

Intel® Converged Security and Management Engine 15.0 Secure Tokens Guide

Revision 1.1

April 2020

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 © 2020, Intel Corporation. All rights reserved.



Contents

Contents

1	Introduction	6
1.1	Goal	6
1.2	Overall Work Flow	6
1.3	Pre-Requisites.....	7
1.4	Tools Used In This Document	7
1.5	Terminology	7
2	Overview of Secure Tokens	8
2.1	Introduction	8
2.2	Preparing the Platform to Accept Secure Tokens	8
3	Creation and Signing of Secure Tokens	9
3.1	Introduction	9
3.2	General Settings – Keys and Signing	9
3.2.1	Using Intel® PFT GUI	9
3.2.2	Using Intel® PFT command-line utility.....	10
3.3	Token creation.....	11
3.3.1	Using template	11
3.3.2	Token creation - “Security” tab.....	16
3.3.3	Using Intel® PFT command-line utility.....	18
3.4	Signing the token	19
3.4.1	Signing token from Intel® PFT Template	20
3.4.2	Using Intel® PFT “Security” tab option	20
3.4.3	Using Intel® PFT command-line utility.....	20
4	Injection of Token on Platform	21
4.1	Introduction	21
4.2	Injection	21
4.2.1	Injection using Intel® FPT.....	21
4.2.2	Injection using Intel® PFT GUI.....	21
4.2.3	Using command line	22
4.2.4	Building a Token into the Firmware Image	24
4.3	Clearing of Token	25
4.3.1	Intel® FPT	25
4.3.2	Intel® Platform Flash Tool GUI for DnX based systems	25
4.3.3	Using command line	26
4.4	Reading of Token	26
4.4.1	Using Intel® PFT GUI for DnX commands	26
4.4.2	Using command line	27
5	Debugging Secure Token Injection	28
6	Intel® PFT options for Intel® CSME 15.0	29
6.1	Platform selection.....	29
6.2	Token template selection.....	30
6.3	Miscellaneous	30
7	References.....	32





Revision History

Revision Number	Description	Revision Date
0.5	Initial Release	Feb 2018
0.80	Updated for DnX over SPI only	May 2019
1.0	Aligned revision number to 1.0	Dec 2019
1.1	Added reference RKL EDS Updated to revision 1.1	April 2020

1 Introduction

This document gives an overview of Secure Tokens for Intel® CSME 15.0

1.1 Goal

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

1. Prepare the platform to work with Secure Tokens
2. Create Secure Tokens
3. Inject Secure Tokens to the platform
4. Clear Secure Token from platform after use.

1.2 Overall Work Flow

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

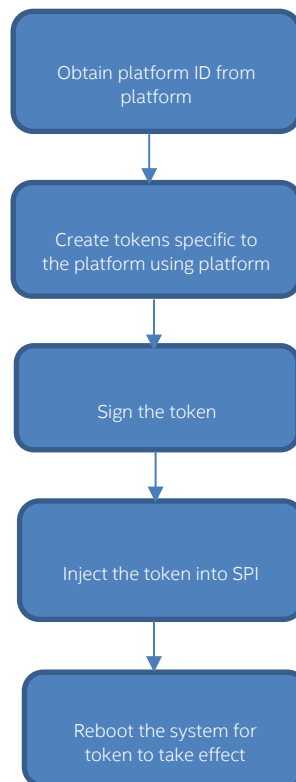


Figure 1

Token is usually created for a specific platform to ensure the security of the platform. The details of each step in the work flow is covered in



the rest of this document. In addition to injecting the token into the platform and using the token, the process for deleting a token is also described.

1.3 Pre-Requisites

The user should download and install the following applications, included in the firmware kit:

- Intel® Platform Flash Tool (PFT)
- Intel® Flash Programming Tool (FPT)

An overview of the signing and manifesting process is described in:

- Intel® CSME 15.0 Signing and Manifesting Guide (included in the FW kits)

1.4 Tools Used In This Document

The following tools are used within this document:

- Intel® Platform Plash Tool (PFT)
- Intel® Flash Programming Tool (FPT)

1.5 Terminology

Term	Description
DnX	Download and Execute
EOM	End of Manufacture
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

Table 1



2 Overview of Secure Tokens

2.1 Introduction

Secure Tokens are used in Intel® CSME 15.0 platforms to allow operations otherwise blocked.

The OEM Unlock Token unlocks debug capabilities such as

- ISH debug, CPU probe, BIOS debug etc

Tokens are digitally signed so that the target platform knows to accept them.

2.2 Preparing the Platform to Accept Secure Tokens

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

Intel® CSME 15.0 platforms are manufactured with an OEM Key Manifest as part of the IFWI image.

One of the fields in the OEM Key Manifest is for the OEM Unlock Token. These should be populated with the hashes of the public keys, matching the private keys with which the tokens will be signed. A token whose key does not match the relevant hash in the OEM Key Manifest will be rejected by the platform.

An overview of the signing and manifesting process is described in "Intel® CSME 15.0 Signing and Manifesting Guide".

Details instruction on creating OEM key manifest are in "FW Bring up guide".

Both documents are included in the firmware kits.



3 Creation and Signing of Secure Tokens

3.1 Introduction

The Intel® Platform Flash Tool (PFT) includes a module which is the tool provided for Secure Token creation on Intel® CSME 15.0 platforms.

The module supports many platforms, and displays many options not supported on the Intel® CSME 15.0 platforms. This guide will only cover the POR features, and show how to create Secure Tokens for Intel® CSME 15.0 platforms.

Intel® PFT is able to generate both the token payload only, and the full signed token (i.e. a signed header appended with the token payload).

Note: Intel® Platform Flash Tool (PFT) uses DnX capability over SPI for token related operations.

3.2 General Settings – Keys and Signing

3.2.1 Using 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 Secure 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 2 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 used to create the key as mentioned above.

