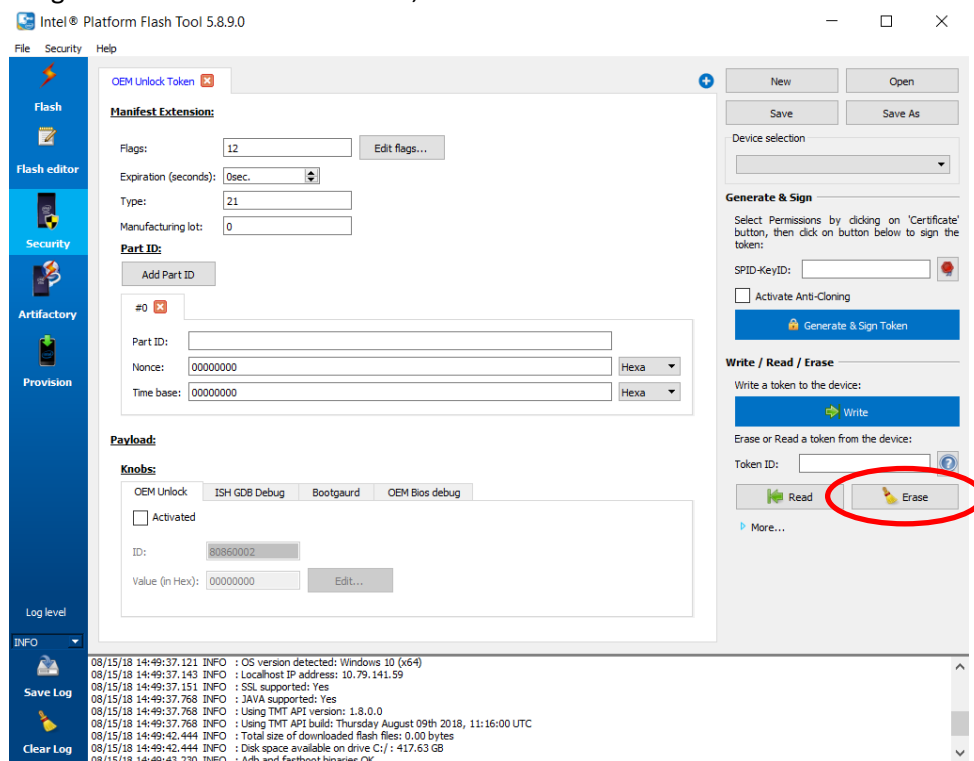


Figure 27



### 5.1.2 Intel® PFT - CLI

Open a command prompt in to execute the “dnxFwDownloader.exe” and use ‘erasetoken’ command.

