Secure Production Programming Solution (SPPS) User Guide

Libero SoC v2025.1

Introduction (Ask a Question)

This user guide describes the Microchip Secure Production Programming Solution (SPPS), and includes the following information:

- High-level SPPS overview
- Use models
- Software/hardware components of SPPS
- Tools flow

SPPS is a solution provided by Microchip for secure programming of SmartFusion® 2, IGLOO® 2, PolarFire®, and PolarFire SoC devices in untrusted environments.

The Operation Engineer (OE) uses SPPS to prepare secure programming jobs based on design information received from design and firmware engineers. The created secure programing job is executed by manufacturing as part of the SPPS on the site of the Contract Manufacturer (CM).

SPPS provides the following during manufacturing in an untrusted environment or where insiders might be present:

- Ensures confidentiality of user design and security keys used through a secure initial key loading mechanism.
- Prevents unauthorized programming of the user design through an overbuild protection mechanism, preventing overbuilding of electronic systems.

Table of Contents

1. Secure Initial Key Loading (Ask a Question)

Secure Initial Key Loading allows the programming security settings, such as encryption keys, pass keys, and security locks, into a blank device under protection of the unique per-device factory key programmed into every SmartFusion 2, IGLOO 2, PolarFire, and PolarFire SoC device. This is achieved using the Authorization Code Protocol (see Authorization Code Protocol) built into Microchip SmartFusion 2, IGLOO 2, PolarFire, and PolarFire SoC devices. For execution during device programming, the Authorization Code Protocol requires the support of a Hardware Security Module (HSM). See HSM Hardware Modules used by SPPS for more information about HSMs.

Secure Initial Key Loading provides strong cryptographic protection of the user design and security settings programmed into a blank device. Programming of user defined security settings disables the factory default key modes, allowing users to access the programmed device. No one, including the Microchip personnel, can gain access to the device without the proper security credentials, if the factory test mode is protected by user security settings.
Note: The user must select the “Protect factory test mode access using FlashLock/UPK1" or "Permanently protect factory test mode access" in the Security Policy Manager (SPM).

2. Preventing Overbuilding (Ask a Question)

The overbuild protection mechanism allows the user to restrict the number of devices programmed using a specific design. The maximum number of allowed devices and the count of already programmed devices are stored in the physical HSM module, and cannot be physically cloned. The HSM module enforces overbuild protection during programming as a part of security protocols execution.

3. Use of Hardware Security Modules (Ask a Question)

An HSM is a physical computing device that safeguards and manages user key information. It allows execution of algorithms making use of those keys inside the security boundaries provided by the module. HSM modules protect the information they process. They consist of specially sealed packaging and various tamper-resistant and tamper-evident protection mechanisms.

The HSM module is attached to the PC (HSM Server) through hardware interfaces such as USB or PCIe. Storage space in the module is limited and most of the data used by the HSM is stored on the PC side in encrypted form. In-module memory contains an encryption key for protecting external data and data that must be physically uncloneable, such as overbuild protection counters.

SPPS uses two types of HSM: User HSM (U-HSM) and Manufacturer HSM (M-HSM).

For more information about HSM modules used in SPPS, see HSM Hardware Modules used by SPPS.

4. SPPS Functions and Services (Ask a Question)

The primary purpose of SPPS is to enable secure programming of SmartFusion 2, IGLOO 2, PolarFire, and PolarFire SoC devices in an untrusted environment. This flow is based on the use of an HSM and in this document it is referred to as HSM flow. However, the primary tool created for SPPS (Job Manager) can also be used for non-secured programming (non-HSM flow), which is the same as the programming supported by Libero®. In the case of non-HSM flow, initial key loading is done using the Key Loading Key (KLK) mode supported for SmartFusion 2, IGLOO 2, PolarFire, and PolarFire SoC devices. KLK mode is suitable for programming in a trusted environment or if security is not a concern.

The following SPPS supported common functions are available in HSM flow and non-HSM flow:

- Generation of programming bit streams and programming jobs outside the Libero tool
- Generation of Initiater programming bit streams (for initial key loading)
  - Generation of Update bit streams for updating Fabric and/or eNVM of an already programmed device
- Overwrite of eNVM Client(s) data
- Selection of eNVM Clients in generated bit streams
- Security settings overwrite
- Generation of SPI Flash file directory

The main SPPS flow (HSM flow) assumes the presence of HSM servers on the customer side and in most cases on the manufacturer side as well.

Features available in HSM flow only:

- HSM-protected user key generation
  - The user does not have access to the encryption and pass key values; nor do insiders
  - HSM generated keys are never present on the host workstation except in encrypted form
  - Secured Initial Key Loading through the Authorization Code Protocol (see Authorization Code Protocol)
    - Initial programming of the project key (same for all devices)
    - Initial programming of per-device keys
- Overbuild protection
- Generation and validation of Certificates of Conformance (CoC) of the programmed design
- Verifying Microchip device authenticity

5. SPPS Ecosystem (Ask a Question)

SPPS includes the following tools and servers (see Figure 6-1).

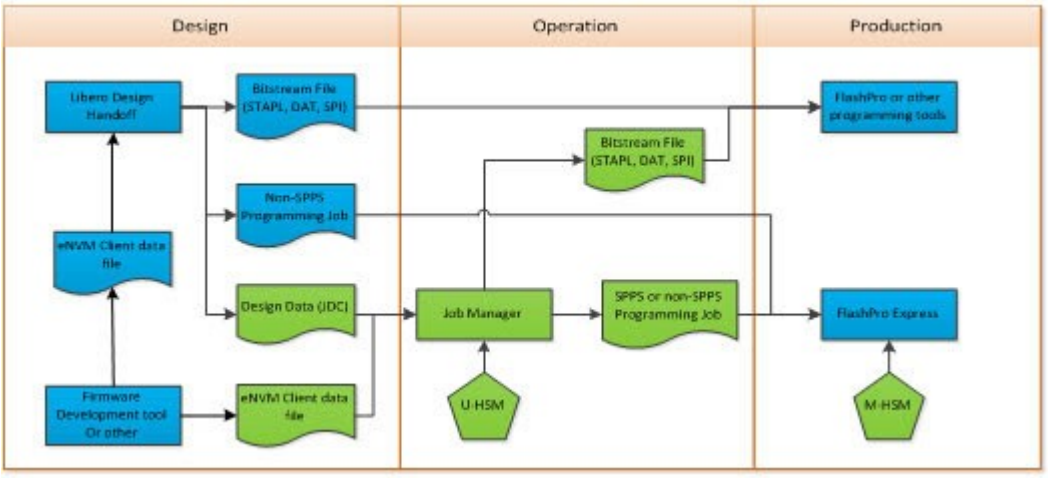
- **Libero Design Tool** —the design engineer creates and exports a Libero design in a Job Data Container (JDC) file.
- **Job Manager Tool** —used by the OE to generate an HSM or non-HSM programming job using a JDC file received from Libero. The Job Manager also allows the OE to modify certain design features, such as eNVM content and security settings.
- **FlashPro Express** —used during production to program the device from programming jobs created by the Job Manager tool.
- **U-HSM Server** —used by the Job Manager to create secured jobs for HSM flow. It also allows the OE to test-execute programming job created with FlashPro Express. The Job Manager uses the U-HSM to validate results of the programming job execution.
- **M-HSM Server** —used by FlashPro Express to generate protocol data and generate per-device key values (which are derived from the device serial number and the base values of the user key) during programming. The M-HSM enforces overbuild protection.
- **Firmware Design Tools** —firmware developers can generate new firmware images that can be sent to the OE and updated inside the Job Manager through the eNVM update feature.

