Security User Guide

Intel FPGA Programmable Acceleration Card N3000 Variants

Online Version
Send Feedback

UG-20261

ID: 683519
Version: 2020.09.08

Contents

1. Overview

1.1. About This Document

Reference this user guide to understand and enable the security features such as Root of Trust (RoT) and FPGA static region (SR) user image signing for all Intel FPGA Programmable Acceleration Card N3000 variations:

- Intel® FPGA PAC N3000-1
- Intel FPGA PAC N3000-2
- Intel FPGA PAC N3000-N

Note: The Intel Arria® 10 in the Intel FPGA PAC N3000 contains a static image. No partial reconfiguration is supported. Thus, any references to FPGA SR image, flat image, or AFU image in context of an Intel FPGA PAC N3000 design is part of the static FPGA design.

Note: References to Intel FPGA PAC N3000 in this document apply to all three variants unless otherwise specified.

1.2. Prerequisites

You must ensure that the host and the Intel FPGA PAC N3000 are using the current version of OPAE tools. Please refer to the latest versions of the *Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000* and *Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000-N* for directions on how to determine if you have the current version of tools.

Related Information

- Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000
- Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000-N

1.3. Related Documentation

Refer to the following documentation while using this guide:

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Table 1. Related Documentation

Document	Description
Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000	Describes how to install and update OPAE and FPGA SR user image.
Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000-N	
Intel FPGA Programmable Acceleration Card N3000 Board Management Controller User Guide	Details features of the board management controller (BMC) not related to security, such as sensor monitoring through PLDM commands.
Intel FPGA Programmable Acceleration Card N3000-N Board Management Controller User Guide	

1.4. Glossary

Table 2. Glossary

Acronym/Term	Expansion	Description
AFU	Accelerator Functional Unit	Hardware Accelerator implemented in FPGA logic which offloads a computational operation for an application from the CPU to improve performance.
CCI-P	Core Cache Interface	CCI-P is the standard interface AFUs use to communicate with the host.
CSK	Code Signing Key	A key used to validate integrity and authenticity of a block of code. Authenticity of this key is established through signing with a root key.
ECDSA	Elliptical Curve Digital Signature Algorithm	An algorithm based on elliptic curve cryptography to create signatures that can be used to evaluate the authenticity of an object.
FIU	FPGA Interface Unit	FIU is a platform interface layer that acts as a bridge between platform interfaces like PCIe* and AFU-side interfaces such as CCI-P.
FIM	FPGA Interface Manager	The FPGA functional block containing the FPGA Interface Unit (FIU) and external interfaces for memory, networking, etc. The FIM may also be referred to as BBS (Blue-Bits, Blue Bit Stream) in the Acceleration Stack installation directory tree and in source code comments. The Accelerator Function (AF) interfaces with the FIM at run time.
HSM	Hardware Security Module	A secure hardware device to hold, protect, and allow access to cryptographic objects; performs cryptographic operations in a trusted environment.
NIST p Curve	National Institute of Standards and Technology prime Curve	P256 is used throughout this document. Without any other associations added, P256 means NIST P256 curves, where p is a 256-bit prime.
OPAE	Open Programmable Acceleration Engine	The OPAE is a software framework for managing and accessing AFs.
PACSign	PAC image signing tool	A standalone tool to manage root entry hash bitstream creation, image signing, and cancellation bitstream creation
PKCS	Public Key Cryptography Standard	PKCS#11 is used throughout this document. PKCS#11 is a commonly used interface for commercial hardware security modules (HSMs).

continued...

Acronym/Term	Expansion	Description
PR	Partial Reconfiguration	The ability to dynamically reconfigure a portion of an FPGA while the remaining FPGA design continues to function.
Root Key	-	A key designated as the primary, constant value for authentication. Typically only used to sign other keys, forming the root of all key chains.
RoT	Root of Trust	A source that can be trusted, such as the TCM in the Intel FPGA PAC.
RSU	Remote System Update	Ability to update firmware and FPGA bitstreams over PCIe.
SR	Static Region	Portion of the FPGA design that does not change.

2. Intel FPGA PAC Security Features

The Intel MAX® 10 board management controller (BMC) acts as a Root of Trust (RoT) and enables the secure update features of the Intel FPGA PAC. The RoT includes features that may help prevent the following:

- Loading or executing of unauthorized code or designs.
- Disruptive operations attempted by unprivileged software, privileged software, or the host BMC.
- Unintended execution of older code or designs with known bugs or vulnerabilities by enabling the BMC to revoke authorization.

The Intel FPGA PAC BMC also enforces several other security policies relating to access through various interfaces, as well as protecting the on-board flash through write rate limitation.

Note: The terms BMC or BMC RoT refer to the Intel FPGA PAC's Intel MAX 10 BMC (as opposed to another BMC, such as the host or motherboard BMC) unless otherwise noted.

The BMC verifies Intel MAX 10 BMC Nios® firmware and Intel MAX 10 FPGA images

The Intel FPGA PAC N3000 BMC RoT is programmed with Intel root entry hashes for BMC firmware, and BMC RTL images during a one-time secure update (OTSU) to preproduction units or at manufacturing, but does not contain a root entry hash for AFUs.

Note: This operation cannot be reversed, and after this operation, AFUs without correct signatures are refused by the Intel FPGA PAC N3000 Intel MAX 10 RoT. A correct signature is one created by a Code Signing Key (CSK) that is both signed by the root key and not yet canceled.

In cases where you have a pre-security production Intel FPGA PAC, you must perform a one-time secure update. For more information, refer to Appendix B Section B.2: *Upgrading from 1.1 Alpha-2 or Older to Production Version* in the *Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000*.

Related Information

- Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000
- Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000-N

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

2.1. Secure Image Updates

The Intel MAX 10 BMC RoT requires that all BMC Nios firmware and Intel MAX 10 FPGA images are authenticated using ECDSA before loading and executing on the card. The RoT achieves this by storing a root entry hash bitstream for the corresponding image in a write-once location and subsequently verifying the signature of the image against the hash. Intel provides the root entry hash for the BMC Nios firmware and Intel MAX 10 FPGA images. You create and program the root entry hash bitstream for your FPGA SR user image. Until you program the FPGA SR user image root entry hash bitstream, the Intel FPGA PAC does not authenticate an FPGA SR user image prior to loading and executing the image.

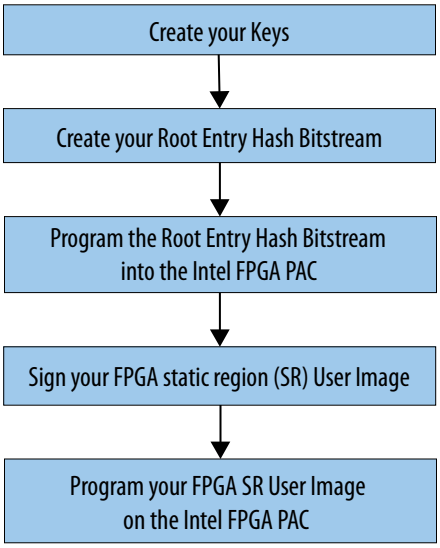
Table 3. Keys and Authentication

Root Key	Origin	Used to Authenticate
Intel MAX 10 BMC root key	Intel	Intel MAX 10 Images and Firmware
FPGA static region (SR) root key	Customer	FPGA SR User Images

When you are in the development or validation phase and have not programmed your root entry hash bitstream, you create a FPGA SR user image that contains the appropriate headers but is not signed using keys. This process is called creating an unsigned image. An Intel FPGA PAC that has not had the FPGA SR user image root entry hash programmed runs any unsigned or signed image. This capability allows you to test and validate the functionality of your FPGA SR user image prior to fully signing the image for deployment into a production environment. Please refer to the *Creating Unsigned Images* section for more information.

You program your FPGA SR user image root entry hash bitstream to enable image authentication. This process establishes you as the owner of the Intel FPGA PAC N3000. The Intel FPGA PAC N3000 then requires you to create signatures based on this root entry hash for each image you intend to load on the Intel FPGA PAC. Intel strongly recommends that you program the root entry hash bitstream for Intel FPGA PACs used in production environments. You must follow this flow to enable FPGA SR user image authentication on your Intel FPGA PAC.

Figure 1. Secure User Image Flow



The chapters within this user guide cover the steps in this flow:

1. **Create your keys:** Create your keys using a Hardware Security Module (HSM) or OpenSSL. You need at least two keys, one which you designate as a root key and another you designate as a code signing key (CSK). These keys are asymmetric keys, meaning they consist of an underlying pair of keys. The first is called a private key and the second is a public key that is derived from the private key. A private key is used to create signatures over objects that can be verified with the corresponding public key. The private key must be kept confidential, as anyone in possession of the private key can create a signature; conversely, if you maintain the confidentiality of the private key, then signatures can be trusted to originate only from you. The public key cannot create signatures or be used to derive the original private key. Therefore, it is not required nor important to protect the confidentiality of the public key; the public key is considered public information.
2. **Create your root entry hash bitstream:** Use the PACSign tool to create a bitstream that contains the root entry hash. You create a root entry hash bitstream from your root public key. This hash is a representation of your root public key and can only be created with an exact copy of the root public key. The root entry hash bitstream is then programmed to the Intel FPGA PAC. The Intel FPGA PAC then uses this hash to verify the integrity of the root public key, which is included with all images transmitted to an Intel FPGA PAC. After the integrity of the root public key is confirmed, it can be used in the signature verification process.
3. **Program your root entry hash bitstream into the Intel FPGA PAC**. You must use the `fpgasupdate` command to program the bitstream containing your root entry hash into the flash on the board. Until you program the root entry hash bitstream, the Intel FPGA PAC loads and executes any signed or unsigned image. Intel strongly recommends that you create and program a root entry hash bitstream for Intel FPGA PACs deployed in production environments. Please refer to the *Using fpgasupdate* chapter for more information.

Note: Only the owner who is deploying the Intel FPGA PAC must program the root entry hash bitstream.

4. **Sign your FPGA SR user image.** Using PACSign you can sign your image with the root public key and code signing key. Please refer to the *Using PACSign* chapter for more information.
5. **Program your FPGA SR user image onto the Intel FPGA PAC.** Use the `fpgasupdate` command to program your FPGA SR user image into flash. Then use the `rsu` command to configure your FPGA. The BMC verifies the FPGA SR user image to ensure only an authorized bitstream is loaded on the Intel FPGA PAC. The root public key, code signing public key, signature of the code signing public key, and signature of the image are all attached to the code or design transmitted to the Intel FPGA PAC. The card first verifies the integrity of the root public key, then verifies the signature of the code signing public key using the root public key, and finally proceeds to verify the signature of the code or design using the code signing public key. The code or design is only accepted if all three of these steps are completed successfully.

2.2. Anti-Rollback Capability

The Intel MAX 10 BMC RoT provides anti-rollback capability through the code signing key ID cancellation feature. A CSK is assigned an ID, a number between 0-127, during the signing process. CSK ID cancellation information is stored in 128-bit fields in write-once locations in flash. When a code signing key ID is canceled, the Intel MAX 10 BMC RoT rejects all signatures created with a CSK that is assigned that ID. If a CSK ID that is used in an old update is canceled after applying a new update with a different CSK ID, the Intel MAX 10 BMC RoT rejects the signature of the old update, preventing a rollback to the old update.

Note: If you cancel a key and do not update your image with a different CSK ID, the old image continues to be operational unless the user updates it with the new image signed with a different CSK ID.