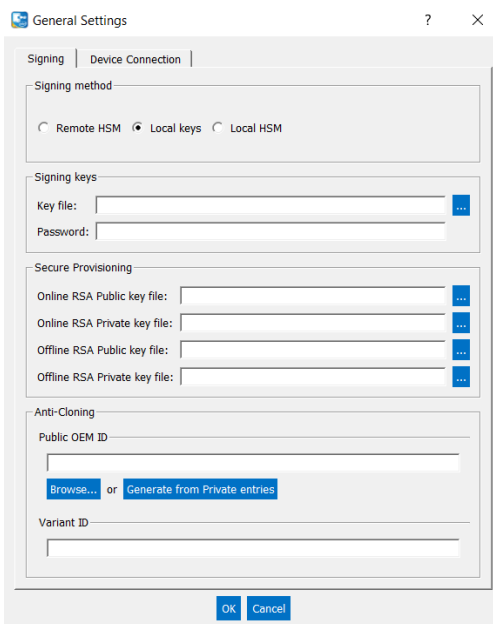


Figure 2

3.2.2 Using Intel® PFT command-line utility

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\swaydand\Desktop\unlock_exp\OpenSSL-Win32\bin\privkey.pem" local_passphrase="" spid="" keyid="" />
</globalSettings>
```

Figure 3

As shown here, in the "connection" tab, the type is 3 for dnx_fw

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.



3.3 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. Also, the tool allows to separately generate the token payload binary and separately sign the token per the requirements of the user, thus allowing more control over the token creation and signing. Both these techniques are described below.

3.3.1 Using template

As shown in Figure 4, using the Intel® PFT, Click on the **"New"** button, and then select Tigerlake or Rocketlake as the target platform, and OEM Unlock Token as the token template for an OEM Unlock Token.

Note: Below figure uses Lakefield as an example.

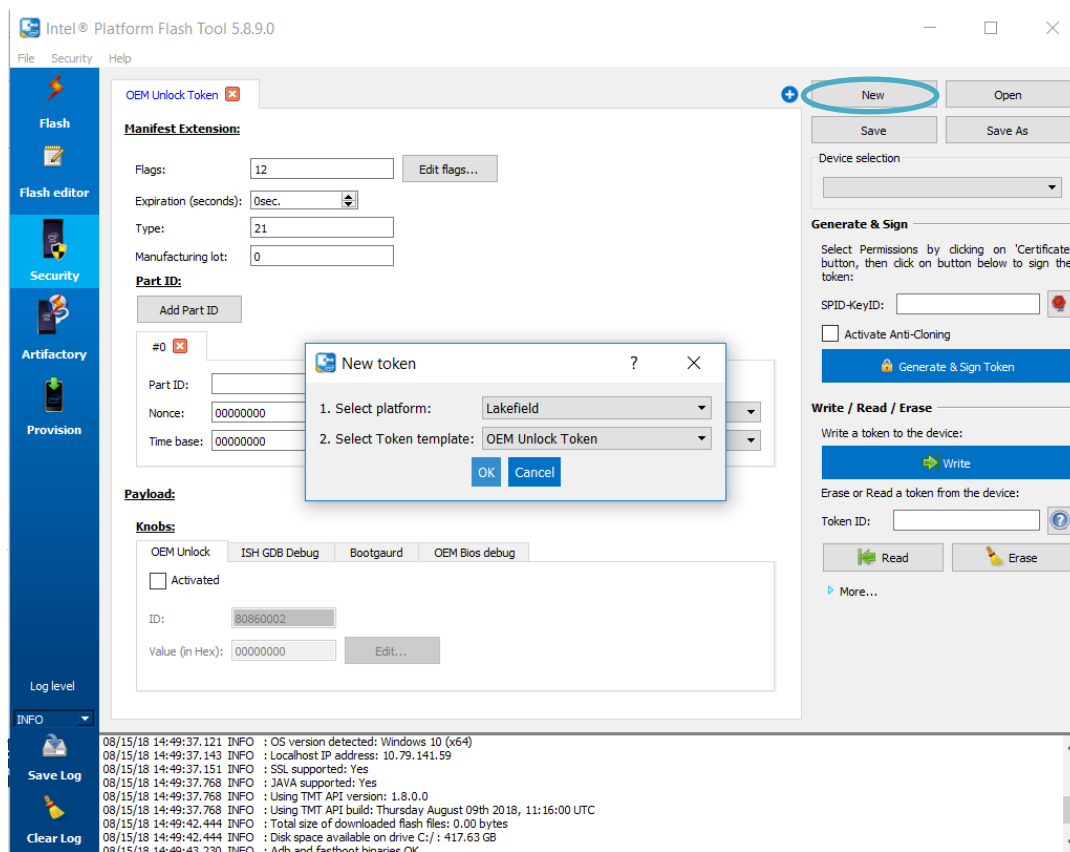


Figure 4



3.3.1.1 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:

3.3.1.1.1 Flags

In the “Flags” section you can select either -

- **Globally valid:** This means that the token can be used on any platform whose token key hash matches that of the token, and is not tied to a particular platform ID. Once platform is post End of Manufacturing only part specific token is accepted i.e., only token with PID will work.
- **No Anti-replay:** Anti-Replay protection stops a token being re-used on the same device after it has been cleared. This option is only relevant for tokens tied to a particular platform ID.
- **No expiration:** This means that the token has no time limit. **Token expiration is only relevant on tokens with anti-replay,** because otherwise you can re-use the token after RTC clear.

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

3.3.1.1.2 Part ID

In the main screen you can set:

- **Expiration timeout** (if relevant)
- **Part ID.** This is only relevant for a token that is not Globally Valid.
 - You can retrieve the Part ID data using Intel® FPT, by calling
FPT.exe -GETPID <file>
This command will retrieve the part ID into a file. You can open the file to copy and paste the data into the relevant fields.

If there are multiple Part IDs, you can use the “Add Part ID” button to create new windows (Figure 5)



Part ID:

Add Part ID

#0

Part ID:

Nonce: Hexa

Time base: Hexa

Figure 5

- To automatically derive the Part ID using DnX :
With DnX you can set the General Setting **“Automatically get the device data when generating tokens”** (Figure 6), and then to uncheck the **“Globally valid”** flag in the flags section. This will then get the Part ID data directly from the platform as the token is generated, and generate the token specifically for that platform.
- DnX commands can be used on a command line tool to get the part id as well. Please refer to Intel® DnX User Guide.

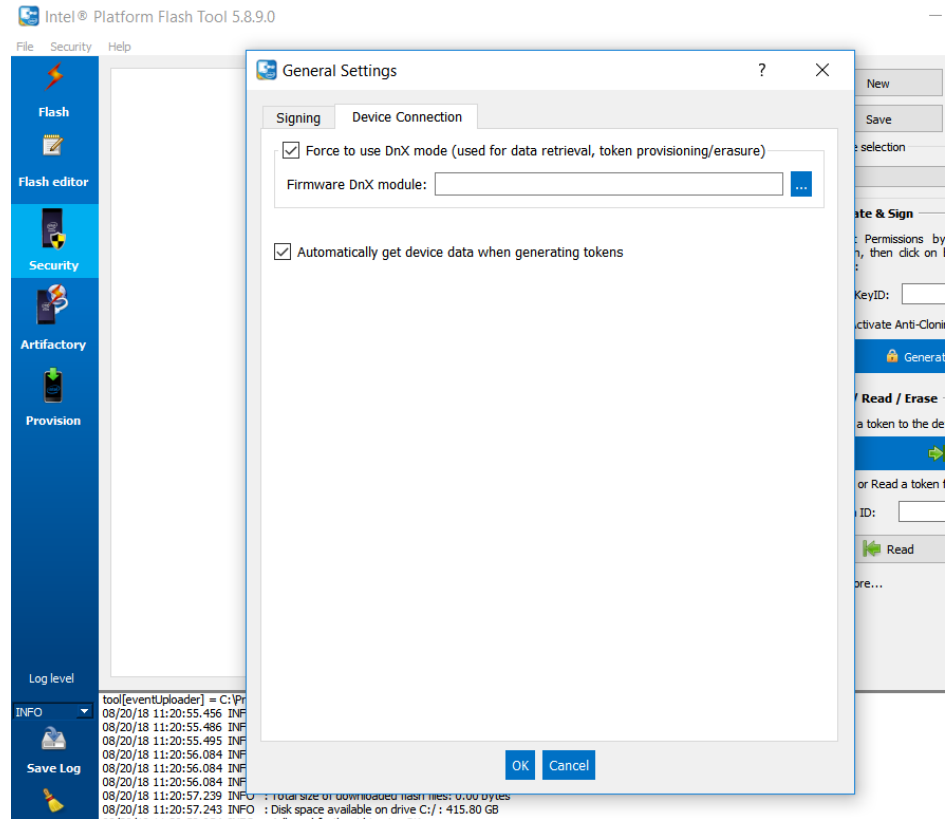


Figure 6

This requires a USB connection between the management console, and the target platform, and also that you have enabled the DnX option in the General Settings Dialog.

3.3.1.1.3 Knobs