6. Programming Production (Ask a Question)

The following figure shows the Microchip programming production flow. The flow supports simple programming jobs and bit stream file exports directly from the Libero tool, represented with the blue components in the following figure. Libero-based production flow only supports non-secured programming, based on KLK key mode. For more information, see the Libero SoC Design Flow User Guide.

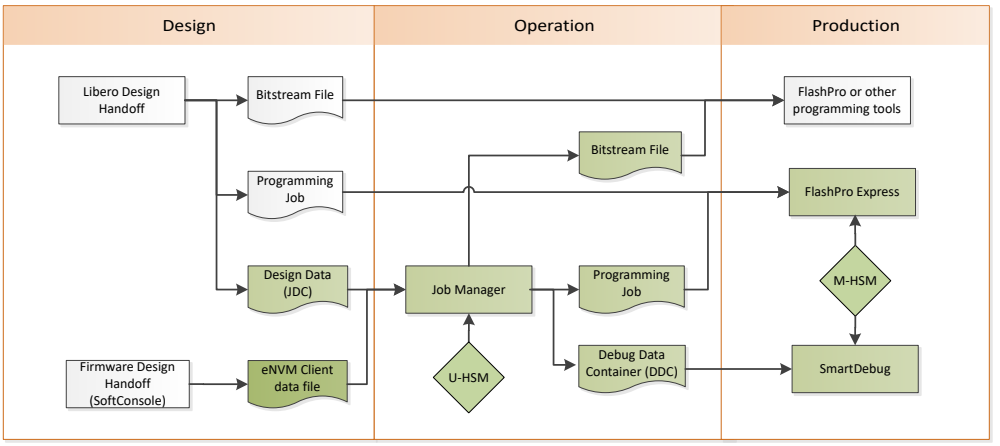
The Job Manager extends that capability: allowing bit stream file and programming job generation outside of the Libero tool, and the support of SPPS flow, represented with green components in the following figure.

Figure 6-1. Microchip Programming Flow



Bitstream files (STAPL, DAT, and SPI) are programmed using FlashPro, DirectC, or Silicon Sculptor. STAPL file can be used by third party JTAG production programming tools to program Microchip devices. Programming jobs are supported by FlashPro Express only. The following figure shows the debug flow. It allows DDC generation outside of the Libero tool supports SPPS flow, represented with green components.

Figure 6-2. Microchip Debug Flow



The main SPPS flow is based on the use of HSM in the Operation and Production site. It supports the advanced production the programming models and SmartDebug as described in the following sections.

6.1. Main SPPS Flow (Ask a Question)

The main objectives of this flow are to support secured programming of a blank device (initial programming), enforce overbuild protection, and allow an upgrade flow for already-programmed devices.

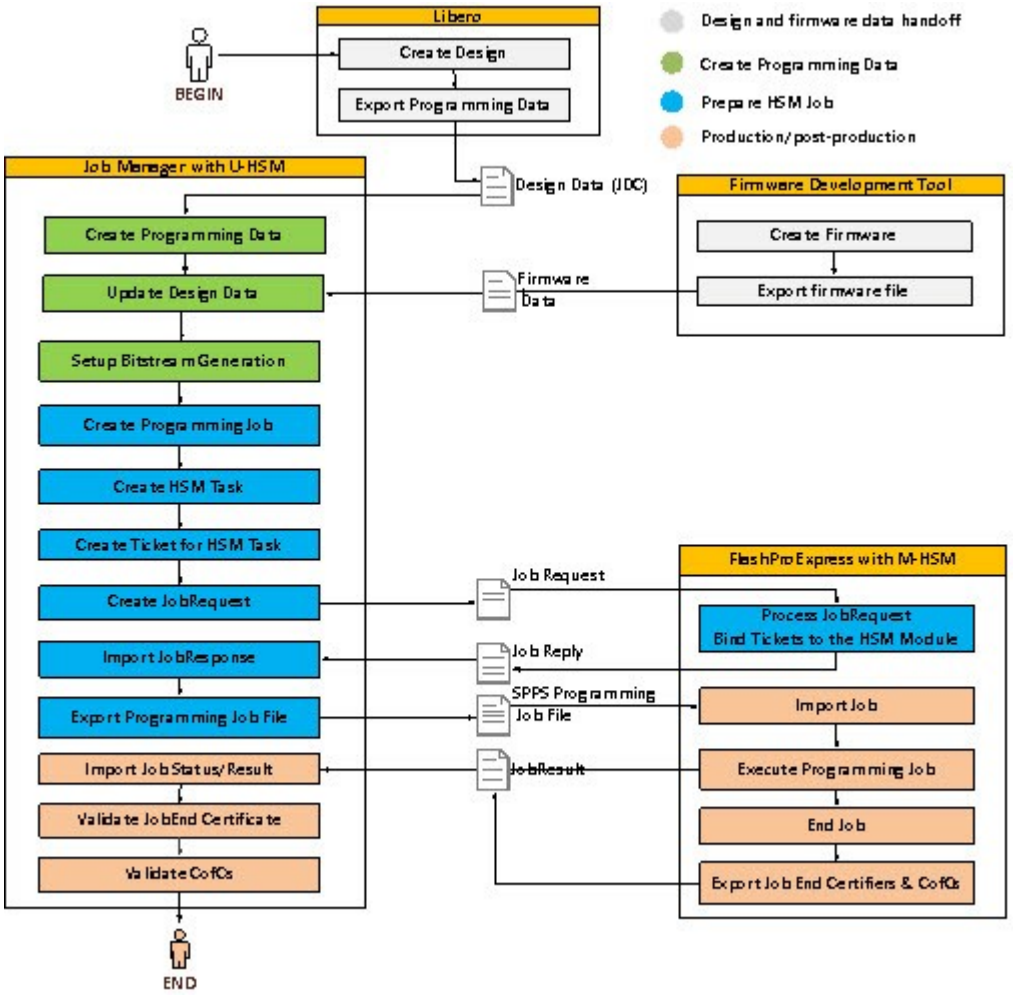
In this flow, all user-selected design information such as Fabric, eNVM, and Security is exported from Libero through a JDC file and handed off to the OE, which uses the Job Manager to create and submit an HSM Programming Job for production. The eNVM client data can be updated in the Job Manager during this step.

Tip: The eNVM data is typically provided by (but not limited to) the firmware development team.

All operations related to generation and use of encryption keys, pass keys, and other protected data is handled by the User HSM (U-HSM). Production executes the received HSM Programming Jobs using FlashPro Express. The Manufacturer HSM (M-HSM) serves FlashPro Express security protocol requests, and enforces the overbuild protection policy. Proof of programming results such as Certificates of Conformance generated by the programmed devices and the job end certifiers are sent back to the OE and can be validated using the U-HSM.

From a design security standpoint, design and operation areas are always considered to be trusted environments. OE can modify security settings from Libero through the JDC file. Production in the main (HSM-based) SPPS flow is considered an untrusted area with respect to design, security, and job execution. The following figure shows the main SPPS flow for programming.