2.3. Key Management

The Intel MAX 10 BMC RoT uses ECDSA with a key length of 256 bits to authenticate:

- Intel MAX 10 BMC Nios firmware and Intel MAX 10 FPGA images
- FPGA static region (SR) user image

The Intel MAX 10 BMC RoT supports separate key chains for each image, and each key chain must consist of a root key and a CSK.

The Intel MAX 10 BMC RoT does not support a signature of any image with a root key. You must use a key designated as a CSK to sign your image. Steps you are responsible for when creating keys, root entry hashes and programming your image on the Intel FPGA PAC are:

- You must manage assigning CSK IDs to CSKs and consistently using the same ID for a given CSK. Neither an Intel FPGA PAC nor the PACSign tool associate a particular key's value with its ID. It is possible to assign a given CSK multiple IDs, or multiple CSKs to a given ID. This may result in unintended consequences when attempting to cancel a CSK. Intel recommends exclusive ID assignments for each CSK.
- You are responsible for creating the appropriate key cancellation bitstreams. You must use the same ID number for key cancellation as the one you assigned to the CSK at key creation. Key cancellation bitstreams must be signed with the applicable root key. This helps avoid denial of service through an unintended cancellation of all key values.
- You are responsible for generating and managing your FPGA static region image root key and CSKs. You generate the FPGA SR user image root entry hash bitstream using your root key.
- You are also responsible for programming this root entry hash bitstream on the Intel FPGA PAC. If your Intel FPGA PAC does not have a programmed FPGA SR user image root entry hash bitstream stored, it executes any signed or unsigned image.

  Note: Intel strongly recommends programming an image root entry hash bitstream. You must protect the confidentiality of the root private key throughout the life of the Intel FPGA PAC.

The Intel MAX 10 BMC RoT bitstreams in the on-board flash for:

1. BMC Nios firmware and Intel MAX 10 FPGA images
2. FPGA SR user image

The BMC is architected so that all root entry hashes cannot be revoked, changed, or erased once programmed.

In the future, Intel-provided updates to the Intel MAX 10 BMC firmware or Intel MAX 10 images may necessitate an Intel key cancellation in order to help prevent an unintended rollback to a prior version. In this case, Intel provides the update with a signed CSK that has a different ID than all prior updates. Intel provides a separate key cancellation bitstream to cancel the appropriate Intel keys. You may test an update by applying it before programming the key cancellation bitstream. The prior BMC firmware or update images continue to be accepted as valid updates until the new key cancellation bitstream is applied.

2.4. Authentication

To enable authentication:

1. Use the PACSign tool to create a root entry hash bitstream.
2. Use the `fpgasupdate` tool to program the bitstream onto the Intel FPGA PAC.

   ```
   $ sudo fpgasupdate [--log-level=<level>] file [bdf]
   ```

3. Power cycle your card to load the new bitstream by running the following command:

   ```
   $ sudo rsu bmcimg 3e:00.0
   ```

   Note: In this example, the [bdf] is `3e:00.0`. You must provide the BDF assigned to the `PCIe DevID 0b30` on your system.

All key operations are done using PACSign. PACSign is a standalone tool that is not required to be run on a machine with the Intel FPGA PAC installed. Key creation, signing, and cancellation bitstream creation are not runtime operations and can be performed at any time. The signing process prepends the signature to the FPGA SR User image file. The BMC RoT does not need access to the HSM at any point to verify a signature.

The signing process requires a root key and a Code Signing Key (CSK). PACSign first signs the CSK with the root key, and then signs the image with the CSK. The signature process prepends two "blocks" of data to the image file.

Note: If you are using an Intel Acceleration Stack version 1.1 production or greater, your FPGA SR user image must have prepended signature blocks, even if the corresponding root entry hash bitstream has not been programmed. PACSign allows you to prepend the required blocks with an empty signature chain.

2.5. Encryption

FPGA SR user image encryption is not supported on the Intel FPGA PAC N3000.

3. Intel FPGA PAC Security Flow

The following steps describe the flow to enable Intel FPGA PAC security. See the corresponding sections in this chapter for detailed instructions on each step.

1. **Install PACSign.**
2. If you are in development, you may optionally create an unsigned FPGA SR user image to test and validate the functionality of your image prior to fully signing the image for deployment into a production environment. Please refer to the *Creating Unsigned Images* section for more information.
3. **Create your root key and CSK(s). You can use OpenSSL or an HSM for this action.**

Figure 2. Key Creation Using OpenSSL

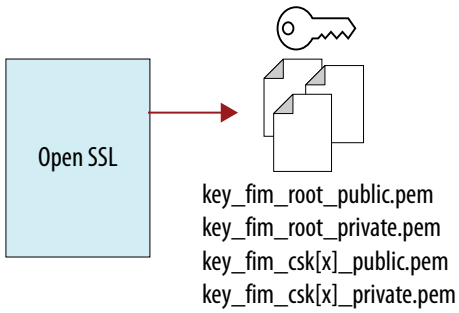
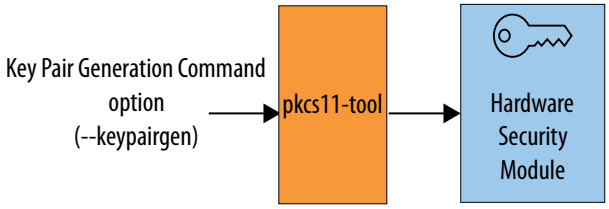


Figure 3. Key Creation Using HSM pkcs11_tool



4. **Create your root entry hash bitstream.**

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 4. Creating Root Entry Hash Bitstream with OpenSSL

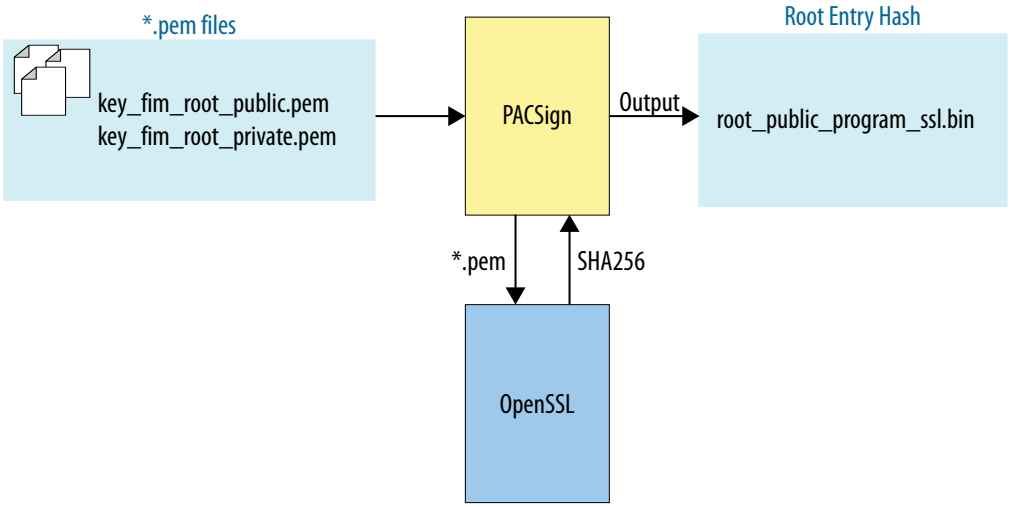
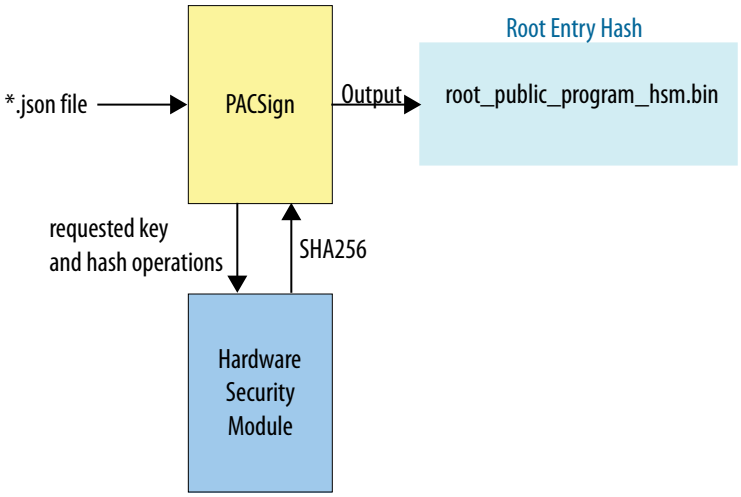


Figure 5. Creating Root Entry Hash Bitstream with HSM pkcs11_manager



5. **Program your root entry hash bitstream onto the Intel FPGA PAC.**
6. **Sign your FPGA SR user image.**

Figure 6. Signing your image with OpenSSL

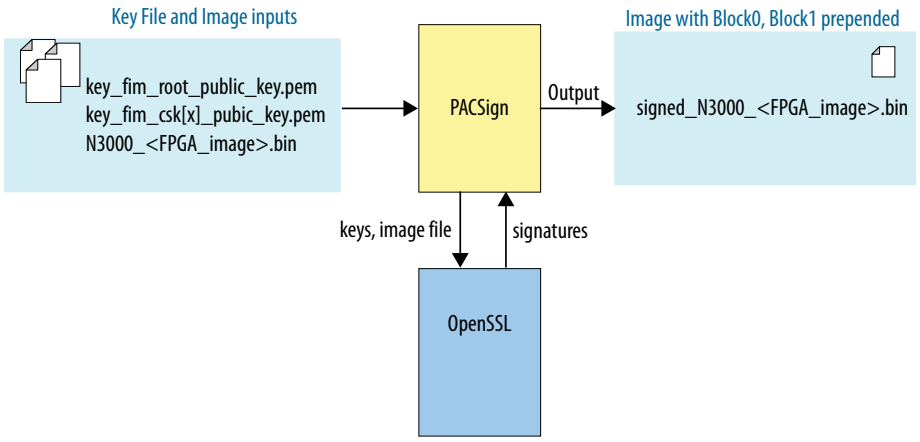
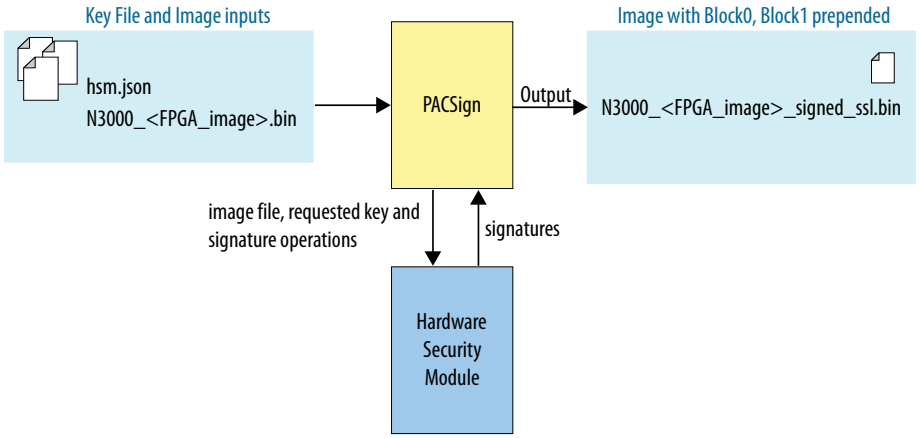


Figure 7. Signing your image with pkcs11_manager



7. **Program your FPGA SR user image into the Intel FPGA PAC.** For directions on how to program your FPGA SR user image, refer to the *Using fpgasupdate* chapter.