The Intel® Platform Flash Tool provides options for “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 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.

See Figure 7 below.

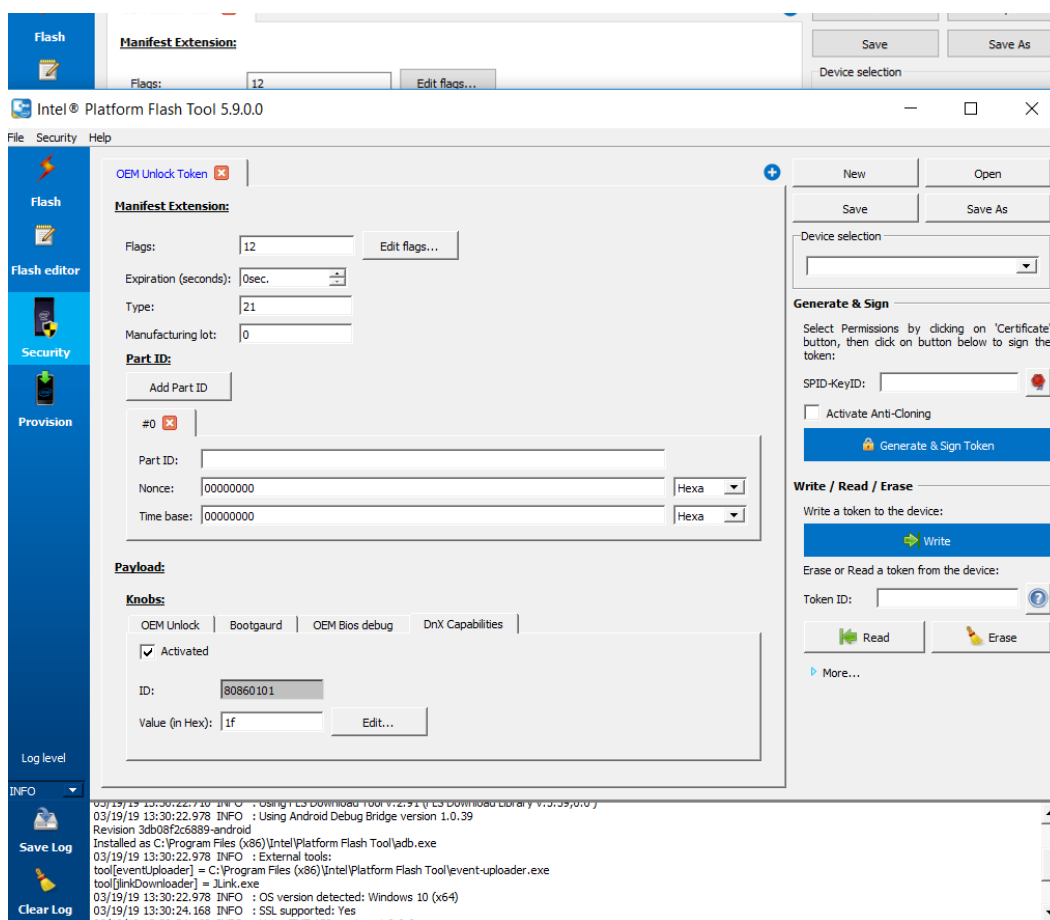


Figure 7

Note: Do not attempt to change the value manually.

Table 2 below gives an explanation of the various knobs:

Knob	Meaning
OEM Unlock	Allow an OEM (Orange) unlock
ISH GDB Debug	Enable ISH GDB supportThis allows OEMs to pass 32bit data into BIOS via debug token
BootGuardKnob	<p>BootGuardDisabled: Boot Guard Disable is used for platforms that have BTG enabled in IFP fuse but who do not wish to enable boot block verification process while using secure tokens.</p> <p>BootGuardNoEnforcement: This will allow users to boot to OS with a BTG failing system in order to collect additional debug</p>



	<p>information or to check other features without being blocked by BTG.</p> <p>BootGuardNoTimeouts: This Token has two usages:</p> <ol style="list-style-type: none">1. Allows to debug ACM\BIOS related flows without BTG flow failing.2. If BTG flow fails due to timeout, this token will allow OEMs to see if the problem is the timer duration being too short or a problem with ACM\BIOS. <p>BootGuardnoEnforcementAndTimeouts: This is a combination of the two features above. Allows for debug capabilities without getting a BTG enforcement (in the event that the issue is not timeout related).</p> <p>.</p>
OEMbiosDebug	This allows OEMs to pass 32bit data into BIOS via debug token.
DnXCapabilities	This allows the OEMs to perform certain DnX capabilities after EOM. Please refer to the Intel® DnX User Guide

Table 2

3.3.2 Token creation - “Security” tab

The user might want to create the token payload binary only, that the user will sign later with either Intel® PFT, or through another mechanism.

For this, open a token (saved/new one), update fields as needed, and then select “Security – Generate Token Payload Binary” from the menu as shown in Figure 8 and 9 below.

Please note, the options under “Security” tab provides only the step by step control over creation of token payload binaries and signed tokens. The user can opt for “Generate and Sign” button on the main GUI page for one-shot generation and signing of tokens.

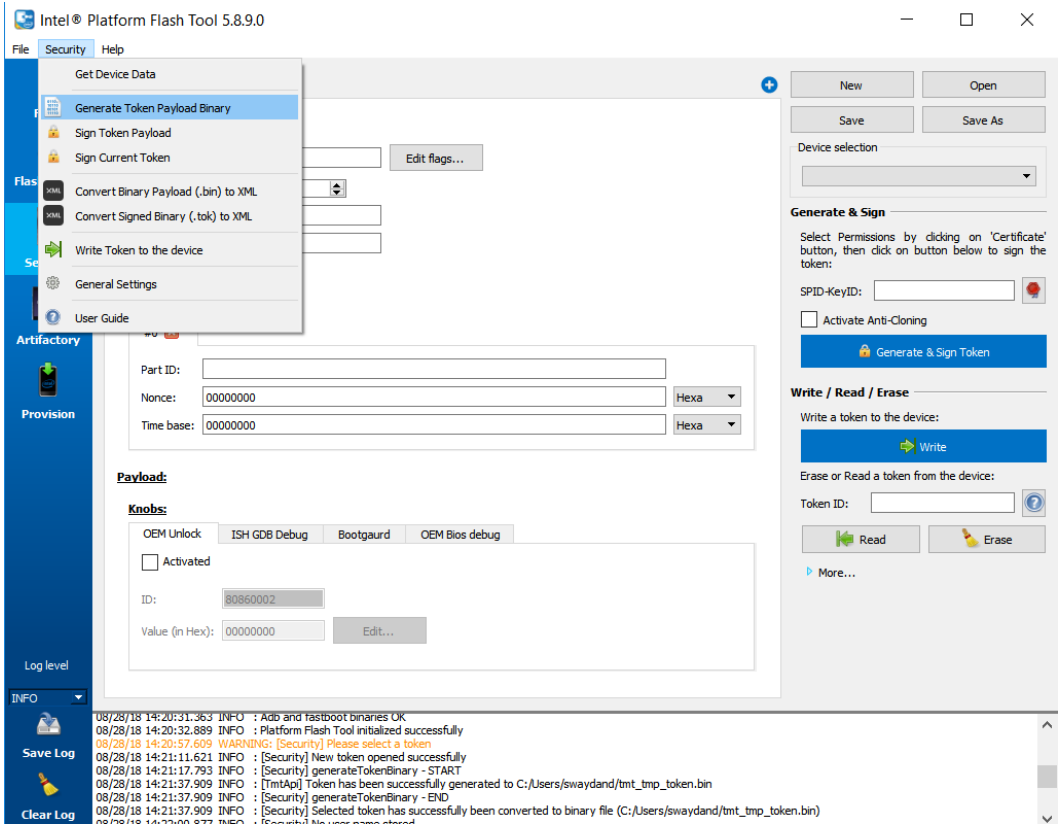


Figure 8

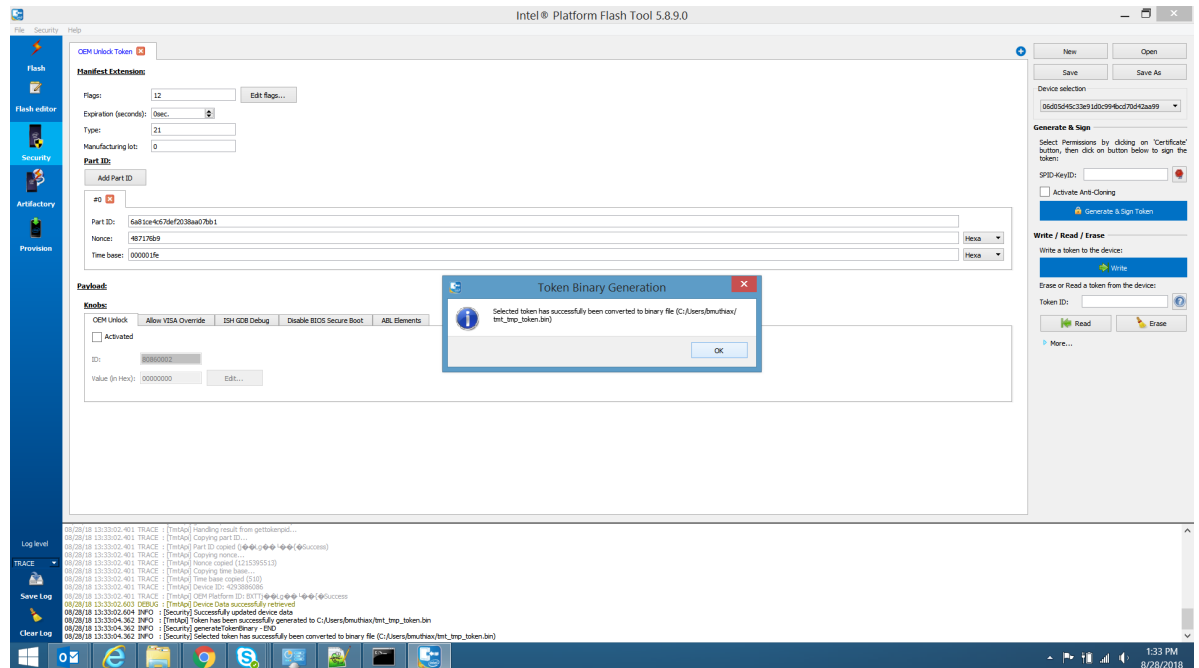


Figure 9

3.3.3 Using Intel® PFT command-line utility

To create the token, browse to the Intel® PFT's installation folder and launch the file `oem_unlock_token_template_Tigerlake.xml` or `oem_unlock_token_template_Rocketlake.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_ids="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="0x80860003" data="0x00000002" activated="1"/>
      <knob name="OEM Bios debug" id="0x80860001" data="0x00000001" activated="1"/>
      <!-- "activated" points to which knob is activated
      <"data" depends on the list below. The data is in hex>
      <Bitmap options>
        <EnableNvmProperties - 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 10

As shown above, set the token Flags as mentioned in section 3.1.1.1 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.

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

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, use 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!
```