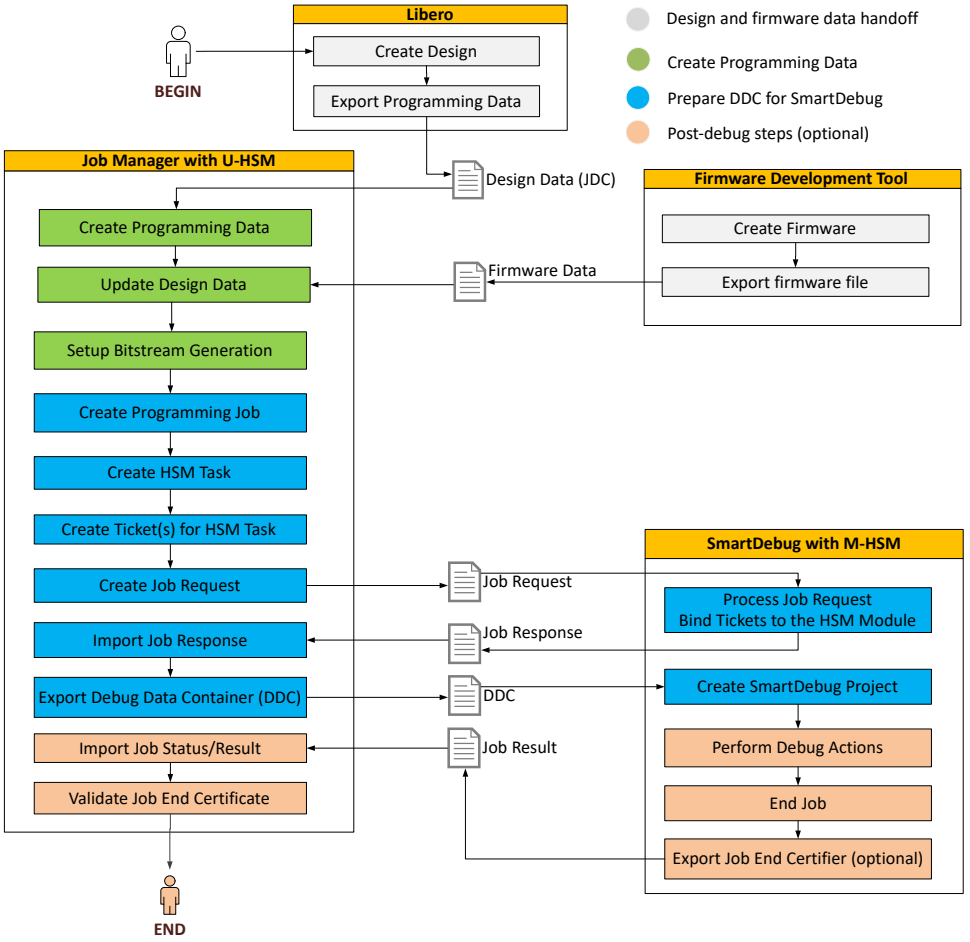
Figure 6-3. Main SPPS Flow



In the SmartDebug flow, all user-selected design information such as Fabric, eNVM, and Security is exported from Libero through a JDC file and handed off to the OE, which uses the Job Manager to create and submit an HSM Debug Job for production.

All operations related to generation and use of encryption keys, pass keys, and other protected data is handled by the User HSM (U-HSM). Production executes received HSM Debug Jobs using SmartDebug. The Manufacturer HSM (M-HSM) serves SmartDebug security protocol requests and enforces the overbuild protection policy. From a design security standpoint, design and operation areas are always considered to be trusted environments. OE can modify security settings from Libero through the JDC file. Production in the main (HSM-based) SPPS flow is considered an untrusted area with respect to design, security, and job execution. The following figure shows the main SPPS flow for SmartDebug.

Figure 6-4. Main SPPS Flow for SmartDebug



6.1.1. Libero Design Handoff (Ask a Question)

The design engineer exports design information to the Job Manager using the Export Job Manager Data tool in Libero. This export tool allows the user to select the components to be exported: Security, Fabric, or eNVM. The tool has GUI and TCL interfaces.

For more information, see Libero SoC Design Flow User Guide.

6.1.1.1. Security Export (Ask a Question)

Security information includes user-defined security keys, locks, and other settings selected by the user in the Libero SPM. If the user security settings are not defined, then the JDC file is exported with default device security settings.

Exported security settings can be changed by the OE in the Job Manager, depending on specific use cases explained in Main SPPS Flow and Non-HSM Flow.

6.1.1.2. Fabric Export (Ask a Question)

The Fabric component is exported and used in the Job Manager as it is.

6.1.1.3. eNVM Export (Ask a Question)

eNVM data exported through a JDC file includes all eNVM clients available in the Libero project. The OE can modify eNVM client data within the Job Manager project. This mechanism is intended to support cases such as firmware development/upgrade flow, which is described in the following section. The Job Manager can only modify client data; it cannot modify the eNVM client configuration. eNVM configuration is managed by the design engineer in the Libero project.

6.1.2. Firmware Handoff (Ask a Question)

While the initial version of the firmware for Microchip SmartFusion 2/IGLOO 2 or PolarFire/PolarFire SoC devices is typically included in newly created Libero designs, and can be exported from Libero through a JDC file, it is possible to issue re- or post-production firmware updates directly to Operation. Files with firmware update can be exported by firmware development tools such as IAR or SoftConsole in one of formats supported by Libero (see the Libero SoC Design Flow User Guide). Job Manager accepts and applies new firmware data through eNVM update, which is explained in eNVM or sNVM Update. Similar capability is also supported for Microchip PolarFire and PolarFire SoC devices with sNVM update.

6.1.3. Programming Data and Bitstream Initialization (Ask a Question)

Programming data is created by the OE within a Job Manager project from design data received from Libero through a JDC file. Its primary objective is to provide a programming job with programming bitstreams according to a selected programming scenario (initial programming, upgrade, and so on.)

6.1.3.1. Programming Data Creation through JDC Import (Ask a Question)

Programming data creation requires the user to point to the specific JDC file. Also, in the HSM flow, all key material is handled by the U-HSM on the operation side. Therefore, the Programming Data needs a special KeySet file pre-generated by the U-HSM. KeySet files are stored in a repository shared by different Job Manager projects, which enables key sharing. For more information about handling KeySet files, see Job Manager User Guide.

6.1.3.2. Design Data Modification (Ask a Question)

The Job Manager allows the OE to modify imported design data and supports the following use cases:

- eNVM update
- Security settings overwrite

6.1.3.3. eNVM or sNVM Update (Ask a Question)

There are two main uses for eNVM or sNVM update:

- The OE receives an update from the firmware development team and updates the firmware in a Job Manager project.
- The OE performs a custom update of the existing eNVM or sNVM clients. Examples of this case could be programming of custom per-device information or information that becomes available or changed after the device was initially programmed.

eNVM or sNVM update is applied to the existing clients in the loaded design. The updated content size cannot exceed the client size in the design, but may be smaller. This allows the Libero design engineer to reserve more client space to accommodate a potential increase in firmware size.

6.1.3.4. Security Overwrite (Ask a Question)

This use case is designed to give the OE full control over security programmed into the device. The user can import a SPM file created by Libero into the Programming Data entry of a Job Manager project. This new SPM file, called Security Overwrite, substitutes all security settings imported into the Programming Data through the JDC file. The key values are still used from the KeySet file associated with the Programming Data, as all keys are protected by the HSM.

6.1.3.5. Initialization of Initiater Programming Bitstream (Ask a Question)

The Initiater programming bit stream is similar to the Initiater bit stream used in Libero, but is based on secured key loading using the Authorization Code Protocol (see Authorization Code Protocol for protocol description), which makes initial key loading secure and provides overbuilding protection.

The OE uses the Initiater bit stream to program security and any other optional bit stream components (Fabric, eNVM) into a blank device that does not have user security programmed.

Initialization of the Initiater bit stream automatically includes all protocol data generated by the U-HSM on the Job Manager side. This data is then passed and used by the M-HSM on the FlashPro Express (Manufacturing) side.