Related Information

Using fpgasupdate on page 35

3.1. Installing PACSign

PACSign is a standalone tool that interfaces with your HSM to manage root entry hash bitstream creation, image signing, and cancellation bitstream creation. PACSign is implemented in Python and requires Python 3. Using PACSign with the PKCS11 interface requires the python-pkcs11 package. PACSign does not need an Intel FPGA PAC installed in the system to run. Systems where signed images are being deployed to an Intel FPGA PAC do not need PACSign installed nor access to the HSM.

Note: You must install Python 3 to use PACSign.

Note: The Acceleration Stack includes the PACSign package. You can check if you already have this package by typing: `rpm -qa| grep opae`.

1. Unpack the `opae.pac_sign-1.0.1.tar.gz` tarball, which contains the `opae.pac_sign-1.0.4-2.x86_64.rpm` package.

   ```
   sudo yum install opae.pac_sign-1.0.4-2.x86_64.rpm
   ```

   You can use the RTE installer with this command to extract and just install PACSign:

   ```
   ./n3000-1.3.8-rte-setup.sh -t pacsign -n
   ~/n3000_ias_1_1_pv_rte_installer
   Running setup
   Do you wish to install OPAE PACSign ?
   ```

2. Ensure you have installed Python 3, the Python 3 development libraries, and the Python 3 version of the python-pkcs11 package on your system.
3. Use your system package installer to install the .rpm package. `PACSign` installs to your `/usr/local/bin` directory and the necessary Python3.6 modules install to your `/usr/local/lib` directory.

   Note: `PACSign` depends on a Python3 interpreter version 3.6 or later. You must either install Python3 to, or create a symlink in, `/usr/local/bin` for PACSign to work. You must also ensure that the python modules PACSign depends on are visible to your python3 interpreter. You can do this by including the path `/usr/local/lib/python3.6/site-packages/` in the `PYTHONPATH` environment variable.

   ```
   export PYTHONPATH=/usr/local/lib/python3.6/site-packages/
   ```

3.2. PACSign Tool

The PACSign utility is installed on your path.

- Use PACSign by simply calling it directly with the command `PACSign`
- Calling `PACSign` with the `-h` option shows a help message describing the tool usage.
- Typing `PACsign <image_type> -h` shows options available for that image type.

```
[PACSign_Demo]$ PACSign -h
usage: PACSign [-h] {SR,FIM,BBS,BMC,BMC_FW,PR,AFU,GBS} ...

Sign PAC bitstreams

optional arguments:
-h, --help show this help message and exit

Commands:
Image types
{SR,FIM,BBS,BMC,BMC_FW,PR(1),AFU,GBS}
Allowable image types
SR (FIM, BBS)   Static FPGA image
BMC (BMC_FW)    BMC image
PR (AFU, GBS)   Reconfigurable FPGA image

[PACSign_Demo]$ PACSign AFU -h
usage: PACSign SR [-h] -t {UPDATE,CANCEL,RK_256,RK_384} -H HSM_MANAGER
                  [-C HSM_CONFIG] [-r ROOT_KEY] [-k CODE_SIGNING_KEY]
                  [-d CSK_ID] [-i INPUT_FILE] [-o OUTPUT_FILE] [-y] [-v]
```

(1) Intel FPGA PAC N3000 does not support PR

```
optional arguments:
  -h, --help            show this help message and exit
  -t {UPDATE,CANCEL,RK_256,RK_384}, --cert_type {UPDATE,CANCEL,RK_256,RK_384}
                        Type of certificate to generate
  -H HSM_MANAGER, --HSM_manager HSM_MANAGER
                        Module name for key / signing manager
  -C HSM_CONFIG, --HSM_config HSM_CONFIG
                        Config file name for key / signing manager (optional)
  -r ROOT_KEY, --root_key ROOT_KEY
                        Identifier for the root key. Provided as-is to the key
                        manager
  -k CODE_SIGNING_KEY, --code_signing_key CODE_SIGNING_KEY
                        Identifier for the CSK. Provided as-is to the key
                        manager
  -d CSK_ID, --csk_id CSK_ID
                        CSK number. Only required for cancellation certificate
  -i INPUT_FILE, --input_file INPUT_FILE
                        File name for the image to be acted upon
  -o OUTPUT_FILE, --output_file OUTPUT_FILE
                        File name in which the result is to be stored
  -y, --yes             Answer all questions with "yes"
  -v, --verbose         Increase verbosity. Can be specified multiple times
```

3.3. Creating Unsigned Images

The BMC secure firmware does not accept an FPGA SR user image without the prepended authentication blocks generated by PACSign, even if an FPGA SR user image root entry hash bitstream has not been programmed. If you want to operate an Intel FPGA PAC without a root entry hash bitstream programmed, such as in a development environment, you must still use PACSign to prepend the authentication blocks but you may do so with an empty signature chain. An image with prepended authentication blocks containing an empty signature chain is called an unsigned image. PACSign supports the creation of an unsigned image by using the UPDATE operation without specifying keys. Intel recommends using signed images in production deployments.

1. Create unsigned bitstream.

   Using OpenSSL:

   ```
   [PACSign_Demo]$ PACSign SR -t UPDATE -H openssl_manager -i pac-n3000-secure-
   update-raw.bin -o unsigned_N3000_RSU.bin
   ```

   Using HSM:

   ```
   [PACSign_Demo]$ PACSign SR -t UPDATE -H pkcs11_manager -C softhsm.json -i \
   pac-n3000-secure-update-raw.bin -o pac-n3000-secure-update-raw.bin
   ```

   The output prompts you to enter Y or N to continue generating an unsigned bitstream.

   ```
   No root key specified. Generate unsigned bitstream? Y = yes, N = no: Y
   No CSK specified. Generate unsigned bitstream? Y = yes, N = no: Y
   ```

2. Program the unsigned bitstream.

   ```
   [PACSign_Demo]$ sudo fpgasupdate pac-n3000-secure-update-raw.bin b2:00.0
   ```

3. Perform remote system update to power cycle the Intel FPGA PAC N3000 and load the updated image into the FPGA.

   ```
   [PACSign_Demo]$ sudo rsu bmcimg b2:00.0
   ```

3.4. Using an HSM Manager

The PACSign tool does not implement any cryptographic functions. PACSign must interact with a cryptographic service, and it does this through modules called Hardware Security Module (HSM) managers. PACSign provides the following managers:

- openssl_manager: interfaces with OpenSSL
- pkcs11_manager: interfaces with any HSM implementing PKCS#11

Use the `-H` option with the `PACSign` command to select an HSM manager. The following sections provide examples for the PACSign OpenSSL manager using OpenSSL v1.1.1d, and the PACSign PKCS #11 manager using SoftHSM v2.5.0. Examples of key creation and management with both OpenSSL and SoftHSM (through the utilities softhsm2-util and pkcs11-tool) are also provided. To create your own custom HSM manager, refer to the *Custom HSM Manager Creation* topic more information.

Related Information

Creating a Custom HSM Manager on page 24

3.5. Creating Keys

Create your root and code signing keys using your desired key management utility (HSM or OpenSSL). Assign your key CSK IDs during key creation. Intel recommends that you consistently use the same ID for a given key across all image signings.

3.5.1. OpenSSL Key Creation

When using OpenSSL, create a private key and then create the corresponding public key. The PACSign OpenSSL manager requires specific tags in the key file names using a format: *key_<image_type>_<key_type>_<key_visibility>_key.pem*.

Table 4. PACSign OpenSSL Manager Key File Name Requirements

Filename Tag	Options	Description
image_type	• `pr` • `fim`	Identifies image type, partial reconfiguration or static region, for which the key is intended. • For Intel FPGA PAC N3000, use; `key_fim_<key_type>_<key_section>_key.pem`
key_type	• `root` • `csk<x>`	Identifies key type. <x> specifies an ID that you use for cancellation. • Example: `key_fim_csk12_private_key.pem`
key_visibility	• `public` • `private`	Identifies the key visibility.

The following example creates a root key and two code signing keys using OpenSSL.

1. Create the root private key:

```
[PACSign_Demo]$ openssl ecparam -name secp256r1 -genkey -noout -out
key_fim_root_private_key.pem
```

Output:

```
using curve name prime256v1 instead of secp256r1
```

2. Create the root public key:

```
[PACSign_Demo]$ openssl ec -in key_fim_root_private_key.pem -pubout -out
key_fim_root_public_key.pem
```

Output:

```
read EC key
writing EC key
```

3. Create private CSK1:

```
[PACSign_Demo]$ openssl ecparam -name secp256r1 -genkey -noout -out
key_fim_csk1_private_key.pem
```

Output:

```
using curve name prime256v1 instead of secp256r1
```

4. Create public CSK1:

```
[PACSign_Demo]$ openssl ec -in key_fim_csk1_private_key.pem -pubout -out
key_fim_csk1_public_key.pem
```

Output:

```
read EC key
writing EC key
```

5. Create private CSK2:

```
[PACSign_Demo]$ openssl ecparam -name secp256r1 -genkey -noout -out
key_fim_csk2_private_key.pem
```

Output:

```
using curve name prime256v1 instead of secp256r1
```

6. Create public CSK2:

```
[PACSign_Demo]$ openssl ec -in key_fim_csk2_private_key.pem -pubout -out
key_fim_csk2_public_key.pem
```

Output:

```
read EC key
writing EC key
```

3.5.2. HSM Key Creation

If you are using an HSM, you need one token to create and store the root and code signing keys. The following example initializes a token using SoftHSM, with separate security officer and user PINs.

```
[PACSign_Demo]$ softhsm2-util --init-token --label pac-hsm --so-pin hsm-owner \
--pin pac-afu-signer --free
```

Output:

```
Slot 0 has a free/uninitialized token.
The token has been initialized and is reassigned to slot 1441483598
```

After you create a token, you can create keys in that token. The following example initializes a root and two code signing keys in the token created above, similarly using pkcs11-tool to interact with SoftHSM. The HSM, not PACSign, uses the key ID provided in this example. PACSign uses CSK IDs from a configuration `*.json` file in PKCS11 mode. You must manage consistency across ID values in the HSM and those used by PACSign. See the *PACSign PKCS11 Manager *.json Reference* topic for more information on the `*.json` file format.