Sample:

```
dnxFwDownloader.exe --command erasetoken --fw_dnx  
DNXP_0x1.bin --slot 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary DnXP_0x1.bin from CSME Firmware Kit
--slot	Slot Index of the token

## 5.2 NO Intel® DnX

### 5.2.1 Intel® FPT

If customer doesn’t use Intel® DnX technology, the OEM Unlock Token can be injected into a platform using Intel® FPT, running on the platform OS. This should be followed with a global reset.

The token will remain there until it is erased, or the firmware is re-flashed, erasing the token.

Operation	Command Line
Delete the token for the token ID provided	FPT.exe - ERASETOKEN<pid>
	Fpt.exe -greset

Note that these APIs are unable to give any indication if the token passed validation or not. However, this information can be collected via Intel® System Trace.

### 5.2.2 Re-flash a new image

Another option is to flash a completely new image without any OEM token binary which was initially put as mentioned in Section 4.3.2



## 6 Reading the Token

### 6.1 Using Intel® DnX

Please check section 1.2 for details on the setup. The target machine must be in Intel® DnX mode.

Reading of token is currently only supported via Intel® DnX

#### 6.1.1 Intel® PFT – GUI

Click on the “Read” button as shown below. The tool will ask to save this token with a file name. Browse to the location where the tool should save this token to.

The device selected to read the token from should show in the “Device selection” box.

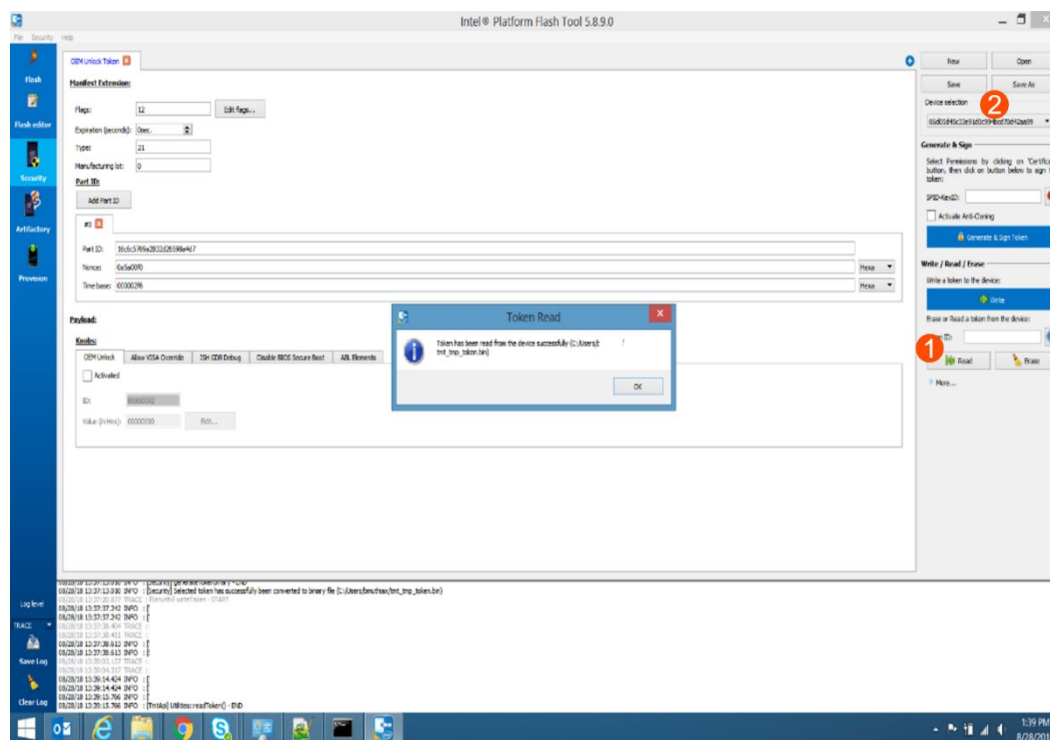


Figure 28



### 6.1.2 Intel® PFT - CLI

Open a command prompt in this location to run the “dnxFwDownloader.exe” using your OS Terminal and use ‘readtoken command.

Sample:

```
dnxFwDownloader.exe --command readtoken --fw_dnx  
DNXP_0x1.bin --path read_token.bin --slot 0
```

Where:

Option	Description
--fw_dnx	path to the DnX module binary DnXP_0x1.bin from CSME Firmware Kit
--path	path to output file to dump the content of the token
--slot	Slot Index of the token



## 7 Debugging OxM Debug Token Injection

---

The OEM Unlock Token is only examined by firmware at system boot, and so the token injection cannot return any failure codes.

If the token is failing to unlock the platform or enable other debug features as expected, Intel® Trace Hub messages must be examined, as they indicate why a token was rejected.

These messages are available using the Intel® System Studio tool available to download via Intel website.

Example when OxM Debug token is rejected due to authentication problem –

Using Intel® Trace Hub messages on the SUT :

Example of a failure case -

---

```
"[RBE] Load Module index (2)" Normal
```

```
"RBE_EVT_TOK_TYPE - 0x3" Normal
```

```
"[RBE] Validate manifest type (1) of partition (4)"  
Normal
```

```
"[RBE] Token authentication failed - 36" Normal
```

```
"Reject secure token error 36" Normal
```

```
"[RBE] CSE Boot stall flow" Normal
```

```
"[RBE] Poll for boot stall done" Normal
```

---



## 8 References

---

Document	Source
575021-575021-cn1-oem-secure-tokens-ug-rev0p5.pdf	<a href="https://www.intel.com/content/www/us/en/design/resource-design-center.html">https://www.intel.com/content/www/us/en/design/resource-design-center.html</a>
PFT_Security_User_Guide.pdf	Intel® Platform Flash tool
Intel® DnX User Guide	CSME FW kit
Intel® Signing and Manifesting Guide	
Video explaining token creation/injection and debug on Customer training portal (use your log in detail)	<a href="https://techtraining.intel.com/ProjectTraining/Course/2000234/">https://techtraining.intel.com/ProjectTraining/Course/2000234/</a>
Intel® System studio	<a href="https://registrationcenter.intel.com/">https://registrationcenter.intel.com/</a>