The Initiater bit stream programs security settings based on SPM information in the Programming Data. If Programming Data has a Security Overwrite (explained in Security Overwrite), the overwrite supersedes the SPM from the original design. Key values are always used from the KeySet file associated with the Programming Data.

When using the Initiater Bitstream in the SPPS HSM flow, note the following:

- After initial key loading, the Initiater bit stream disables all Factory Default Key modes and Factory pass keys. This gives the user exclusive control over device access through programming interfaces with the user pass and encryption keys. Note that the user-defined pass key and encryption keys are only accessible through the U-HSM. The M-HSM can only access the user-defined pass keys and encryption keys with the authorization of the U-HSM through the Job Ticket embedded in the Programming Job.
- The user has to select the “Protect factory test mode access using FlashLock/UPK1” or “Permanently protect factory test mode access” in the SPM.
- An ERASE or VERIFY operation using the Initiater bit stream requires an M-HSM. This is because after initial programming, device security is always locked by UPK1. The UPK1 value is always protected by the HSM, and security unlock can only be done by the HSM through the One Time Pass Key protocol (OTPK) that securely transmits the UPK1 value to the device (see One Time Pass Key (OTPK) Protocol for details).
- A program action from the Initiater bit stream cannot be invoked on a programmed device. Security reprogramming can only be achieved by erasing the device first using the Initiater bit stream that was used to program the device.
- The ERASE programming action does not erase content of eNVM areas programmed by the Initiater bit stream. With this action, security settings will be erased, but eNVM content remains available for access.

The OE can choose to make any of the user-defined keys (UEK1/UPK1/UEK2/UPK2/UEK3/DPK) to be Project Wide keys or Per-device keys.

6.1.3.6. Project Keys (Ask a Question)

All devices programmed from the same job have the same key value programmed. These keys are used from the KeySet file without modification.

6.1.3.7. Per-Device Keys (Ask a Question)

If the OE specifies any of the SPM keys to have a per-device value, every device receives a unique value derived from the respective base key in the KeySet file and Device Serial Number (DSN) of the device being programmed. This type of key derivation is referred to as a Per-Device Key Protocol (see Per-Device Protocol).

As actual device programming is done on the Production side, the programming software reads the DSN from the device and passes it to the M-HSM. The HSM uses the Per-Device protocol to derive per-device key value. This also applies to Erase and Verify operations, if UPK1 is selected to be per-device.

For more information about how to use the Initiater bit stream, see Job Manager User Guide.

6.1.3.8. Initialization of UEK1/UEK2/UEK3 Update Bitstream (Ask a Question)

Update bit stream is designed to update devices already programmed using the Initiater bit stream (see Initialization of Initiater Programming Bitstream).

An Update bit stream can only update eNVM, sNVM and Fabric device features. Reprogramming security is explained in a special use case in Initialization of Initiater Programming Bitstream.

Depending on whether or not the device has security locks on areas targeted for update, and whether project or per-device keys are used, the OE uses one of the use cases listed as follows:

- UEK1/UEK2/UEK3 Project Keys, No Security Lock

In this case, all devices in the project have the same UEK1, UEK2, or UEK3 values and target device feature programming is allowed without FlashLock/UPK1 match. The Job Manager can generate a stand-alone bit stream file or non-HSM programming job that does not require the M-HSM during programming. However, this case assumes the Job Manager uses the U-HSM to encrypt the resulting bit stream using UEK1/UEK2/UEK3 token values in the KeySet file.

- UEK1/UEK2/UEK3 Project Keys, Security Locked

If the target device feature programming is locked, the M-HSM must perform a secured unlock of the device, because plain text value of the lock key cannot be used in an untrusted environment. This type of bit stream must be handled through an HSM Programming Job using the OTPK protocol (see One Time Pass Key (OTPK) Protocol). The OTPK protocol is engaged automatically by FlashPro Express.

- UEK1/UEK2/UEK3 Per-Device Keys, No Security Locks, DSN is Available to OE

If UEK1, UEK2, or UEK3 are per-device keys, target feature programming is not locked, and DSN for the target device is known, the user has an option to generate a device-specific programming bit stream that does not require the M-HSM during production. This bit stream can be exported as a stand-alone bit stream file or non-HSM Programming Job. In either case, DSN must be provided upon bit stream export or during the addition of the device to the job.

- UEK1/UEK2/UEK3 Per-Device, All Other Cases

For all other cases related to per-device UEK1/UEK2/UEK3, the M-HSM and HSM Programming Job must be used. If the target device features are locked by per-device UPK1, UPK1 unlock is performed securely through the OTPK protocol (see One Time Pass Key (OTPK) Protocol). For more information about Initializing the bit stream in the Job Manager, see Job Manager User Guide.

6.1.4. Job Preparation (Ask a Question)

HSM Job Preparation consists of two phases:

1. Job Creation: JTAG chain setup and device association with programming bitstream.
2. Add HSM data to the job (adding HSM Task to the job).

6.1.4.1. Job Creation (Ask a Question)

Upon creation of a job, the user specifies production type, which can be either FlashPro Express or In- House Programming (IHP). For more information, see, the Job Manager User Guide.

6.1.4.1.1. FlashPro Express Job (Ask a Question)

This job type uses FlashPro Express and the M-HSM in the HSM flow, and is based on the JTAG chain. The JTAG chain may have following types:

- Microchip device targeted for programing with Initiater or Update bit stream.
- Microchip bypass device not targeted for programming.
- Non-Microchip bypass device.

6.1.4.1.2. Secured IHP Job (Ask a Question)

General handling of the Secured IHP job type is similar to FlashPro Express Job type. The primary difference is that the Secured IHP job requires only one Microchip device targeted for programming. This is because the actual hardware setup is managed by the IHP system.

6.1.4.2. Adding HSM Data to the Job through HSM Tasks (Ask a Question)

The OE can execute a programming job using multiple Contract Manufacturers or by handing off production volume to the Contract Manufacturer in chunks. The OE can manage this by using the

HSM Tasks. For example, the OE creates a Programming Job and then creates and exports an HSM Task for each manufacturer.

Following are the main purposes of HSM tasks:

- Grant permission to execute a job to a specific physical M-HSM in the contract manufacturer's security world.
  - This is a part of the overbuild protection mechanism that excludes the possibility of replicating the job on any other M-HSM in the same security world.
  - A special handshake mechanism between the U-HSM and the M-HSM is designed to accomplish this, as explained later in this section.
- Enable the OE to limit the number of devices served by a particular programming action (that is, PROGRAM, ERASE, and VERIFY).
  - This is another component of the overbuild protection mechanism.
- Securely deliver keys and protocol data to the M-HSM for programming execution.

All HSM-related information listed above is handled by Job Tickets associated to the HSM Task. The OE adds the Job Ticket to every authorized programming action (PROGRAM, ERASE, and VERIFY) for the target device. This allows the user to separately manage those actions. The Job Ticket allows control of the overbuild protection mechanism. The OE turns Job Ticket overbuild protection ON or OFF and specifies how many devices the ticket is allowed to handle (for more information, see Job Manager User Guide).