1. Initialize the root key:

```
[PACSign_Demo]$ pkcs11-tool --module=/usr/local/lib/softhsm/libsofthsm2.so \
--token-label pac-hsm --login --pin pac-afu-signer --keypairgen \
--mechanism ECDSA-KEY-PAIR-GEN --key-type EC:secp256r1 \
--usage-sign --label root_key --id 0
```

   Output:

```
Key pair generated:
Private Key Object; EC
label: root_key
ID: 00
Usage: decrypt, sign, unwrap
Public Key Object; EC EC_POINT 256 bits
EC_POINT:
0441043d3756347e6c257dac085574cc1cd984cdeee2c1059a0f035dabc3ad6e1950c8717dc7
ac8451a90c2471e95f4a69d6517f02f678830280f90a479c76a3e95d64
EC_PARAMS: 06082a8648ce3d030107
label: root_key
ID: 00
Usage: encrypt, verify, wrap
```

2. Initialize the CSK1:

```
[PACSign_Demo]$ pkcs11-tool --module=/usr/local/lib/softhsm/libsofthsm2.so \
--token-label pac-hsm --login --pin pac-afu-signer --keypairgen \
--mechanism ECDSA-KEY-PAIR-GEN --key-type EC:secp256r1 \
--usage-sign --label csk_1 --id 1
```

   Output:

```
Key pair generated:
Private Key Object; EC
label: csk_1
ID: 01
Usage: decrypt, sign, unwrap
Public Key Object; EC EC_POINT 256 bits
EC_POINT:
0441041a827c903b5da8478c81fe652208704f0621b984190cd961ee154ac5c3ba772d1caa26
964a189262ee31b8e5d77898f293c0589b350103037b664d31adf68924
EC_PARAMS: 06082a8648ce3d030107
label: csk_1
ID: 01
Usage: encrypt, verify, wrap
```

3. Initialize CSK2:

```
[PACSign_Demo]$ pkcs11-tool --module=/usr/local/lib/softhsm/libsofthsm2.so \
--token-label pac-hsm --login --pin pac-afu-signer --keypairgen \
--mechanism ECDSA-KEY-PAIR-GEN --key-type EC:secp256r1 \
--usage-sign --label csk_2 --id 2
```

   Output:

```
Key pair generated:
Private Key Object; EC
label: csk_2
ID: 02
```

```
Usage: decrypt, sign, unwrap
Public Key Object; EC EC_POINT 256 bits
EC_POINT:
04410495f7556912d8753cf873be7a54e7d88c28bca672496abd90d9b44cc95cf50df9169b7a
d043a7340003a2bf96cb461e0575319b541ceb5d873d06334b30d208cc
EC_PARAMS: 06082a8648ce3d030107
label: csk_2
ID: 02
Usage: encrypt, verify, wrap
```

4. After keys are created in your token, it may be useful to inspect the token to verify the expected keys, labels, and IDs are present.

```
[PACSign_Demo]$ pkcs11-tool --module=/usr/local/lib/softhsm/libsofthsm2.so \
--token-label pac-hsm --login --pin pac-afu-signer -O
```

Output:

```
Public Key Object; EC EC_POINT 256 bits
EC_POINT:
04410495f7556912d8753cf873be7a54e7d88c28bca672496abd90d9b44cc95cf50df9169b7a
d043a7340003a2bf96cb461e0575319b541ceb5d873d06334b30d208cc
EC_PARAMS: 06082a8648ce3d030107
label: csk_2
ID: 02
Usage: encrypt, verify, wrap
Public Key Object; EC EC_POINT 256 bits
EC_POINT:
0441043d3756347e6c257dac085574cc1cd984cdeee2c1059a0f035dabc3ad6e1950c8717dc7
ac8451a90c2471e95f4a69d6517f02f678830280f90a479c76a3e95d64
EC_PARAMS: 06082a8648ce3d030107
label: root_key
ID: 00
Usage: encrypt, verify, wrap
Private Key Object; EC
label: root_key
ID: 00
Usage: decrypt, sign, unwrap
Private Key Object; EC
label: csk_2
ID: 02
Usage: decrypt, sign, unwrap
Private Key Object; EC
label: csk_1
ID: 01
Usage: decrypt, sign, unwrap
Public Key Object; EC EC_POINT 256 bits
EC_POINT:
0441041a827c903b5da8478c81fe652208704f0621b984190cd961ee154ac5c3ba772d1caa26
964a189262ee31b8e5d77898f293c0589b350103037b664d31adf68924
EC_PARAMS: 06082a8648ce3d030107
label: csk_1
ID: 01
Usage: encrypt, verify, wrap
```

Related Information

PACSign PKCS11 Manager *.json Reference on page 23

3.6. Root Entry Hash Bitstream Creation

In order to program the root entry hash bitstream to an Intel FPGA PAC, you must use PACSign to create a root entry hash bitstream.

1. In your PACSign command, specify the type RK_256 and select the appropriate HSM manager and configuration.

- To create a root entry hash bitstream using OpenSSL and the key generated in the *OpenSSL Key Creation* topic, type:

```
[PACSign_Demo]$ PACSign SR -t RK_256 -H openssl_manager -r
key_fim_root_public_key.pem -o root_public_program_ssl.bin
```

- To create a root entry hash bitstream using a SoftHSM and the root key generated in the *HSM Key Creation* topic, type:

```
[PACSign_Demo]$ PACSign SR -t RK_256 -H pkcs11_manager -C softhsm.json -
r root_key -o root_public_program_hsm.bin
```

Note: PACSign requires an HSM configuration `*.json` file to request the correct key from the HSM. For more information about the structure and contents of the `*.json` file, refer to the *PACSign PKCS11 Manager .json Reference* topic.

2. After creating the root entry hash bitstream, program the bitstream to an Intel FPGA PAC using the `fpgasupdate` command.

```
[PACSign_Demo]$ sudo fpgasupdate <root entry hash bitstream> b2:00.0
```

This operation is permanent and irreversible. After a FPGA SR user image root entry hash bitstream is programmed, the Intel FPGA PAC validates a FPGA SR user image signature prior to loading. For more details on key management, see the *Key Management* topic. For more information on how to use `fpgasupdate`, refer to the *Using `fpgasupdate`* section.

3. After you program the root entry hash bitstream, power cycle your Intel FPGA PAC.

```
[PACSign_Demo]$ sudo rsu bmcimg b2:00.0
```

3.7. Signing Images

After the root and code signing keys have been created, you may sign your FPGA SR user image. Use the appropriate SR bitstream type with the UPDATE identifier to perform this operation, and specify the HSM configuration, root key, code signing key, and image input and output file names.

The following example demonstrates image signing using OpenSSL and the root and code signing keys generated in *OpenSSL Key Creation* topic.

```
[PACSign_Demo]$ PACSign SR -t UPDATE -H openssl_manager -r
key_fim_root_public_key.pem -k key_fim_csk1_public_key.pem -i hello_afu.bin -o
hello_afu_signed_ssl.bin
```

Note: Even though public keys are specified in the above OpenSSL signing process, the bitstream is indeed signed with the private keys. The OpenSSL signing requires the private keys and they must be the same name with 'public' replaced with 'private'. The reason public keys are specified is because private keys are usually maintained by an HSM and are not available to you.

The following example demonstrates image signing using SoftHSM PKCS11 and the root and code signing keys generated in *HSM Key Creation* topic. Using this method, you must create a `softhsm.json` file. Refer to the *PACSign PKCS11 Manager .json Reference* topic for more information on the `*.json` file.

```
[PACSign_Demo]$ PACSign SR -t UPDATE -H pkcs11_manager -C softhsm.json -r
root_key -k csk_1 -i hello_afu.bin -o hello_afu_signed_hsm.bin
```

You can program signed bitstreams on your Intel FPGA PAC by using the `fpgasupdate` tool and power cycle the card.

```
[PACSign_Demo]$ sudo fpgasupdate <signed bitstream> B:D.F
[PACSign_Demo]$ sudo rsu bmcimg B:D.F
```

An Intel FPGA PAC only authenticates signed bitstreams after a root entry hash bitstream has been programmed. An Intel FPGA PAC that has not been programmed with a root entry hash bitstream accepts a signed bitstream and ignores the contents of the signature chain.

If your `fpgasudate` fails, refer to section Troubleshooting *on page 36* for guidance on interpretation of the error and for corrective action.

Related Information

- OpenSSL Key Creation on page 17
- HSM Key Creation on page 18
- PACSign PKCS11 Manager *.json Reference on page 23
- Using fpgasupdate on page 35
- Troubleshooting on page 36

3.8. Creating a CSK ID Cancellation Bitstream

To cancel a CSK ID on an Intel FPGA PAC, you must use PACSign to create a CSK ID cancellation bitstream. To do this, you must specify the type `CANCEL`, select the appropriate HSM manager and root key, and provide the CSK ID number to cancel. For OpenSSL, the CSK ID used during image signing is derived from the CSK filename. For PKCS11, the CSK ID used during image signing is extracted from the `csk_id` field in the configuration `.json` discussed in the next section.

1. Create a cancellation bitstream.

   Using OpenSSL:

   ```
   PACSign SR -t CANCEL -H openssl_manager -r key_fim_root_public_key.pem -d 1
   -o ssl_csk1_cancel.bin
   ```

   Using PKCS11:

   ```
   PACSign SR -t CANCEL -H pkcs11_manager -C softhsm.json -r root_key -d 1 -o
   hsm_csk1_cancel.bin
   ```

2. Program the CSK ID cancellation on the Intel FPGA PAC using the `fpgasupdate` tool.

   ```
   fpgasupdate ssl_csk1_cancel.bin b2:00.0
   ```

   CSK ID cancellation bitstreams are only valid on Intel FPGA PACs that have been programmed with the corresponding root entry hash bitstream.

3. After you program a CSK ID cancellation bitstream, you must power cycle the Intel FPGA PAC.

```
[PACSign_Demo]$ sudo rsu bmcimg b2:00.0
```

3.9. PACSign PKCS11 Manager *.json Reference

The PACSign PKCS11 Manager uses a `*.json` file that stores information on how to interact with your HSM.

It contains information specific to your HSM, as well as a description of the token and keys that you created for use with PACSign. The PKCS11 examples in this chapter use `softhsm.json`, which contains the following:

```
{
    "cryptoki_version": [2, 40],
    "library_version": [2, 5],
    "platform-name" : "DCP",
    "lib_path" : “/usr/local/lib/softhsm/libsofthsm2.so”,
    "curve": "secp256r1",
    "token": {
        "label": "pac-hsm",
        "user_password": "pac-afu-signer",
        "keys":
            [
                {
                    "label": "root_key",
                    "key_id": "0",
                    "type": "PR",
                    "permissions": "0xFFFFFFFF",
                    "csk_id": "0xFFFFFFFF",
                    "is_root": true
                },
                {
                    "label": "csk_1",
                    "key_id": "1",
                    "type": "PR",
                    "permissions": "0x4",
                    "csk_id": "0x1",
                    "is_root": false
                },
                {
                    "label": "csk_2",
                    "key_id": "2",
                    "type": "PR",
                    "permissions": "0x4",
                    "csk_id": "0x2",
                    "is_root": false
                }
            ]
        }
}
```

The `cryptoki_version` and `library_version` information is determined by your HSM and can be reported by pkcs11-tool:

```
[PACSign_Demo]$ pkcs11-tool --module=/usr/local/lib/softhsm/libsofthsm2.so -I
```

Output:

```
Cryptoki version 2.40
Manufacturer SoftHSM
Library Implementation of PKCS11 (ver 2.5)
Using slot 0 with a present token (0x55eb4b4e)
```

- `platform-name`: Always set to DCP.
- `lib_path`: Your HSM software library installation determines this path.
- `curve`: Always set to `secp256r1` because this is the only elliptic curve currently supported by the BMC.
- The `token` entry contains:
  - `label`: determined when you initialize the token in your HSM
  - `user_password`: determined when you initialize the token in your HSM
  - `keys`: lists the keys in the token available for use by PACSign
- Within the `key` field are:
  - `label`: determined when you initialize the token in your HSM
  - `key_id`: determined when you initialize the token in your HSM

    Note: Each `label` and `key_id` must match what you used when you created the key.
  - `type`: Either PR or SR for partial reconfiguration or static region, respectively.
  - `permissions`: Set to 0x1 for static region signing; 0x2 for BMC signing; 0x4 for partial reconfiguration region signing.
  - `csk_id`: What PACSign uses when signing a FPGA SR user image; does not need to match the `key_id` field. Valid values are `0xFFFFFFFF` for root keys and `0x0-0x7F` for Intel FPGA PAC N3000 code signing keys.
  - `is_root`: Allows you to designate to PACSign the intended use of the key as a root key or code signing key.

3.10. Creating a Custom HSM Manager

PACSign is a Python tool that uses a plugin architecture for the HSM interface. PACSign is distributed with managers for both OpenSSL and PKCS #11. This section describes the functionality required by PACSign from the HSM interface and shows how to construct a plugin.