3.4 Signing the token

Once the token is configured or the token payload binary created, the next step is to sign the token with the correct signing keys as updated in the “General settings” tab.



3.4.1 Signing token from Intel® PFT Template

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 11. Login information for signing server will be needed for this.

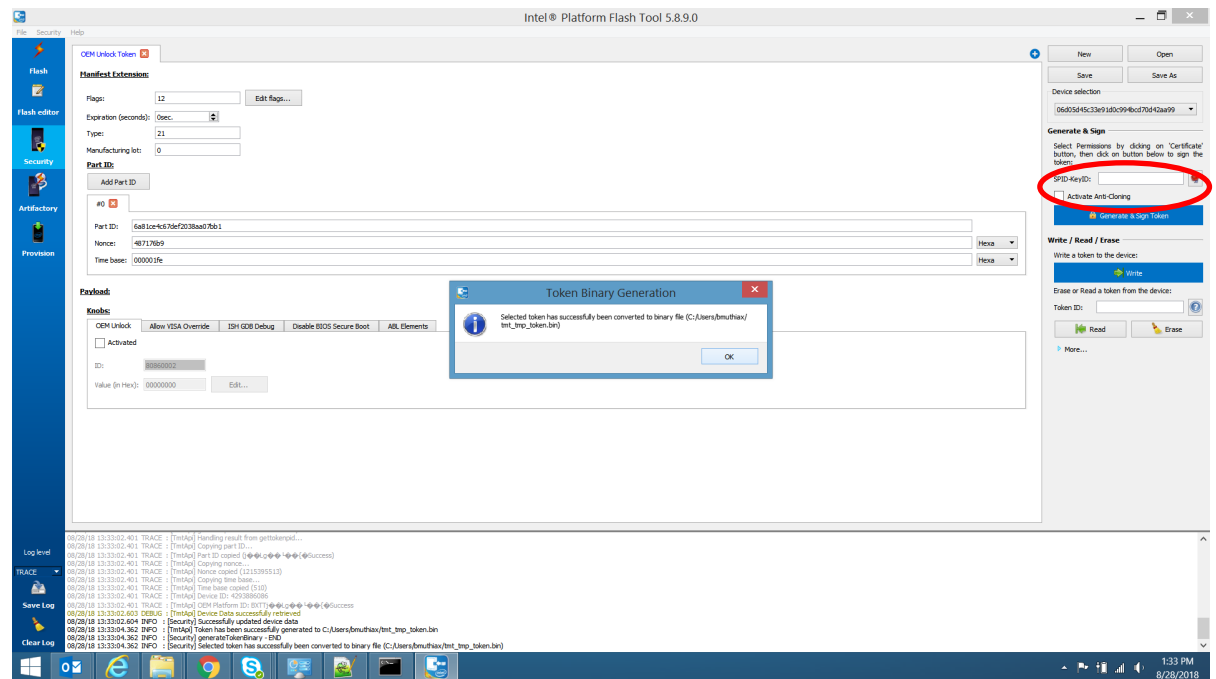


Figure 11

3.4.2 Using Intel® PFT “Security” tab option

If an existing token payload binary should be only signed fully using Intel® PFT, the option “Sign Token Payload” under “Security” tab can be used.