6.1.4.2.1. HSM Task Flow (Ask a Question)

The following HSM task flow steps must be done by the OE (see Figure 6-3 for details):

1. Create an HSM Task for the Programming Job.
2. Create a Job Ticket for the programming bitstream action for the target Microchip programming device:
   - Programming actions that require HSM support need a valid Job Ticket to execute.
   - The maximum number of devices to be allowed for the ticket can be specified in a ticket. The user can select not to turn on overbuild protection for a specific ticket.
3. The OE executes the Job Request-Reply handshake protocol between the U-HSM and the M-HSM to bind Job Tickets of the HSM Task to the specific physical M-HSM:
   - The OE exports the file with the Job Request from the Job Manager.
   - The OE passes this Job Request to the CM.
     - The OE and CM agree on the transport type for delivering the Job Request, depending on their specific case and security policies.
   - The CM loads the received Job Request with FlashPro Express and M_HSM, and exports a Job Reply.
   - The CM sends the Job Reply file back to the OE.
   - The OE imports the received Job Reply into the HSM Task on the Job Manager side.
4. The OE generates and exports the HSM Job from the HSM Task and creates the SPPS Programming Job file.
   - The HSM job can only be executed on the physical M-HSM that was used to generate the Job Reply.
   - The generated job is sent to CM for execution.

See Job Manager User Guide for more information about the content of the HSM job file and HSM Task Flow steps.

6.1.5. Production (Ask a Question)

The CM receives the SPPS Programming Job file from the OE and creates a FlashPro Express project with the HSM Job in the SPPS Programming Job file. For the HSM job, some ticket information, such as ticket IDs and overbuild protection data, gets loaded into the specific M-HSM module that participated in the Job Request-Reply handshake protocol. The rest of the ticket-protected information, such as JTAG chain configuration and programming bitstreams, is loaded into the FlashPro Express project. For more information, see FlashPro Express User Guide.

After creating the FlashPro Express project, the job is ready for execution.

6.1.5.1. Overbuild Protection (Ask a Question)

For each execution of a programming action (PROGRAM, ERASE, and VERIFY) that has an associated job ticket with overbuild protection turned on, the M-HSM decreases the counter of the remaining devices for the ticket. The overbuild protection counters reside inside the HSM module and are physically uncloneable. When the allowed number of devices have been programmed, the M-HSM stops processing any further protocol request associated with the ticket.

6.1.5.2. Job Status (Ask a Question)

Job status can be requested in FlashPro Express during job execution. Job status can be printed out in the FlashPro Express log windows or exported into the Job Status file and sent back to the OE. Job status contains a list of job tickets loaded into the FlashPro Express project and the M-HSM. For tickets with active overbuild protection, status will show max allowed device numbers and current state of remaining device counters.

Job Status also includes device CoC. The CoC is generated and cryptographically signed by the device with a (symmetric) message authentication code. This information can be verified by the U-HSM on the Job Manager side. See Device Certificate of Conformance (CoC) for more information.

6.1.5.3. Device Authentication (Ask a Question)

FlashPro Express performs a Microchip device authenticity check at the beginning of each programming action. This check is embedded into the check chain procedure. A failed device authentication check will abort programming action. The device authentication protocol is explained in Device Authenticity Check Protocol.

6.1.5.4. Job Completion (Ask a Question)

Job execution can be stopped in two ways:

- Normal job ending upon finishing programming of the target number of devices
  - For example, all job ticket overbuild protection counters are exhausted
- Job termination
- If some job ticket overbuild protection counters still have devices to program, then both the types of job termination will do the following:
  - Remove job tickets from the M-HSM and archive them in a dedicated folder on the M-HSM.
  - Generate Job Status with the following data:
    - Device Certificates of Conformance generated by devices (see Device Certificate of Conformance (CoC)).
    - Ticket End certifiers. This is a cryptographically validated proof of removing ticket data from the M-HSM module. Validation of this data can be done by the Job Manager using the U-HSM. This check confirms that the job ticket can no longer be used by the M-HSM. See Job End Certifier Protocol.

6.1.6. Post Production (Ask a Question)

The post production step on the Job Manager side is optional. It is used to analyze and cryptographically verify validators of Job Status received from the Production side. This is done by

importing the Job Status file into the HSM Task. The Job Manager uses the U-HSM to check all status validators.

6.1.7. Special Cases (Ask a Question)

6.1.7.1. Test Execution of the HSM Job on Operation Side (Ask a Question)

As a part of HSM job preparation, the OE might want to test-execute jobs on his side. This test execution can be done using FlashPro Express connected to the same U-HSM that was used for job creation.

The U-HSM has the complete functionality of the M-HSM. This feature is called the M-HSM function of the U-HSM. Configuring the M-HSM function of the U-HSM is explained in the User HSM Installation and Setup User Guide. All other aspects related to HSM job execution by FlashPro Express using the M-HSM function is similar to job execution in a real Production environment.

6.1.7.2. Using Internal (Automatic) Job Request-Reply for HSM Job Test Execution (Ask a Question)

If the OE uses the same physical U-HSM to create the HSM job and test-execute it, it is possible to eliminate the user-initiated Job Request-Reply flow steps (see Adding HSM Data to the Job through HSM Tasks). In this case, an HSM Task is created in a special “INTERNAL” mode (explained in the Job Manager User Guide). This mode tells the Job Manager to automatically execute Job Request-Reply protocol for the user, without involving regular request processing steps on the FlashPro Express side (see Figure 6-3).

6.2. Non-HSM Flow (Ask a Question)

The SPPS tool Job Manager supports a non-HSM flow that can be used without the U-HSM and the M-HSM. In this flow, all key material is handled in plain text form and initial key loading is done using KLK-key mode.

The main objective for non-HSM flow is to give the OE a way to prepare non-HSM jobs outside the Libero design tool. This eliminates the need for the user to be familiar with design process and general use of Libero, having to install Libero and eliminates Libero licensing requirements. It allows eNVM update to be handled within the Job Manager. Security can also be overridden at the Operation level similarly to the main HSM-based flow.

This flow is capable of producing a standalone bitstream file (as shown in Figure 6-5) or non-HSM jobs (as shown in Figure 6-6).