The distribution of PACSign uses the following directory structure:

```
hsm_managers
├── openssl_manager
│   └── library
├── pkcs11_manager
source
```

The top level contains `PACSign.py` with the generic signing code in source. The HSM managers reside each in their own subdirectory under `hsm_managers` as packages. The directory name is what is given to PACSign's `--HSM_MANAGER` command-line option. If the specific manager requires additional information, you can provide it using the optional `--HSM_config` command-line option. For example, the PKCS #11 plugin requires a `*.json` file describing the tokens and keys available on the HSM.

You must place each plugin that is to be supported in a subdirectory of the `hsm_managers` directory. Use a descriptive name for the directory that clearly describes the supported HSM. This subdirectory may have an `__init__.py` file whose contents import the modules needed by the plugin. The names of the plugin modules are not important to the proper functioning of PACSign.

The newly-created plugin must be able to export one attribute named HSM_MANAGER that is invoked by PACSign with an optional configuration file name provided on the command-line. Invocation of HSM_MANAGER(`config_file`) returns a class with certain methods exposed, which are described in later sections.

Current implementations of HSM_MANAGER define it as a Python class object. The initialization function of the class reads and parses the configuration file (if present) and performs HSM initialization. For the PKCS #11 implementation, the class looks like this:

```
class HSM_MANAGER(object):
    def __init__(self, cfg_file = None):
        common_util.assert_in_error(cfg_file, \
            PKCS11 HSM manager requires a configuration file")
        self.session = None
        with open(cfg_file, "r") as read_file:
            self.j_data = json.load(read_file)
        j_data = self.j_data

        lib = pkcs11.lib(j_data['lib_path'])
        token = lib.get_token(token_label=j_data['token']['label'])
        self.session = token.open(user_pin=j_data['token']['user_password'])
        self.curve = j_data['curve']

        self.ecparams = self.session.create_domain_parameters( \
        pkcs11.KeyType.EC, {pkcs11.Attribute: \
        pkcs11.util.ec.encode_named_curve_parameters(self.curve)}, \
        local=True)
```

Error handling code has been omitted for clarity. This code does the following:

- Opens and parses the `*.json` configuration file.
- Loads the vendor-supplied PKCS #11 library into the program.
- Sets up a session with the correct token.
- Retrieves the proper elliptic curve parameters for the curve you select.

The following sections describe the required exported methods of this class.

3.10.1. HSM_MANAGER.get_public_key(public_key)

This method returns an instance of a public key that is described by 'public_key', which was provided via a command-line option (`--root_key` or `--code_signing_key`). The HSM manager must know how to properly identify the key on the HSM given this string.

The public key instance is required to supply the public methods described in the sections that follow. The PKCS #11 implementation of this function, `get_public_key`, is below:

```
def get_public_key(self, public_key):
    try:
        key_, local_key = self.get_key(public_key, ObjectClass.PUBLIC_KEY)
        key_ = key_[Attribute.EC_POINT]
```

```
    except pkcs11.NoSuchKey:
        pass # No key found
    except pkcs11.MultipleObjectsReturned:
        pass # Multiple keys found
    return _PUBLIC_KEY(key_[3:], local_key)
```

3.10.1.1. PUBLIC_KEY.get_X_Y()

This function returns a `common_util.BYTE_ARRAY()` that contains the elliptic curve point associated with the key. The returned value should be X concatenated with Y, each with the proper number of bytes. For our implementation, each of X and Y are 32 bytes (256 bits) because secp256r1 curve parameters are required.

3.10.1.2. PUBLIC_KEY.get_permission()

Intel FPGA PAC keys have associated permissions. This function returns an integer that corresponds to the assigned key permissions. For Intel FPGA PACs, all root key permissions must be the constant `0xFFFFFFFF`. For code signing keys, the permissions are described below.

Table 5. Key Permissions

Value	Name	Permission
1	SIGN_SR	Sign the FIM or Static Region
2	SIGN_BMC	Sign the card BMC Nios firmware and/or the Intel MAX 10 image
4	SIGN_PR	Sign the PR Region or AFU

3.10.1.3. PUBLIC_KEY.get_ID()

Intel FPGA PACs have a laddering key mechanism that allows for cancellation of code signing keys. This method returns the integer key ID of the specified key. The root key ID must be the constant `0xFFFFFFFF`. Root keys cannot be canceled.

Intel FPGA PAC N3000 FPGA SR user image code signing key IDs must be in the range 0 to 127 (7-bit unsigned).

3.10.1.4. PUBLIC_KEY.get_content_type()

Code signing keys and root keys can be restricted to signing only certain types of content. For instance, there are separate root keys for PR, SR, and BMC bitstreams as well as corresponding code signing keys. This method should return the bitstream type associated with this key, and must be one of `{FIM, SR, BBS, BMC, BMC_FW, AFU, PR, or GBS}`.

3.10.2. HSM_MANAGER.sign(data, key)

This method uses the key provided to generate an ECDSA signature over the provided data.

The return value of this method is a `common_util.BYTE_ARRAY()` containing the R and S values of the signature concatenated. PACSign only signs hashes, so the length of the data to be signed will be a fixed-length 32 byte array.

3.10.3. Signing Operation Flow

A PACSign command that invokes the PKCS #11 manager plugin initializes it with the configuration file name.

PACSign performs insertion of authentication blocks into the bitstream, signed by the root and code signing keys. The resultant signed bitstream is written to the specified output file.

PACSign requests that the HSM manager retrieve the public key X and Y values for the root key and the code signing key. The HSM manager returns the R and S signature over PACSign-provided 256-bit hash values using the root key and code signing key. The following code snippet demonstrates how PACSign utilizes the HSM manager.

```
self.pub_root_key_c = self.hsm_manager.get_public_key(args.root_key)
common_util.assert_in_error(self.pub_root_key_c, \
    "Cannot retrieve root public key")
        self.pub_root_key = self.pub_root_key_c.get_X_Y()
        self.pub_root_key_perm = self.pub_root_key_c.get_permission()
        self.pub_root_key_id = self.pub_root_key_c.get_ID()
        self.pub_root_key_type = self.pub_root_key_c.get_content_type()

self.pub_CSK_c = self.hsm_manager.get_public_key(args.code_signing_key)
common_util.assert_in_error(self.pub_CSK_c != None, \
    "Cannot retrieve public CSK")
        self.pub_CSK = self.pub_CSK_c.get_X_Y()
        self.pub_CSK_perm = self.pub_CSK_c.get_permission()
        self.pub_CSK_id = self.pub_CSK_c.get_ID()
        self.pub_CSK_type = self.pub_CSK_c.get_content_type()

sha = sha256(block0.data).digest()
rs = self.hsm_manager.sign(sha, args.code_signing_key)
sha = sha256(csk_body.data).digest()
rs = self.hsm_manager.sign(sha, args.root_key)
```

3.11. PACSign Man Page

PACSign man page is reproduced here for convenience.

```
SYNOPSIS
python PACSign.py [-h] {FIM,SR,BBS,BMC,BMC_FW,AFU,PR,GBS} ...
python PACSign.py <CMD> [-h] -t {UPDATE,CANCEL,RK_256,RK_384} -H HSM_MANAGER [-
C HSM_CONFIG] [-r ROOT_KEY] [-k CODE_SIGNING_KEY] [-d CSK_ID] [-i INPUT_FILE] [-
o OUTPUT_FILE] [-y] [-v]

DESCRIPTION
PACSign is a utility designed to insert proper authentication markers on
bitstreams targeted for the PACs. To accomplish this, it uses a root key and an
optional code signing key to digitally sign the bitstreams to validate their
origin. The PACs will not accept loading bitstreams without proper
authentication.
The current PACs only support elliptical curve keys with the curve type
secp256r1 or prime256v1. PACSign is distributed with managers for both OpenSSL
and PKCS #11.

BITSTREAM TYPES
The first required argument to PACSign is the bitstream type identifier.

{SR,FIM,BBS,BMC,BMC_FW,PR,AFU,GBS}

Allowable image types. FIM and BBS are aliases for SR, BMC_FW is an alias for
BMC, and AFU and GBS are aliases for PR.

SR (FIM, BBS)
```

```
Static FPGA image

BMC(BMC_FW)

BMC image, including firmware for some PACs

PR (AFU, GBS)

Reconfigurable FPGA image

REQUIRED OPTIONS
All bitstream types are required to include an action to be performed by
PACSign and the name and optional parameter file for a key signing module.

-t, --cert_type <type>

Values must be one of UPDATE, CANCEL, RK_256, or RK_384[^1].
`UPDATE` - add authentication data to the bitstream.
`CANCEL` - create a code signing key cancellation bitstream.
`RK_256` - create a bitstream to program a 256-bit root key to the device.
`RK_384` - create a bitstream to program a 384-bit root key to the device.
[^1]:Current PACs do not support 384-bit root keys.

-H, --HSM_manager <module>

The module name for a manager that is used to interface to an HSM. PACSign
supplies both openssl_manager and pkcs11_manager to handle keys and signing
operations.

-C, --HSM_config <cfg> (optional)

The argument to this option is passed verbatim to the specified HSM manager.
For pkcs11_manager, this option specifies a JSON file describing the PKCS #11
capable HSM's parameters.

OPTIONS
-r, --root_key <keyID>

The key identifier recognizable to the HSM manager that identifies the root key
to be used for the selected operation.

-k, --code_signing_key <keyID>

The key identifier recognizable to the HSM manager that identifies the code
signing key to be used for the selected operation.

-d, --csk_id <csk_num>

Only used for type CANCEL and is the key number of the code signing key to
cancel.

-i, --input_file <file>

Only used for UPDATE operations. Specifies the file name containing the data to
be signed.

-o, --output_file <file>

Specifies the name of the file to which the signed bitstream is to be written.

-y, --yes

Silently answer all queries from PACSign in the affirmative.

-v, --verbose

Can be specified multiple times. Increases the verbosity of PACSign. Once
enables non-fatal warnings to be displayed; twice enables progress information.
Three or more occurrences enables very verbose debugging information.
```

```
NOTES
Different certification types require different sets of options. The table
below describes which options are required based on certification type:

UPDATE
    root_key     code_signing_key     csk_id     input_file     output_file
SR   Optional[^2]  Optional[^2]           No          Yes           Yes
BMC  Optional[^2]  Optional[^2]           No          Yes            Yes
PR   Optional[^2]  Optional[^2]           No          Yes            Yes

CANCEL
    root_key     code_signing_key     csk_id     input_file     output_file
SR     Yes         No               Yes         No           Yes
BMC    Yes         No               Yes         No           Yes
PR     Yes         No               Yes         No           Yes

RK_256 / RK_384[^1]
    root_key     code_signing_key     csk_id     input_file     output_file
SR     Yes         No               No          No           Yes
BMC    Yes         No               No          No           Yes
PR     Yes         No               No          No           Yes
[^2]: For UPDATE type, both keys must be specified to produce an authenticated
bitstream. Omitting one key generates a valid, but unauthenticated bitstream
that can only be loaded on a PAC with no root key programmed for that type.

EXAMPLES
The following command will generate a root hash programming PR bitstream. The
generated file can be given to fpgasupdate to program the root hash for PR
operations into the device flash. Note that root hash programming can only be
done once on a PAC.

python PACSign.py PR -t RK_256 -o pr_rhp.bin -H openssl_manager -r
key_pr_root_public_256.pem

The following command will add authentication blocks to hello_afu.gbs signed by
both provided keys and write the result to s_hello_afu.gbs. If the input
bitstream were already signed, the old signature block is replaced with the
newly-generated block.

python PACSign.py PR -t update -H openssl_manager -i hello_afu.gbs -o
s_hello_afu.gbs -r key_pr_root_public_256.pem -k key_pr_csk0_public_256.pem

The following command will generate a code signing key cancellation bitstream
to cancel code signing key 4 for all BMC operations. CSK 4 bitstreams that
attempt to load BMC images will be rejected by the PAC.

python PACSign.py BMC -t cancel -H openssl_manager -o csk4_cancel.gbs -r
key_bmc_root_public_256.pem -d 4
```

3.12. Accessing Intel FPGA PAC N3000 Version and Authentication Information

Throughout product development and deployment, you may want to:

- Verify the version of Intel FPGA PAC with which you are developing or deploying
- Identify or verify the root entry hash of your FPGA SR user image
- Collect data about the number of times the Staging flash has been programmed to assess any potential threats like flash wear-out
- Determine all cancellation CSK IDs you used for your FPGA SR user image

OPAE software provides three ways to obtain version or authentication information:

- `fpgainfo security` command
- `sysfs` files
- `bitstreaminfo` tool

For all three methods explained in the following sections, use the BMC root entry hash to identify the version of the Intel FPGA PAC N3000. Each Intel FPGA PAC N3000 has a unique BMC root entry hash.