Please note, the options under “Security” tab provide only the step by step control over creation of token payload binaries and signing of tokens. The user can opt for “Generate and Sign” button on the main GUI page for one-shot generation and signing of tokens.

3.4.3 Using Intel® PFT command-line utility

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 Injection of Token on Platform

4.1 Introduction

Tokens can be injected into a platform using the HECI interface, and tools such as Intel® FPT, or using 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 DnX, via a UI button.

4.2 Injection

4.2.1 Injection using Intel® FPT

The OEM Unlock Token can be injected into a platform using Intel® FPT, running on the platform OS. The token will be read 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.

Intel recommends never releasing to customers a platform with an erased OEM Unlock Token, but to re-flash the full firmware image instead.

Operation	Command Line
Writes the token where the filename is the token name	Fpt.exe -WRITETOKEN<file>
Delete the token for the token ID provided	FPT.exe - ERASETOKEN<pid>

Table 3

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.

4.2.2 Injection using Intel® PFT GUI

The OEM Unlock Token can be injected into a platform using DnX. This requires the management console to be connected to the target platform with a USB cable. The target machine must enter into DnX mode. Depending on OEM implementation, there may be an explicit



hardware trigger for this. Alternatively, connecting the target system to the management console should enter the target system into DnX mode until the DnX timeout is reached. The DnX APIs for tokens are only available while the target system is in DnX mode.

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

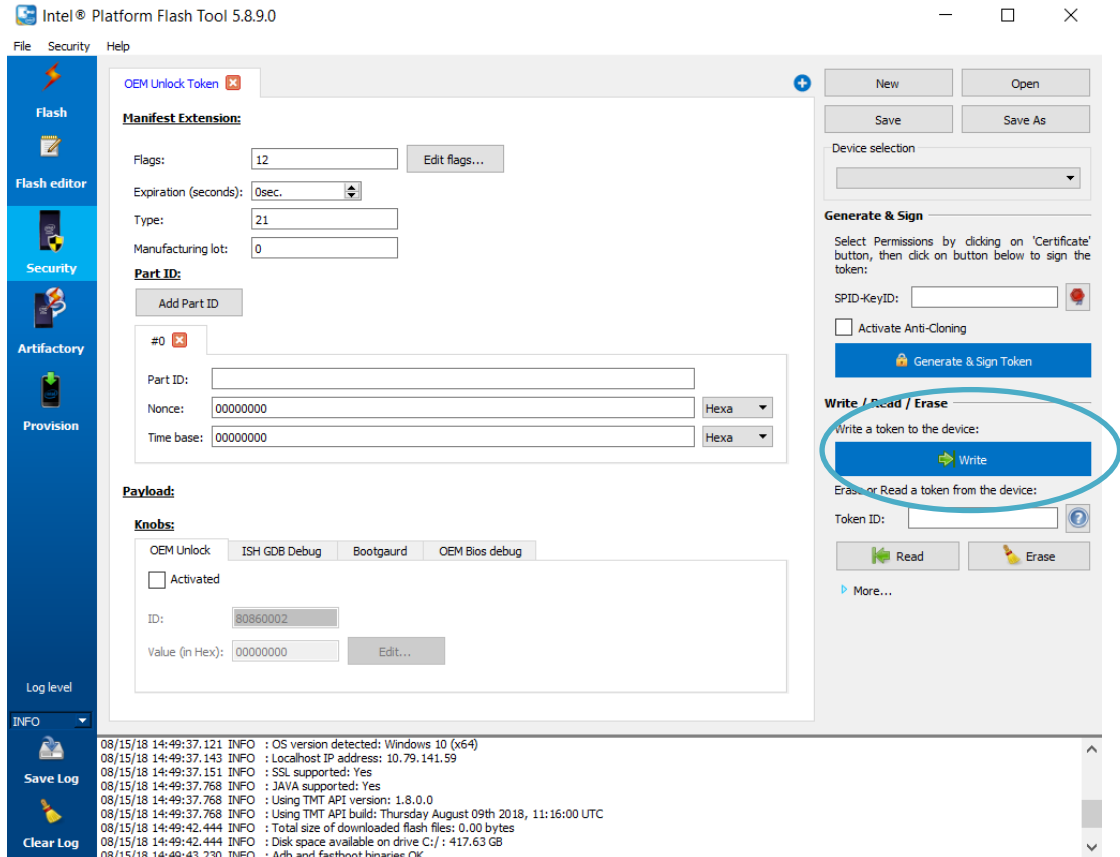


Figure 12

4.2.3 Using command line

This is covered using in the Intel® DnX User Guide.

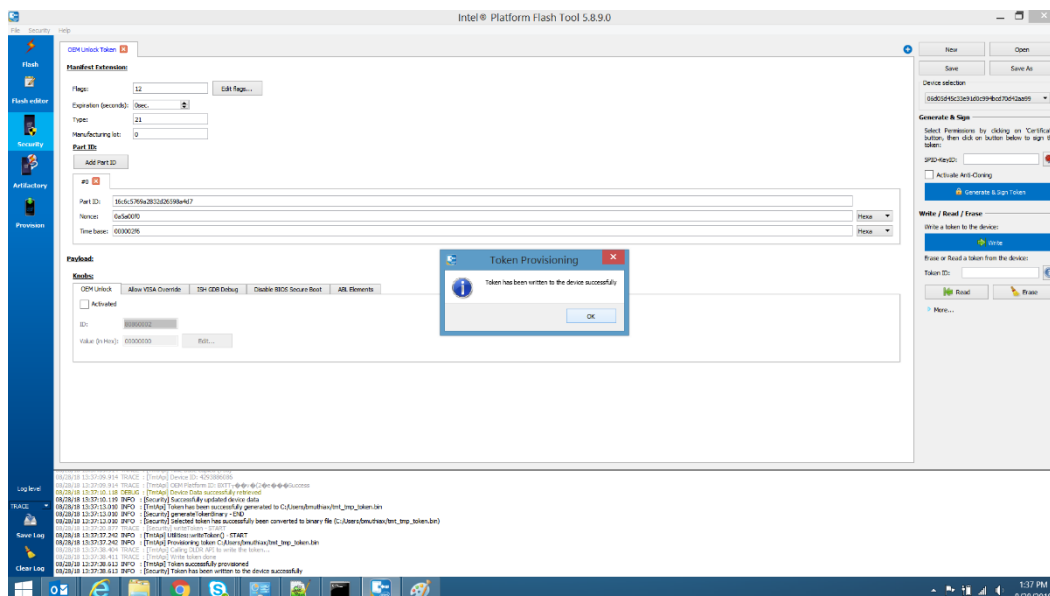


Figure 13

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.

Intel recommends never releasing to customers a platform with an erased OEM Unlock Token, but to re-flash the full firmware image instead.