Figure 6-5. Non-HSM SPPS Flow for Bitstream Generation

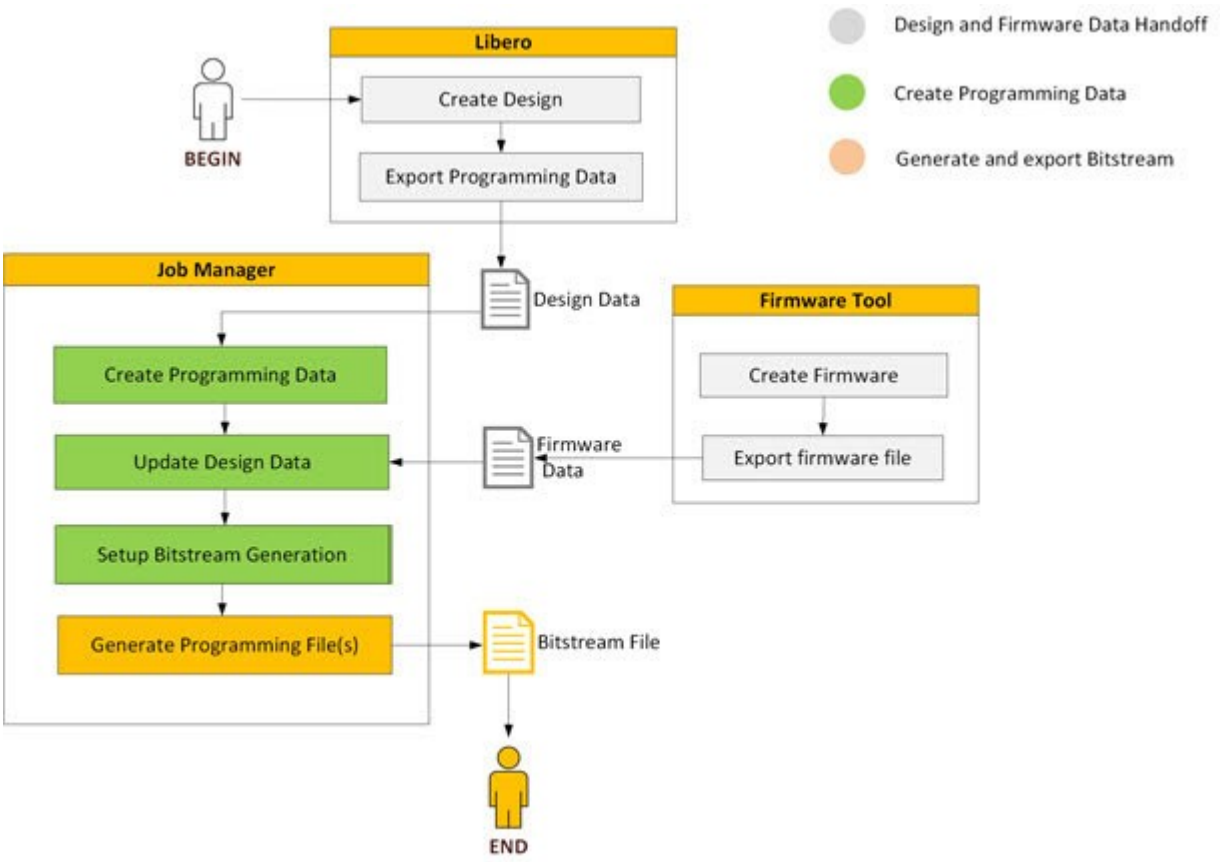
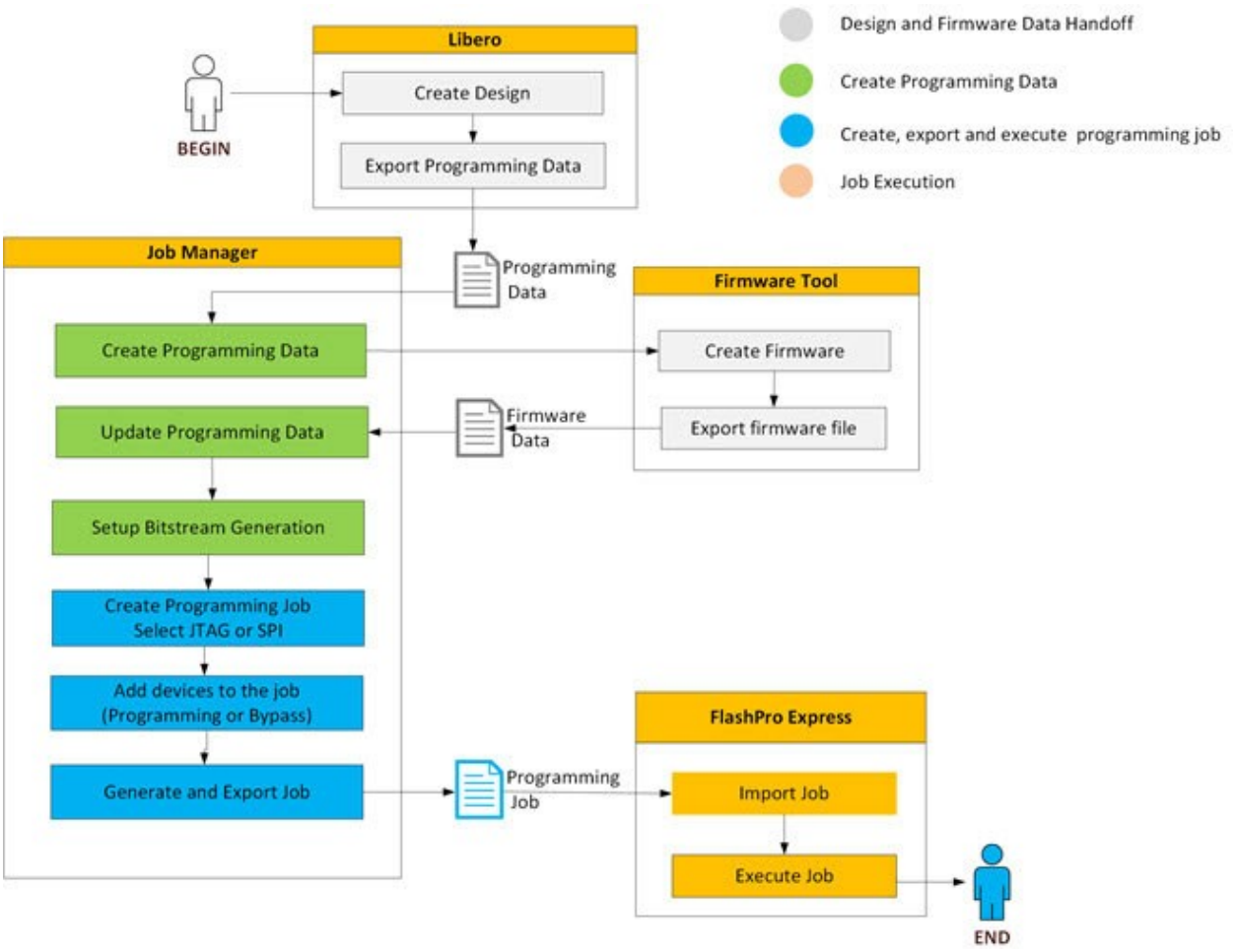


Figure 6-6. SPPS Flow for non-HSM Job Export



In both the cases, Design and Firmware data handoff is done similarly to the Main SPPS Flow (see Libero Design Handoff and Firmware Handoff).

Programming Data entry is created as in the main flow step (see Programming Data and Bitstream Initialization), with some differences in security management, explained as follows.

6.2.1. Security Management in Non-HSM Flow (Ask a Question)

By default, all security settings and key values are used from the design data loaded from the JDC file.

The user can overwrite security settings from an external SPM file received from Libero. This feature is similar to the one in the main flow (see Security Overwrite).

In addition to Security Overwrite, the non-HSM flow supports Key Overwrite.

6.2.1.1. Key Overwrite (Ask a Question)

Key Overwrite is available in non-HSM flow only. Key Overwrite allows the OE to replace one or more key values (pass keys, encryption, or debug key) received from Libero in the original JDC file or from applied security overwrite from the SPM file. Overwritten key values always supersede any key values in Programming Data.

6.3. Bitstream Initialization (Ask a Question)

Non-HSM flow supports the following programming bit stream options:

- Trusted Facility Bitstream
- Initiater Bitstream
- UEK1/UEK2/UEK3 Update Bitstream

6.3.1. Trusted Facility Bitstream (Ask a Question)

This bitstream type can program Fabric and/or eNVM. It does not program device security. The entire bitstream is encrypted with the KLK encryption key.

6.3.2. Initiater Bitstream (Ask a Question)

Supports initial programming of Security and other optional components such as Fabric and eNVM. The entire bit stream is encrypted with the KLK encryption key.

After initial programming with user keys, all Microchip factory default key modes, including KLK, DFK, KFP, and KFPE key modes, become disabled. After this step, the device becomes secured, and further updates can be applied in an untrusted environment (see UEK1/2/3 Update Bitstream).

Notes:

- Per security policy, programming of UEK1 or UEK2 programs the UPK1 or UPK2 passkeys respectively, and lock security segment. As a result, ERASE and VERIFY actions in generated bit stream files or programming jobs contain plain text UPK1/UPK2 values. This is required to unlock security segments for the programming actions when using the non-HSM flow (the HSM- based flow uses encrypted one-time passcodes).
- Factory ECC key modes (KFP, KFPE) are only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.