Compare your BMC root entry hash output to the following table to identify your Intel FPGA PAC N3000 version.

Table 6. BMC Root Entry Hash Identifier for Intel FPGA PAC N3000

Platform	MMID (found on side cover of the Intel FPGA PAC)	BMC Root Entry Hash
Intel FPGA PAC N3000-1	999H1K (8 x 10G)	`0x757f524c2f45db58ac2a6c93e72b9167149979b795195d09d5e2efad82f2b031`
Intel FPGA PAC N3000-2	999HGN (2 x 2 x 25G)	
Intel FPGA PAC N3000-N	999PJD (2x2x25G, NEBS-friendly)	`0xec0f42d3af138e3eca7141107f7fed5f7c13846fadbba884e51ad26bf36a3d21`

3.12.1. Using `fpgainfo security` Command

The `fpgainfo security` command provides the following key identifying information for your Intel FPGA PAC and bitstreams:

Output	Description
FIM/SR root entry hash	Root entry hash programmed by you. If you have not programmed the FPGA SR user image root entry hash, this output reports as "hash not programmed."
BMC root entry hash	Root entry hash programmed by Intel.
PR root entry hash	Not applicable for Intel FPGA PAC N3000 and reports "hash not programmed" in output.
BMC flash update counter	Indicates how many times the BMC flash has been updated. This data can be useful in detecting threats. *Note:* When the BMC flash counter reaches 1000, the Intel MAX 10 BMC does not allow writes for 30 seconds after device startup and between updates. When the BMC flash counter reaches 2000, the Intel MAX 10 BMC does not allow writes for 60 seconds after device startup and between updates.
FIM/SR CSK IDs cancelled	Indicates the IDs of the FIM code signing keys that are cancelled.
BMC CSK IDs cancelled	Indicates the IDs of the BMC code signing keys that are cancelled.
AFU CSK IDs cancelled	Not applicable for Intel FPGA PAC N3000 and reports "None"

Because partial reconfiguration is not supported for the Intel FPGA PAC N3000, you can ignore the output for "PR root entry hash" and "AFU CSK IDs cancelled".

Using this command requires sudo or root privileges on your host.

```
$ sudo fpgainfo security

Board Management Controller, MAX10 NIOS FW version D.2.1.24
Board Management Controller, MAX10 Build version D.2.0.7
//****** SECURITY ******//
Object Id                        : 0xEC00001
```

```
PCIe s:b:d.f                    : 0000:8a:00.0
Device Id                       : 0x0b30
Numa Node                       : 1
Ports Num                       : 01
Bitstream Id                    : 0x2300011001030F
Bitstream Version               : 0.2.3
Pr Interface Id                 : f3c99413-5081-4aad-bced-07eb84a6d0bb
FIM/SR root entry hash          : hash not programmed
BMC root entry hash             :
0xec0f42d3af138e3eca7141107f7fed5f7c13846fadbba884e51ad26bf36a3d21
PR root entry hash              : hash not programmed
SMB parameters update counter(2) : 0
User flash update counter       : 1
FIM/SR CSK IDs canceled         : None
BMC CSK IDs canceled            : None
AFU CSK IDs canceled            : None
```

3.12.2. Reading sysfs Files for Identifying Information

The information provided by the `fpgainfo security` command is also available in sysfs entries. The sysfs entries are found in two locations:

1. `/sys/class/ifpga_sec_mgr/ifpga_sec<X>/security`
2. `/sys/class/fpga/intel-fpga-dev.<X>/intel-fpga-fme.<X>/spi-altera.<X>.auto/spi_master/spiX/spi<X>.<X>/ifpga_sec_mgr/ifpga_sec<X>/security`

Note: The <X> found in the following paths is a numeric value that is assigned by the kernel and is indeterminate.

The first pathname above uses a symlink to reference the same location as the second pathname. To correlate the two pathnames above, type:

```
ls -l /sys/security/ifpga_sec_mgr/ifpga_sec<X>
```

A listing of this directory displays the files in the table below:

Table 7. Sysfs File List

Sysfs File	Output	Description	File Data Format
sr_root_hash	SR root entry hash	Root entry hash programmed by you. If you have not programmed the FPGA SR user image root entry hash, this output reports as "hash not programmed."	Long hexadecimal output prefixed with "0x" or "hash not programmed" if the bitstreams is unsigned.
bmc_root_hash	BMC root entry hash	Root entry hash programmed by Intel.	Long hexadecimal output prefixed with "0x".
pr_root_hash	PR root entry hash	Not applicable for Intel FPGA PAC N3000 and reports "hash not programmed" in output.	N/A
user_flash_update_counter	User Flash update counter	Indicates how many times the staging area flash is updated. has been updated. This data can be useful in detecting threats.	Single, numeric value

continued...

(2) The SMB parameters update counter is not used and does not increment.

Sysfs File	Output	Description	File Data Format
		Note: When the staging area flash counter reaches 1000, the Intel MAX 10 BMC does not allow writes for 30 seconds after device startup and between updates. When the BMC flash counter reaches 2000, the Intel MAX 10 BMC does not allow writes for 60 seconds after device startup and between updates.	
`sr_canceled_csks`	SR CSK IDs canceled	Indicates the IDs of the FIM code signing keys that are cancelled.	Comma-separated list of decimal numbers and ranges, such as: 0, 3-6, 8-10
`bmc_canceled_csks`	BMC CSK IDs canceled	Indicates the IDs of the BMC code signing keys that are cancelled.	Comma-separated list of decimal numbers and ranges, such as: 0, 3-6, 8-10
`pr_canceled_csks`	AFU CSK IDs canceled	Not applicable for Intel FPGA PAC N3000.	Comma-separated list of decimal numbers and ranges, such as: 0, 3-6, 8-10

3.12.3. Using `bitstreaminfo` Tool

The `bitstreaminfo` tool also displays authentication information for `*.bin` files. Information includes any JSON header strings and authentication header block information. For FPGA SR user image bitstreams, the `bitstreaminfo` command also displays a small portion of the payload for FPGA SR user image bitstreams. The `bitstreaminfo` tool requires sudo or root privileges on your host:

```
$ sudo bitstreaminfo <file>
```

An example:

```
$ sudo bistreaminfo firmware.bin
```

This command displays the Block 0 and Block 1 content prepended by the PACSign tool to the FPGA SR user image. Depending on if your bitstream is signed or unsigned Block 1 output varies:

- Unsigned bitstream: Block 1 output reports 0x0 for Root public key X,Y and Code signing key X,Y.
- Signed bitstream: Block 1 output reports a value for Root public key X,Y and Code Signing key X,Y.

The magic number output in Block 0 and 1 are static values populated by PACSign.

Table 8. Block 0 Fields

Parameter	Description
Content length	Indicates the length of the FPGA SR user image. PACSign performs an internal check to see if the length is within the maximum length for Intel FPGA PAC N3000.
Content type	SR or BMC

continued...

Parameter	Description
Cert type	For an FPGA SR user image, Cert type can be: • Update : Unsigned/signed FPGA SR user image • Root Key Hash Programming : Root entry hash bitstream • Cancellation Certificate : Cancelled Code Signing key ID bitstream for FPGA SR user image. After you program a cancellation certificate, the Intel FPGA PAC prohibits you from loading any FPGA SR user image that was signed with the cancelled CSK ID. For an Intel-provided bitstream, Cert type can be: • Update : Signed BMC firmware or unsigned FPGA SR user image • Cancellation Certificate : Cancelled Code Signing key ID bitstream for BMC. After you program a cancellation certificate, the Intel FPGA PAC prohibits you from loading any BMC bitstream that was signed with the cancelled CSK ID.
Protected content SHA-256	SHA-256 is computed over the entire protected bitstream and it is compared against the SHA-256 calculated by PACSign and programmed into Block 0. You can check if `bitstreaminfo` reports a Match as shown below.
Protected content SHA-384	SHA-384 is computed over the entire protected bitstream and it is compared against the SHA-256 calculated by PACSign and programmed into Block 0. You can check if `bitstreaminfo` reports a Match as shown below. *Note:* Current Intel FPGA PAC N3000 versions do not support 384 bit root key but the tool computes the SHA-384 on the protected content.

Table 9. Block 1 Fields

Parameter	Description
Root Entry Permissions	Constant value: `0xffffffff`
Root Entry Key ID	Constant value: `0xffffffff`
Root public key x,y	Value populated if bitstream was signed using root key and CSK.
Expected root entry hash	Hash of all the root fields in Block 1 are computed. You can visually compare this against the FPGA SR user image root entry hash that is programmed into the card. `fpgainfo security` displays the FPGA SR user image root entry hash. If `fpgainfo security` reports "FIM/SR root entry hash not programmed", then the `bitstreaminfo` tool skips the compatibility check.
CSK key ID	The CSK ID can range from 0 - 127. `fpgainfo security` displays a list of CSK IDs canceled. If bitstream uses a CSK ID that matches the cancelled CSK ID, `fpgasupdate` prohibits programming the bitstream.
Code signing key x,y	Value reported if Bitstream was signed using root key and CSK.
Signature R, S	Signature over hash of CSK Public Key using private root key. Your HSM populates this signature.
Expected CSK hash	This field varies when the CSK ID changes. It is a hash of the CSK fields.
Signature R, S	Signature over hash of Block 0 using CSK private key.

The signature along with CSK fields help verify the bitstream.