4.2.3.1 Write token failures

In case of failure to write a token, the Intel® PFT shall display an error pop-up as shown.

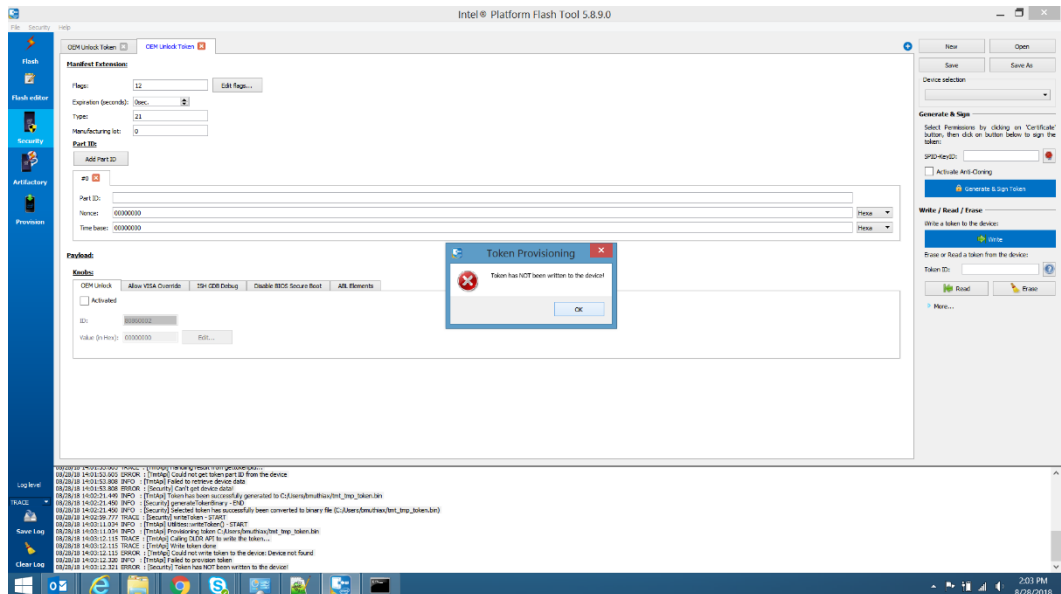


Figure 14

4.2.4 Building 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.

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.

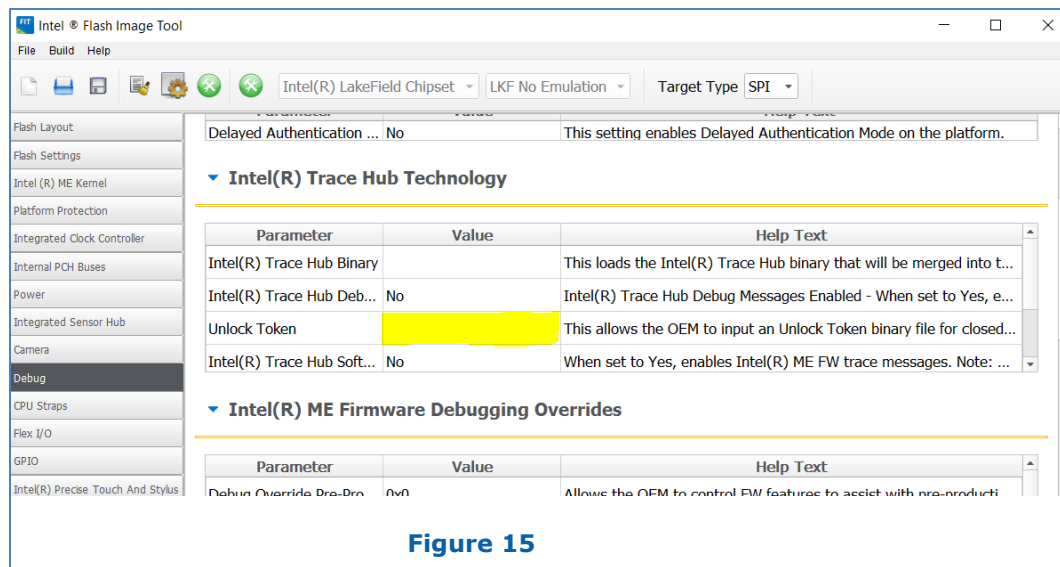


Figure 15



4.3 Clearing of Token

The OEM Unlock token survives a reboot, and must be erased using the following options.

4.3.1 Intel® FPT

Using `FPT.exe - ERASETOKEN <pid>` via the command line can erase the token.

4.3.2 Intel® Platform Flash Tool GUI for DnX based systems

Using the “Erase” button on the UI, this tool can clear the token from the device. This requires the management console to be connected to the target platform with a USB cable. The target machine must enter into DnX mode.

Intel recommends never releasing to customers a platform with an erased OEM Unlock Token, but to re-flash the full firmware image instead.

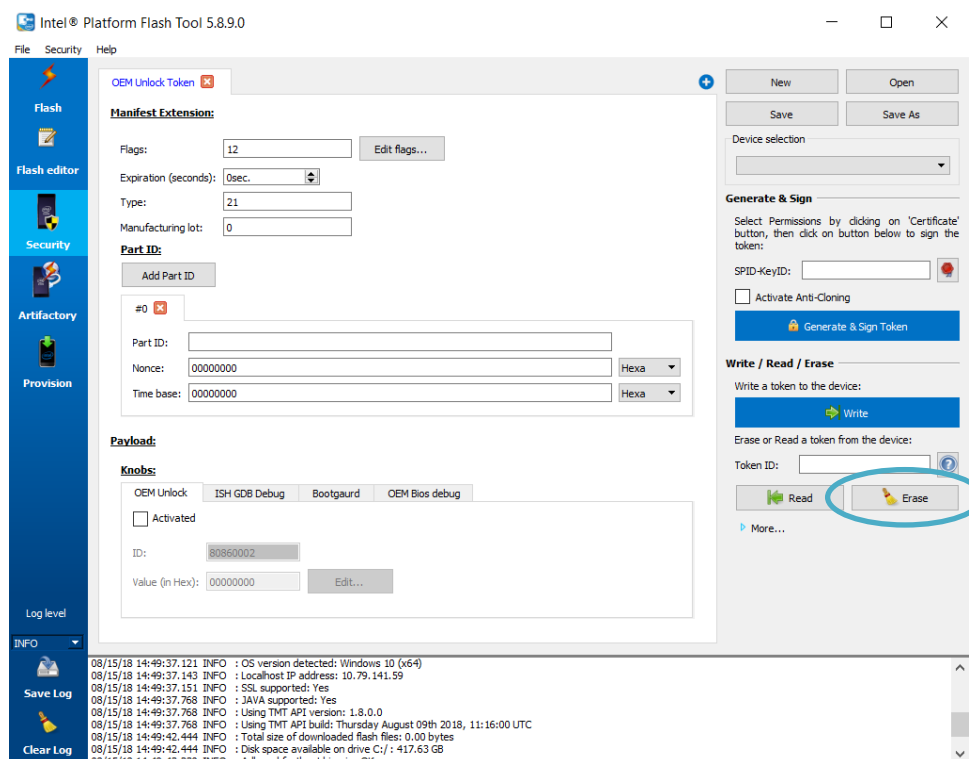


Figure 16



4.3.2.1 Erase Token failures

The Intel® PFT tool will show an error dialog box in case of failure erasing a Token as shown below.