6.3.3. UEK1/2/3 Update Bitstream (Ask a Question)

A device programmed with a Initiater bit stream can later be updated using an Update bit stream. This bit stream can reprogram Fabric and/or eNVM device features only. Device security cannot be updated in this flow.

Notes:

- If Fabric or eNVM device components targeted by this bit stream are protected by FlashLock or UPK1, the plain text value of UPK1 will be included in the exported bit stream file/programming job when using the non-HSM flow.
- UEK3 is only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.

6.4. Creation of Programming Bitstream Files (Ask a Question)

The bitstream file can be exported after the bitstream initialization step is complete. The Job Manager supports different file formats (STAPL, SPI, DAT, and so on). For more information, see the Job Manager User Guide.

The generated bitstream file can be sent for production to program the devices using FlashPro, DirectC, Silicon Sculptor, IAP/ISP services, or other third-party programming tools that support STAPL format.

6.5. Creation and Execution of non-HSM Jobs (Ask a Question)

The creation of a non-HSM job is similar to an HSM Job with the exception that there is no IHP job type option, and the job is exported directly without HSM Tasks (see Adding HSM Data to the Job through HSM Tasks).

The user adds target and bypass devices. Target Microchip devices are associated with the bitstreams in Programing Data.

Job generation is done at the job level (explained in the Job Manager User Guide). The created job is sent to Production and can only be executed by FlashPro Express programming software.

6.6. Creation of SPI Directory (Ask a Question)

SPPS provides a way to create an SPI directory similar to that of the Libero tool. For more information, see the Libero SoC Design Flow User Guide.

The SPI directory allows the user to set up addresses for device self-programming images (golden and update).

This operation does not require HSM support and is not bound to a particular Job Manager project. See Job Manager User Guide for details.

7. SPPS Protocols (Ask a Question)

This section describes the protocols used by SPPS. It includes device-supported protocols, such as Authorization Protocols and OTPK, and protocols created above the device level, such as the Per-device key protocol.

7.1. Authorization Code Protocol (Ask a Question)

This is the main protocol used by SPPS. Authorization Code protocol support is built into SmartFusion 2, IGLOO 2, PolarFire, and PolarFire SoC devices. It allows SPPS to securely program initial security settings into the blank device, in an untrusted environment where simple monitoring or man-in-the-middle attacks would otherwise be possible.

This protocol is used by the Initiater bit stream in the HSM flow (see Initialization of Initiater Programming Bitstream). It is also used by the UEK1/2/3 Update bit stream to program devices that have per-device UEK1/2/3 keys programmed into them (see Initialization of UEK1/UEK2/UEK3 Update Bitstream).

The Authorization Code protocol enables secure delivery of a randomly generated encryption key (IP key, and KIP) to the device. The device can use the delivered KIP to decrypt an incoming programming bit stream. KIP remains in the device only until the end of the programming cycle, at which point it is erased—it is not stored in the device's nonvolatile memory.

A KIP-encrypted user bit stream can only be programmed into those devices which are "authorized" to be programmed by this protocol.

KIP can only be delivered to the device if encrypted with one of the keys known to the device. In the case of a blank device, there are two such keys, depending on the type of device that is supported by SPPS.

- A per-device Factory Key (FK) that is pre-placed in the device by Microchip during its manufacture.
- A per-device Factory ECC public private key pair that is available in larger SmartFusion 2, IGLOO 2 devices (M2S060, M2GL060, M2S090, M2GL90, M2S150, and M2GL150), and all PolarFire and PolarFire SoC devices. The Factory ECC public private key pair are pre-placed in the device by Microchip during its manufacture. See UG0443: SmartFusion2 and IGLOO2 FPGA Security and Best Practices User Guide and PolarFire Family Security User Guide for details about this key pair.
- A per-device User ECC public private key pair generated by the device internally prior to loading of the bit stream. See PolarFire Family Security User Guide for details about this key pair. The User ECC public private key pair is available only for programming using the Initiater Programming Bitstream.

If user security settings have already been programmed into the device, the Factory Key modes and User ECC key modes are disabled. For such a device, the user can only program FPGA, eNVM, and sNVM components with the Update bit stream using one of the user keys that was loaded with the Initiater bit stream. If the encryption key that must be used during this update is a per-device key, the Update bit stream is encrypted with KIP, and KIP is sent to the device through the Authorization Code protocol, which is protected by one of the user keys that was loaded.

The following sections provide more information about using the Authorization Code protocol during initial key loading and update flows.

7.1.1. Authorization Code Protocol in Initial Key Loading (Ask a Question)

SPPS initial key loading through the Initiater bit stream is based on the Authorization Code protocol. This protocol makes use of Diversified Factory Key (DFK) or Factory ECC Public Key Modes, which are described in the following sections.

7.1.1.1. Diversified Factory Key (DFK) and DFK Database (Ask a Question)

Following SmartFusion 2 and IGLOO 2 devices: M2S005, M2GL005, M2S010, M2GL010, M2S025, M2GL025, M2S050, M2GL050, have a unique Factory Key programmed into it. Microchip customers who use the SPPS HSM-based solution are given a database (DB) of the Diversified Factory Keys (DFKs) upon registering their U-HSM through the Microchip Portal. Upon registration, a UUID is assigned to the user HSM. The UUID is a 32-hex symbol string identifier (40-for M-HSM). The Customer UUID is used to diversify device Factory Key values to build a Diversified Factory Key Database (DFK DB) for the customer. Diversification is a non-reversible cryptographic operation that takes the device-unique Factory Key and Customer UUID as inputs. The resulting key is called the Diversified Factory Key or DFK. The DFK value is irreversible, unique per-device and per- customer, and is stored in the DFK database in encrypted form. Encryption is done using U-HSM and M-HSM Public Keys, thus allowing only HSM modules to directly access plain text DFK values.

Blank devices do not have knowledge of the Customer UUID, so it is passed to the device through the authorization code. The UUID is authenticated but not encrypted. The device then uses the value of the Factory Key and received UUID to derive the DFK value.

The process of obtaining a DFK DB by the OE, using it for test runs, and preparing for the M-HSM in production is described in the setup section of the User HSM Installation and Setup User Guide.

7.1.1.2. Factory ECC Public Key Modes (Ask a Question)

Larger SmartFusion 2, IGLOO 2 (M2S060, M2GL060, M2S090, M2GL90, M2S150, and M2GL150), and PolarFire and PolarFire SoC devices have Elliptic Curve Cryptography (ECC) hardware IP along with the SRAM-PUF, which allows these devices to have a unique-per-device factory enrolled ECC public private key pair. This allows customers to use the factory ECC public key instead of the Diversified Factory Key (DFK) for initial key loading

To use these key modes, the customer uses the Elliptic Curve Diffie-Hellman (ECDH) protocol to securely establish a shared secret key, which is used to encrypt the authorization code. The ECDH operation occurs within the security boundaries of the HSM and the FPGA, and the shared secret key is not known to the outside world. Before the HSM uses the Factory ECC Public Key from the device, it validates the public key by verifying the signature on the device certificate using Microchip keys. This ensures that the public key is valid and that it belongs to a Microchip device.

There are two available key modes:

- KFP, in which the device uses the certified key pair and the HSM uses a randomly generated ephemeral key pair. They follow the ECDH protocol to derive the shared secret key.
- KFPE, in which the device uses the certified key pair along with a second randomly generated ephemeral key pair, and the HSM uses two randomly generated ephemeral key pairs. In this case, the ECDH protocol is run twice, which results in two shared secret keys that are used in another round of key derivation to generate a single shared secret key. This key mode is preferred over KFP key mode, as it uses randomly generated key pairs and therefore is more secure. However, this key mode takes longer, because there are two ECDH operations and key generations.