The sample below show `bitstreaminfo` output using the signed 2 x 2 x 25G factory bitstream:

```
$ sudo bitstreaminfo $N3000_PLATFORM_ROOT/bin/sr_vista_rot_2x2x25G-v1.3.16.bin
File $N3000_PLATFORM_ROOT/bin/sr_vista_rot_2x2x25G-v1.3.16.binBlock 0:
    Block 0 magic =        0xb6eafd19
    Content length =    0x02a86000
    Content type =        SR
    Cert type =        UPDATE
    Protected content SHA-256:
            0xc10a77f9162945ab45dd943ca136e13f1b6d5278be722ad7519fbafacdedc73f
    Calculated protected content SHA-256:
            0xc10a77f9162945ab45dd943ca136e13f1b6d5278be722ad7519fbafacdedc73f
```

```
        Match
    Protected content SHA-384:

0x226a5f616c7b69f806da8b03316307c19e364449b46787d24e57bedadd9c9c3aa0510fa958b0d0
4fa5fec8b5465eb90c
    Calculated protected content SHA-384:

0x226a5f616c7b69f806da8b03316307c19e364449b46787d24e57bedadd9c9c3aa0510fa958b0d0
4fa5fec8b5465eb90c
        Match
Block 1:
    Block 1 magic =    0xf27f28d7
        Root Entry magic =        0xa757a046
        Root Entry curve magic =    0xc7b88c74
        Root Entry permissions =    0xffffffff
        Root Entry key ID =        0xffffffff
        Root public key X =
0x0000000000000000000000000000000000000000000000000000000000000000
        Root public key Y =
0x0000000000000000000000000000000000000000000000000000000000000000

        Expected root entry hash =
0xf8ff7e0a52a378483c85301df49c7d55ffd26f794121bdb8b102d7e1c3132bb9

        CSK magic =            0x14711c2f
        CSK curve magic =        0xc7b88c74
        CSK permissions =        0xffffffff
        CSK key ID =            0x00000000
        Code signing key X =
0x0000000000000000000000000000000000000000000000000000000000000000
        Code signing key Y =
0x0000000000000000000000000000000000000000000000000000000000000000
        CSK signature magic =        0xde64437d
        Signature R =
0x0000000000000000000000000000000000000000000000000000000000000000
        Signature S =
0x0000000000000000000000000000000000000000000000000000000000000000

        Expected CSK hash =
0xbe8a02e7932d98aff66584598978d84412e3c641927efac2cb786a1754cfcd4e

        Block 0 Entry magic =        0x15364367
        Block 0 Entry signature magic = 0xde64437d
        Signature R =
0x0000000000000000000000000000000000000000000000000000000000000000
        Signature S =
0x0000000000000000000000000000000000000000000000000000000000000000
Payload:
    80 20 01 00 3a 65 80 00 20 00 00 00 ff ff ff ff
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

    ...
    ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

For more examples of `bitstreaminfo` command, see Appendix A.

4. Using fpgasupdate

Use the `fpgasupdate` command to securely update the following files in flash:

- BMC Nios firmware and Intel MAX 10 FPGA images
- FPGA SR user images

When you call `fpgasupdate` the BMC orchestrates the update.

- The BMC restricts all access to the flash until the `fpgasupdate` tool sends a request to the BMC to begin the update process.
- The BMC rejects an update request if another update is currently in progress. The BMC monitors flash write and update counts and delays an update 30 seconds if more than 1,000 updates have occurred, and 60 seconds if more than 2,000 updates have occurred.
- The BMC grants access only to a staging area in the flash, and only for enough time for the host to write an update into the staging area.
- The BMC then restricts all flash write access to ensure the update image cannot be changed during or after the authentication process.
- During the `fpgasupdate` process, the Nios in the BMC stops polling the sensors and updating the platform level data model (PLDM) registers but responds to PLDM requests. Thus, any PLDM reads or `fpgad` polling during `fpgasupdate` returns stale data from before the update began.
- If authentication is successful, the BMC copies the image from the staging area into the appropriate section in flash.

To use the command type:

```
$ sudo fpgasupdate [--log-level=<level>] file [bdf]
```

where the following options are as follows:

Table 10. fpgasupdate Options

Parameters	Options	Notes
level	state, ioctl, debug, info, warning, error, critical. Default value is state.	N/A
file	The secure update file that you program in the Intel FPGA PAC	N/A
[bdf] *Note:* You must provide the BDF assigned to the `PCIe DevID 0b30` on your system.	`[ssss:]bb:dd:f`, corresponding to PCIe segment, bus, device, function. The segment is optional; if omitted, a segment of `0000` is assumed.	If there is only one Intel FPGA PAC in the system, then `bdf` may be omitted. In this case, `fpgasupdate` determines the address automatically.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

To program your FPGA SR user image after an `fpgasupdate`, type the following command:

```
$ sudo rsu bmcimg 3e:00.0
```

Related Information

Intel Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000

4.1. Troubleshooting

`fpgasupdate` provides descriptive errors when it cannot complete the requested operation.

When using `fpgasupdate` to program bitstreams created or signed with PACSign, the tool may reject the bitstream if, for example, there was an error in the signing process or if the signed bitstream is corrupted. The OPAE driver reports the BMC doorbell and authentication status register values into the system messages log. You may find this log file in a location such as `/var/log/messages` or `/etc/syslog` depending on the OS you are using. The error entry contains the keywords `intel-max10`. An example of output in the log file might look something like this:

[4971.546624] intel-max10 spi2.0: RSU error status: 8'h10022104

[4971.548681] intel-max10 spi2.0: RSU auth result: 8'h00000011

In this example the error status value, bit[23:16] is the RSU error value to reference in the *BMC Doorbell Register Values and Error Descriptions* table.

You may use the following tables to decode the authentication status and associated errors.

Table 11. BMC Doorbell Register Values and Error Descriptions

RSU-error [23:16] Value	Status Name	Status Description	Corrective Action
8'h00	Normal status	-	Not applicable.
8'h01	Host timeout	Flow Error: Host timeout sending bitstream. Possible OS or system issue.	Attempt sending bitstream again.
8'h02	Authentication failure	-	Ensure bitstream is properly signed with the correct keys.
8'h03	Image copy failure	Flow Error: Image copy failure	Attempt copy again. If issue persists, contact Intel support.
8'h04	Fatal, error, Nios boot-up failure	-	Contact Intel support.
8'h05	Reject C827 Retimer EEPROM update	-	Ensure installed retimer version is actually older than the attempted updated version.
8'h06	Staging area non-incremental write failure	-	Contact Intel support.
8'h07	Staging area erase failure	-	Contact Intel support.

continued...

RSU-error [23:16] Value	Status Name	Status Description	Corrective Action
8'h08	Staging area write wearout	-	Contact Intel support.
8'h80	Nios boot OK	-	Not applicable.
8'h81	Update OK	Update image okay	Not applicable.
8'h82	Factory OK	Factory image okay	Not applicable.
8'h83	Update Failure	-	Contact Intel support.
8'h84	Factory Failure	-	Contact Intel support.
8'h85	Nios Flash Open Error	-	Contact Intel support.
8'h86	FPGA Flash Open Error	-	Contact Intel support.
Others	Reserved	-	-

The errors in the following Authentication Status Register table are for failures that occur when programming the root key hash bitstream or the cancellation key bitstream. These error types might occur if for example a root entry hash bitstream is signed with the incorrect key. These registers do not capture errors for signed FPGA SR user image bitstream programming.

Table 12. Authentication Status Register Values and Error Descriptions

Authentication Status Value	Error Name	Error Description	Corrective Action
32'h00000000	Authenticate Pass	Authenticate Pass	Not applicable.
32'h00000001	Block0 Magic value error	Bitstream Format Error: Block 0 bad magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h00000002	Block0 ConLen error	Bitstream Format Error: Block 0 content length error. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h00000003	Block0 ConType B[7:0] > 2	Bitstream Format Error: Block 0 content type error. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h00000004	Block1 Magic value error	Bitstream Format Error: Block 1 bad magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h00000005	Root Entry Magic value error	Bitstream Format Error: Root entry bad magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h00000006	Root Entry Curve Magic value error	Bitstream Format Error: Root entry bad curve magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h00000007	Root Entry Permission error	Root entry bad permissions. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.

continued...

Authentication Status Value	Error Name	Error Description	Corrective Action
32'h00000008	Root Entry Key ID error	Bitstream Format Error: Root entry bad key ID. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h00000009	CSK Entry Magic value error	Bitstream Format Error: CSK bad magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h0000000A	CSK Entry Curve Magic value error	Bitstream Format Error: CSK bad curve magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h0000000B	CSK Entry Permission error	Authentication Error: CSK bad permission. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h0000000C	CSK Entry Key ID error	Bitstream Format Error: CSK invalid key ID, Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h0000000D	CSK Entry Signature Magic value error	Bitstream Format Error: CSK bad signature magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h0000000E	Block0 Entry Magic value error	Bitstream Format Error: Block 0 entry bad magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h0000000F	Block0 Entry Signature Magic value error	Bitstream Format Error: Block 0 entry bad signature magic number. Indicates bitstream corruption.	Ensure bitstream is properly signed with the correct keys.
32'h00000010	Root Entry Hash bitstream not programmed for RSU and Cancellation	Authentication error: Cancellation attempted with no root entry hash bitstream programmed.	Program root entry hash bitstream.
32'h00000011	Root Entry verify SHA failed	Authentication Error: Root hash mismatch.	Ensure bitstream is properly signed with the correct keys.
32'h00000012	CSK Entry verify ECDSA and SHA failed	Authentication Error: CSK signature invalid. Indicates CSK or root entry hash tampering.	Ensure bitstream is properly signed with the correct keys.
32'h00000013	Block0 Entry verify ECDSA and SHA failed	Authentication Error: Block 0 entry signature invalid. May indicate image tampering.	Ensure bitstream is properly signed with the correct keys.
32'h00000014	KEY ID of authenticate blob is invalid	Bitstream Format Error: CSK invalid key ID. Indicates you are using an ID value greater than what is allowed.	Ensure bitstream is properly signed with the correct keys.
32'h00000015	KEY ID is cancelled	Authentication Error: CSK canceled. Indicates you are attempting to program an image with a cancelled CSK.	Ensure bitstream is properly signed with the correct keys.

continued...

Authentication Status Value	Error Name	Error Description	Corrective Action
32'h00000016	Update content SHA verify failed	Authentication Error: Payload SHA mismatch. May indicate tampering of the bitstream.	Verify correctness of bitstream; may need to resign.
8'h00000017	Cancellation content SHA verify failed	Authentication Error: Payload SHA mismatch. May indicate tampering of the cancellation certificate.	Verify correctness of bitstream; may need to resign.
8'h00000018	HASH Programming content SHA verify failed	Authentication Error: Payload SHA mismatch. May indicate tampering of the root key.	Verify correctness of bitstream; may need to resign.
8'h00000019	Invalid cancellation ID of cancellation certificate	Bitstream Format Error: CSK invalid key ID	Verify correctness of bitstream; may need to resign.
8'h0000001A	KEY hash has been programmed for KEY hash programming certificate	Authentication Error: Attempt to program root entry hash when the root entry hash bitstream has already been programmed.	You may only program root entry hash bitstream one time.
8'h0000001B	Invalid operation of Block0 ConType	-	Contact Intel support.
8'h000000FF	Generic Authentication Failure	-	Contact Intel support.

5. Document Revision History for Security User Guide

Document Version	Changes
2020.06.15	• Added the following sections: — Accessing Intel FPGA PAC N3000 Version and Authentication Information • Using `fpgainfo security` Command • Reading `sys` files for Identifying Information • Using `bitstreaminfo` Tool — Appendix A: `bitstream` Tool Examples • Added a new `fpgainfo security` command
2019.12.12	Modified `rsu` command in *Authentication* and Using *fpgasupdate* sections.
2019.11.25	Initial release.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

A. bitstreaminfo Tool Examples

Output example for CSK1 cancellation certificate:

```
$ bitstreaminfo ssl_csk1_cancel.bin

Output:
File ssl_csk1_cancel.bin:
Block 0:
   Block 0 magic =         0xb6eafd19
   Content length =        0x00000080
   Content type =          SR
   Cert type =             CANCEL
Protected content SHA-256:
0xed4fc1d85afa5175e4973c9780b78fa000f070c00230ec18d6190133cb915db5
Calculated protected content SHA-256:
0xed4fc1d85afa5175e4973c9780b78fa000f070c00230ec18d6190133cb915db5

    Match

Protected content SHA-384:
0x23c1a67cdd52bf7c6a4f34ebc96b64e5d51d3010ab7754572007e81701b6eb4bcedad337ccde56
3817a19a1e17601a31