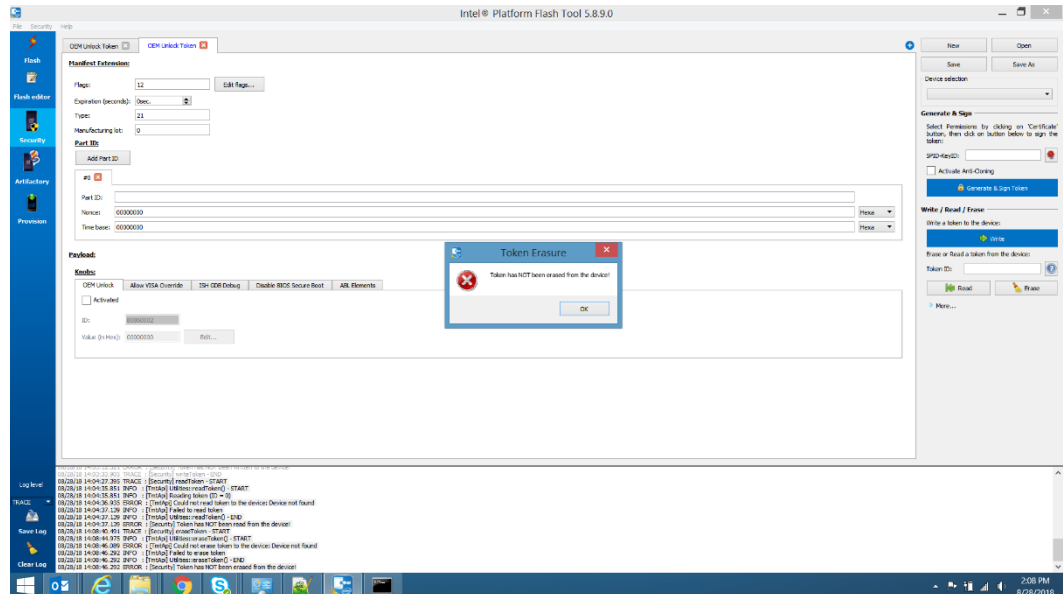


Figure 17

4.3.3 Using command line

This requires a USB connection between the management console, and the target platform, and also that you have enabled the DnX option in the General Settings Dialog.

This is covered using concrete examples in the Intel® DnX User Guide.

4.4 Reading of Token

4.4.1 Using Intel® PFT GUI for DnX commands

This requires a USB connection between the management console, and the target platform, and also that you have enabled the DnX option in the General Settings Dialog

Intel® PFT allows the secure token to be read back from the device.

Also note that, the device selected to read the token from should show in the "Device selection" box.

Refer to Figure 18 for a successful read and Figure 19 for an error on read token.

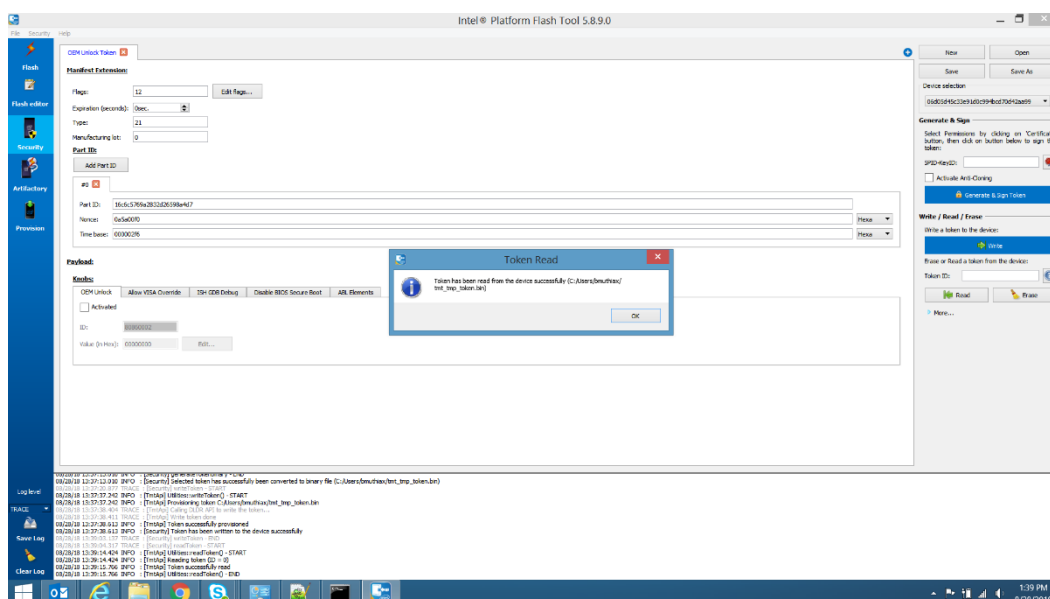


Figure 38

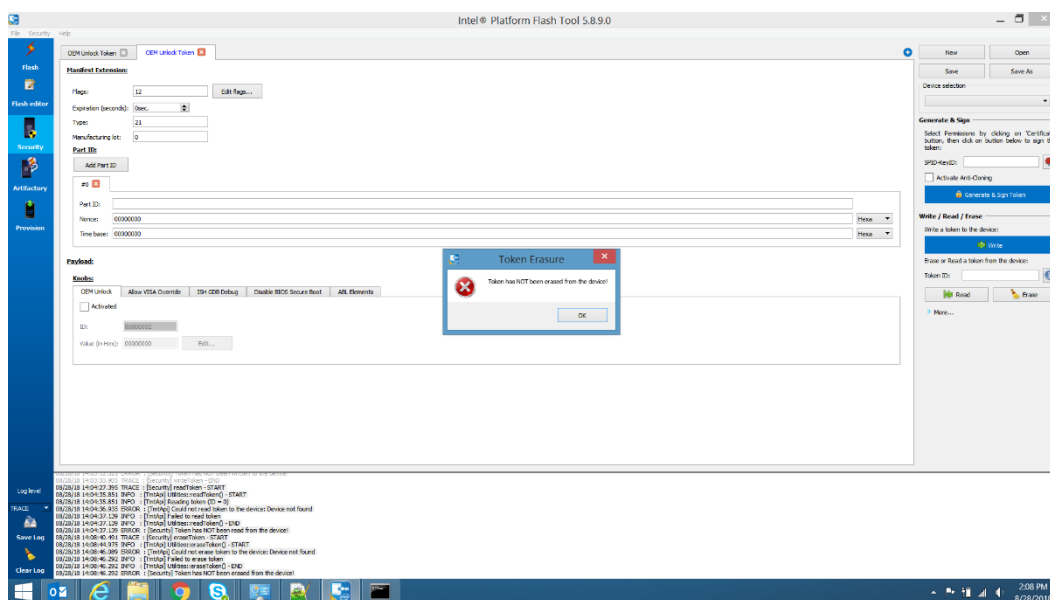


Figure 19

4.4.2 Using command line

This requires a USB connection between the management console, and the target platform, and also that you have enabled the DnX option in the General Settings Dialog.

This is covered using concrete examples in the Intel® DnX User Guide.