Calculated protected content SHA-384:

0x23c1a67cdd52bf7c6a4f34ebc96b64e5d51d3010ab7754572007e81701b6eb4bcedad337ccde56
3817a19a1e17601a31
                Match
Block 1:
    Block 1 magic = 0xf27f28d7
    Root Entry magic =              0xa757a046
    Root Entry curve magic =        0xc7b88c74
    Root Entry permissions =        0xffffffff
    Root Entry key ID =             0xffffffff
    Root public key X =
0xd562f7c475598a44f4cfb3b96e29822a11b823873da1600660a1f2ef7460c109
    Root public key Y =
0x9dab9ea9cb25505c9b40ef509245bb23fd9dcdfa3c9f2d7250e9e8063527ef11

    Expected root entry hash =
0xe9e618adf1818bf0327cd993a4f706451e877d046283a7bbf5b4df1a3fcc5dad
        No CSK

   Block 0 Entry magic =           0x15364367
   Block 0 Entry signature magic = 0xde64437d
   Signature R =
0x1a0d878aebe9bf0a719ca7c1f33fec44e1357f85b54063d79999bff2aa07cdd6
   Signature S =
0x46bd1dac9937a847bb3620559901ed3e57a137384eef2b1994d4b3d4cc2f5ad8
Payload:
        01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Output example for unsigned Intel Arria 10 GT Bitstream:

```
$ bitstreaminfo sr_vista_rot_4x25G-v1.3.15.bin
File sr_vista_rot_4x25G-v1.3.15.bin:
Block 0:
   Block 0 magic =         0xb6eafd19
   Content length =        0x02b00000
   Content type =          SR
   Cert type =             UPDATE
   Protected content SHA-256:
       0xe4ecd5f6b332bba7b03bcdbe5f9c28317dda59e403148cedec4550f5fa5644b4
   Calculated protected content SHA-256:
        0xe4ecd5f6b332bba7b03bcdbe5f9c28317dda59e403148cedec4550f5fa5644b4
                Match
   Protected content SHA-384:

0x4c56e31d8a4d37d3cdab616a8d6a73a6cce12bd9f0737a4676b3a736bfe4425aaabc046a1c3cc3
713cae90dd9d1136ef
   Calculated protected content SHA-384:

0x4c56e31d8a4d37d3cdab616a8d6a73a6cce12bd9f0737a4676b3a736bfe4425aaabc046a1c3cc3
713cae90dd9d1136ef
              Match
Block 1:
   Block 1 magic = 0xf27f28d7
   Root Entry magic =              0xa757a046
   Root Entry curve magic =        0xc7b88c74
   Root Entry permissions =        0xffffffff
   Root Entry key ID =             0xffffffff
   Root public key X =
0x0000000000000000000000000000000000000000000000000000000000000000
   Root public key Y =
0x0000000000000000000000000000000000000000000000000000000000000000

  Expected root entry hash =
0xf8ff7e0a52a378483c85301df49c7d55ffd26f794121bdb8b102d7e1c3132bb9

   CSK magic =                     0x14711c2f
   CSK curve magic =               0xc7b88c74
   CSK permissions =               0xffffffff
   CSK key ID =                    0x00000000
   Code signing key X =
0x0000000000000000000000000000000000000000000000000000000000000000
   Code signing key Y =
0x0000000000000000000000000000000000000000000000000000000000000000
    CSK signature magic =           0xde64437d
     Signature R =
0x0000000000000000000000000000000000000000000000000000000000000000
     Signature S =
0x0000000000000000000000000000000000000000000000000000000000000000

     Expected CSK hash =
0xbe8a02e7932d98aff66584598978d84412e3c641927efac2cb786a1754cfcd4e

     Block 0 Entry magic =           0x15364367
     Block 0 Entry signature magic = 0xde64437d
     Signature R =
0x0000000000000000000000000000000000000000000000000000000000000000
     Signature S =
0x0000000000000000000000000000000000000000000000000000000000000000
Payload:
        80 20 01 00 3a 65 80 00 20 00 00 00 ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

```
        ...
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

Output example for signed Intel Arria 10 GT Bitstream:

```
$ bitstreaminfo signed_sr_vista_rot_4x25G-v1.3.15.bin
File unsigned_sr_vista_rot_4x25G-v1.3.15.bin:
Block 0:
   Block 0 magic =         0xb6eafd19
   Content length =        0x02b00000
   Content type =          SR
   Cert type =             UPDATE
   Protected content SHA-256:

0xe4ecd5f6b332bba7b03bcdbe5f9c28317dda59e403148cedec4550f5fa5644b4
   Calculated protected content SHA-256:

0xe4ecd5f6b332bba7b03bcdbe5f9c28317dda59e403148cedec4550f5fa5644b4
                Match
   Protected content SHA-384:

0x4c56e31d8a4d37d3cdab616a8d6a73a6cce12bd9f0737a4676b3a736bfe4425aaabc046a1c3cc3
713cae90dd9d1136ef
   Calculated protected content SHA-384:

0x4c56e31d8a4d37d3cdab616a8d6a73a6cce12bd9f0737a4676b3a736bfe4425aaabc046a1c3cc3
713cae90dd9d1136ef
                Match
Block 1:
    Block 1 magic = 0xf27f28d7
    Root Entry magic =              0xa757a046
    Root Entry curve magic =        0xc7b88c74
    Root Entry permissions =        0xffffffff
    Root Entry key ID =             0xffffffff
    Root public key X =
0x09b39cb8cb5c51b649ad6555e0ca1b150932c4289024015f34cd4bb5d47b77f5
    Root public key Y =
0x9a9a9affef8f6b45b0b99a2efaa9c118469e3ea0396cb2fe50247d51fb7dba16

    Expected root entry hash =
0x5c47ce0b1edc53b2bc02bf9b8aecab95b139b1f07f15fd6f25df7eb25942c0e0

    CSK magic =                     0x14711c2f
    CSK curve magic =               0xc7b88c74
    CSK permissions =               0xffffffff
    CSK key ID =                    0x00000001
    Code signing key X =
0xfed4bf4826cf71c4246c9576892b474b1465bba137e141d1f6731fe03b7c312c
    Code signing key Y =
0x50e784b7209d5c6af35b55f7d140a3b19769d5bc19babd9c9170d05a3822a6d6
    CSK signature magic =           0xde64437d
    Signature R =
0x754ab8c579ac2fd0841fb50c978962f95bbc162ecc9544f1f18b99945cf655fd
    Signature S =
0x9f9af231cd7a39ba1c6d629023f2b4d316e010fd08eca130efbecbf0caf8e83e

    Expected CSK hash =
0xaaaac919f6aecb2532ce6322a76bb57b0f1f285dd4d71d178544ac59f2b78fda

   Block 0 Entry magic =           0x15364367
   Block 0 Entry signature magic = 0xde64437d
   Signature R =
0x680a36f4422137836963656O4e6789c4b2f6d20b9eb6c8b34abdef6e16bdb1f2
   Signature S =
0xfb2764d6db7eb658cd11f55084e981ba5db229c136e66afe8d1ab9e78f0f7510
Payload:
        80 20 01 00 3a 65 80 00 20 00 00 00 ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

```
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

        ...
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
```

Output example for Root Key hash bitstream:

```
$ bitstreaminfo root_public_program_ssl.bin
File root_public_program_ssl.bin:
Block 0:
        Block 0 magic =         0xb6eafd19
        Content length =        0x00000080
        Content type =          SR
        Cert type =             Root Entry Hash (256)
        Protected content SHA-256:

0xade5140d232e010fda6b79542d1d9f31a9de413b0a10d32bfd2208b01119d658
        Calculated protected content SHA-256:

0xade5140d232e010fda6b79542d1d9f31a9de413b0a10d32bfd2208b01119d658
                Match
        Protected content SHA-384:

0x033cd07c8917d11242d174f608cc7301051bb0145a13527340fcf0b370f98f88ef795029c6cead
dca27a4d221b1f7035
        Calculated protected content SHA-384:

0x033cd07c8917d11242d174f608cc7301051bb0145a13527340fcf0b370f98f88ef795029c6cead
dca27a4d221b1f7035
                Match
Block 1:
        Block 1 magic = 0xf27f28d7
        No root entry
        No CSK
        No block 0 entry
Payload:
        5c 47 ce 0b 1e dc 53 b2 bc 02 bf 9b 8a ec ab 95
        b1 39 b1 f0 7f 15 fd 6f 25 df 7e b2 59 42 c0 e0
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Output example for BMC RTL/Firmware bitstream from Intel:

```
$ bitstreaminfo VistaCreekBravoBMCFW_Release_WW13.2.bin
File VistaCreekBravoBMCFW_Release_WW13.2.bin:
Block 0:
        Block 0 magic =         0xb6eafd19
        Content length =        0x000d4e80
        Content type =          BMC
        Cert type =             UPDATE
        Protected content SHA-256:

0x7f49e08241f8390cc5b939843ecb14af73d464c9aa4998a9aff5cddac26b8bb6
        Calculated protected content SHA-256:

0x7f49e08241f8390cc5b939843ecb14af73d464c9aa4998a9aff5cddac26b8bb6
                Match
        Protected content SHA-384:

0x243d2e99486bb68ede871d6b052cabf0b441b1e0538fec8f8450fec58a4c9537b85f95d473972e
842924c7e334ebbebb
```

```
        Calculated protected content SHA-384:

0x243d2e99486bb68ede871d6b052cabf0b441b1e0538fec8f8450fec58a4c9537b85f95d473972e
842924c7e334ebbebb
                Match
Block 1:
        Block 1 magic = 0xf27f28d7
                Root Entry magic =              0xa757a046
                Root Entry curve magic =        0xc7b88c74
                Root Entry permissions =        0xffffffff
                Root Entry key ID =             0xffffffff
                Root public key X =
0x78a0db7ecef9f13c336e99334d34d10c33829cb290901b48af8c34fce107b3e7
                Root public key Y =
0x57cc5b60b89203bc9d975f59c813d1ffd8499d292b2c42262adb9483167832d4

                Expected root entry hash =
0x77698ea203e459f6cb0e65b54a1dd4ab47a6a6600e7988f723ad89f5b7f3673a

                CSK magic =                     0x14711c2f
                CSK curve magic =               0xc7b88c74
                CSK permissions =               0x00000002
                CSK key ID =                    0x00000000
                Code signing key X =
0xad481a506b8bf261fd0644eb7f0be98cde8152c015eb17a2d08ebd6b2af131df
                Code signing key Y =
0x2541eaff9213bb26247b593646aa45ce618a46cf5575de9f1ac21563c9f9570c
                CSK signature magic =           0xde64437d
                Signature R =
0xbfaf53b0fe2359ea3c86e2c35103f2a5df021f0231681216ab615a1c5f8255bf
                Signature S =
0xffccfd664e04f5dcef68c16b4d96708a91c59b1c2677ca3b07a7dc227ee5f31c

                Expected CSK hash =
0x6f0b20617a824725757482a23ff39a9b1096aa400436217103ed5a52fde5f52c

                Block 0 Entry magic =           0x15364367
                Block 0 Entry signature magic = 0xde64437d
                Signature R =
0x771201ca87d1622994ad21e8a75a0edb945b42bc885447903487ec79ed399750
                Signature S =
0x27823b355b81f25c996f2298c44fe7fd8cbb9e14f46fa8de6836b807c463632d
Payload:
        00 80 0a 00 80 80 0a 00 8c cd 02 00 50 ff 47 ff
        4d ff 49 c2 43 ff 42 fb ee 1c ae 00 ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
        ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

        ...
        00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```