5 *Debugging Secure Token Injection*

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

In the event that the token is failing to unlock the platform, Intel® System Trace also known as North Peak messages must be examined, as they indicate why a token was rejected.



6 Intel® PFT options for Intel® CSME 15.0

Please note the following GUI options on the Intel® PFT are not usable on Intel® CSME 15.0 for Tokens as the Intel® PFT is a generic tool supported on multiple platforms.

6.1 Platform selection

Please use **Tigerlake** or **Rocketlake** only as the platform to be used.

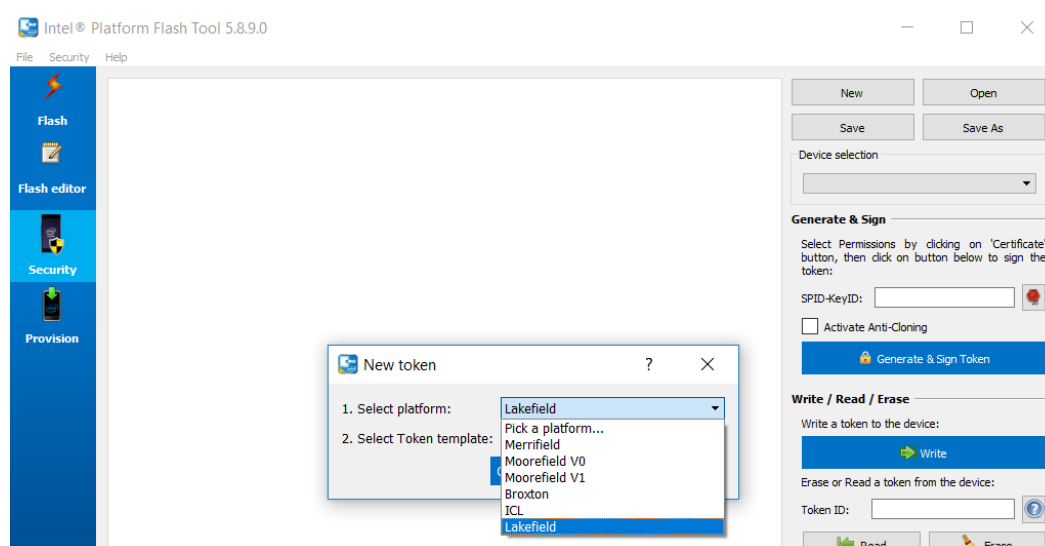


Figure 20



6.2 Token template selection

Please use **OEM Unlock Token** as the Token template.

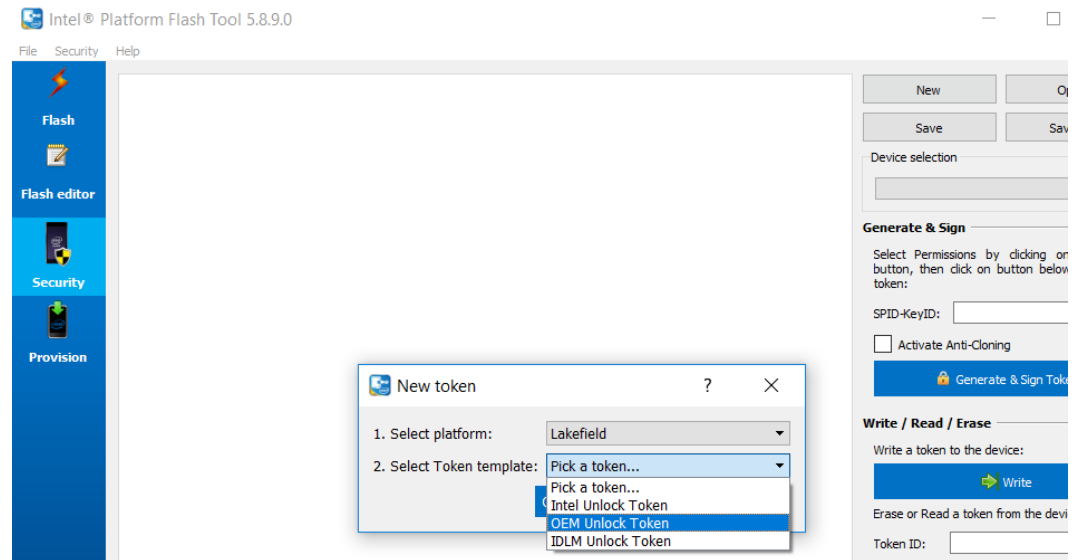


Figure 21

6.3 Miscellaneous

Highlighted in below Figure are the options not applicable for Intel® CSME 15.0

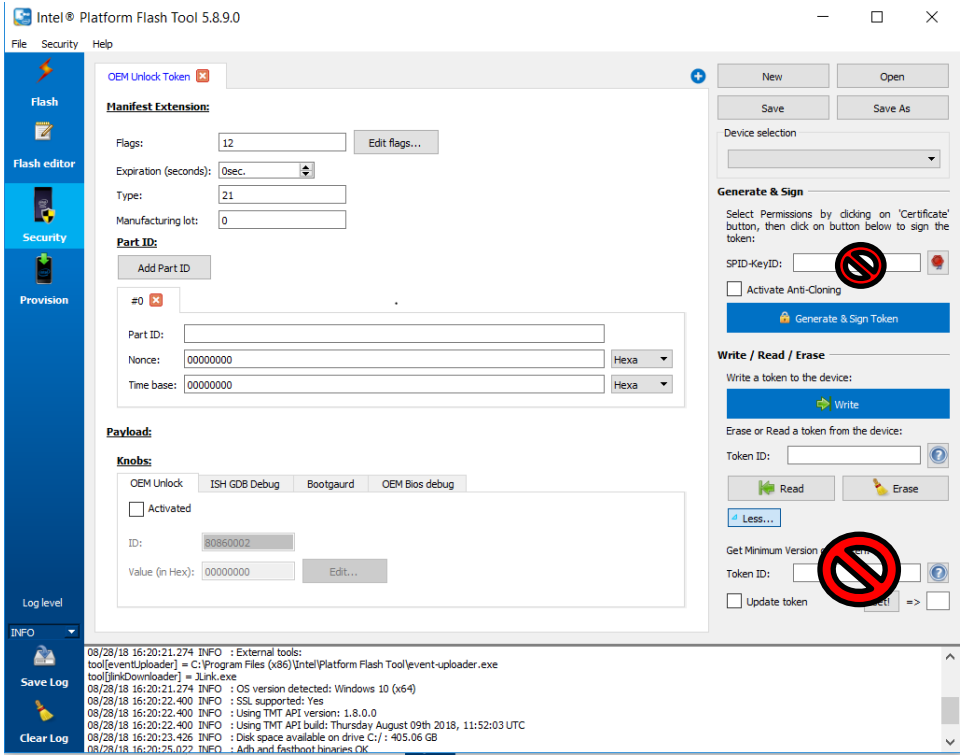


Figure 22



7 References

Document	Source
575021-575021-cn1-oem-secure-tokens-ug-rev0p5.pdf	https://www.intel.com/content/www/us/en/design/resource-design-center.html
PFT_Security_User_Guide.pdf	Intel® Platform Flash tool
Intel® DnX User Guide	VIP