For more information, see UG0443: SmartFusion2 and IGLOO2 FPGA Security and Best Practices User Guide and PolarFire Family Security User Guide.

7.1.1.3. Initial Key Loading of Project Keys (Ask a Question)

Bitstream Initialization

By selecting Initiater bit stream in the HSM-based flow, the user instructs the bit stream generation software to use the Authorization Code protocol with either DFK or Factory ECC key mode. The generated bit stream has a special programing algorithm for requesting the Authorization Code component from the M-HSM. The Authorization Code bit stream is generated dynamically by the M-HSM during production. The Authorization Code component is cryptographically linked to the programming bit stream, so it cannot be used with any other bitstreams, if intercepted during

programming. Additional protocol-specific information is created for use by the M-HSM during generation of Authorization Code component.

HSM Job Export

Exported programming job includes the following protocol-specific information:

- Instruction to use Authorization Key Protocol along with the key mode (DFK, KFP, or KFPE)
- Header for the Authorization Code component generation and its binding with the associated bit stream
- KIP value encrypted with the Ticket Key (see Adding HSM Data to the Job through HSM Tasks)

Job Execution

In the beginning of device programming, a programming algorithm reads the Device Serial Number (DSN) and makes a request to FlashPro Express software to provide the Authorization Code.

FlashPro Express, based on the context of the current programming action (device, programming action), retrieves ticket information, encrypted KIP value, and other controlling data and passes them to the M-HSM as a part of the request for the Authorization Code.

If the key mode for Authorization code is DFK, the M-HSM finds the encrypted DFK value in its DFK DB and passes a request to the secure execution engine (SEE machine) inside its HSM module. The firmware running in the SEE machine in the HSM decrypts KIP, the bit stream binding information, Authorization Code Header, and assembles the Authorization Code component. As the last step, it decrypts DFK and uses it to encrypt the resulting Authorization Code bit stream component.

Programming software shifts the Authorization Code component into the device and then shifts in the main programming bit stream components. In the case of programming a new device with a initiater bit stream, the device uses its pre-placed factory key (FK) and the UUID from the Authorization Code header to compute DFK. It uses DFK to authenticate and unwrap (decrypt) KIP from the encrypted payload of the Authorization Code. Finally, it uses KIP to authenticate and decrypt the main initiater bit stream components (for example, the security component and the optional FPGA fabric and eNVM components).

If the key mode for authorization code is KFP or KFPE, the M-HSM generates an ephemeral ECC key pair and exchanges the public key with the device. It then uses the ECDH protocol to securely derive the shared secret key inside the secure execution engine (SEE machine). The firmware running in the SEE machine in the HSM decrypts the KIP, the bit stream binding information, and Authorization Code Header, and assembles the Authorization Code component. As the last step, it uses the shared secret key to encrypt the resulting Authorization Code bit stream component. Programming software shifts the Authorization Code component into the device and then shifts in the main programming bit stream components. In the case of programming a new device with a initiater bit stream, the device uses the shared secret key that it generated when running the ECDH operation to authenticate and unwrap (decrypt) KIP from the encrypted payload of the Authorization Code. Finally, it uses KIP to authenticate and decrypt the main initiater bit stream components (for example, the security component and the optional FPGA fabric and eNVM components).

In the case of updating a device using a unique-per-device user key, the device retrieves the selected user key from its NVM security segment and uses it to authenticate the Authorization Code and unwrap KIP from its payload. KIP is used to authenticate and decrypt the main update bit stream components containing one or both of the FPGA fabric and eNVM components.

Initial Key Loading of Per-device Keys

This flow is the same as Initial Key Loading with Project keys, but with additional steps as follows:

The Job Manager-generated Initiater bit stream does not include a complete security component. This is because, the per-device key values can only be derived during programming, once DSN is known. Therefore, the security component is passed to the M-HSM as a pre-filled template through data structures protected by the Job Ticket. In addition to the standard Authorization Code

protocol data, HSM module attached to the M-HSM receives the security component template along with the base key(s) (originally from the KeySet file) and derives actual per-device key(s) upon programming (see protocol details in Per-Device Protocol). Derived keys are inserted into the template of the security component. Then the HSM module firmware links all the bit stream components cryptographically and proceeds to the same steps of project keys listed in Initial Key Loading of Project Keys).

Update Flow with Per-device Encryption Keys

This flow is also similar to SPPS Main flow that does the initial key loading of project keys. The difference is that the Authorization Code is encrypted with the same per-device UEK1, UEK2, or UEK3 derived by the M-HSM upon initial programming.

In this case, ticket information passed to the M-HSM includes the base key value for the per-device key to be derived. This value is originally taken from the KeySet file on the U-HSM side. Firmware in the HSM module attached to the M-HSM will use the received DSN and the base key in the Per-device protocol (see Per-Device Protocol) to derive the actual per-device key value. That value then encrypts the Authorization Code component that is sent to the device in a way similar to Initial Key Loading of Project Keys. The Authorization Code is followed by the bit stream components (FPGA or eNVM) that are being updated.
Note: This flow does not require DFK DB on the M-HSM side.

7.1.2. Overbuild Protection (Ask a Question)

The Authorization Code protocol allows the user to limit any programming action to a specific maximum number of devices.

This is possible due to following:

- The target devices do not have KIP and KIP needs to be sent to every target device.
- KIP is not stored in the device even after being used during programming cycle.
- KIP is delivered encrypted using a per-device encryption key (DFK, KFP, KFPE, or per-device UEK1/2/3). Therefore, the same authorization code cannot be used on other devices.

The HSM hardware module owns overbuild protection counters as a part of the Ticket information and will stop generating new Authorization Codes when the maximum allowed number of Authorization Codes has been generated.

7.2. Per-Device Protocol (Ask a Question)

The Per-device protocol is used to calculate per-device values of UPK1/UPK2/UEK1/UEK2/UEK3/DPK keys. Per-device keys are built on top of standard SmartFusion 2, IGLOO 2, PolarFire, and PolarFire SoC key types. They are programmed into the device as standard UPK1/UPK2/UEK1/UEK2/UEK3/DPK keys. If any encryption keys are designated as per-device unique keys, the passcode that protects that key from overwriting is a per-device unique passcode.

The Per-device protocol derives device-specific keys from the base key and DSN. This enables programming user models in which programming operations can be authorized per certain devices based on their DSN.

Derivation of per-device key values is done using a one-way cryptographic operation that takes the base key and DSN as inputs by firmware running inside the HSM module. The U-HSM generated KeySet file contains base key values for UPK1/UPK2/UEK1/UEK2/UEK3/DPK key types, as needed. Those values are passed to the M-HSM through Tickets under protection of the Ticket key.

The per-device protocol is used by SPPS in following cases:

- Initial key loading through Initiater bit stream—to program per-device keys
- Update bit stream, if device has per-device encryption keys programmed— to generate the required Authorization Codes

- By the OTPK protocol—to unlock the device if per-device pass keys are used (see One Time Pass Key (OTPK) Protocol)

Note: UEK3 is only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.

7.3. One Time Pass Key (OTPK) Protocol (Ask a Question)

This protocol is supported by SmartFusion 2, IGLOO 2, PolarFire, and PolarFire SoC devices and is used to unlock overwrite-protection or no-verify security policy settings (if used) in:

- Erase/Verify actions in Initiater bit stream flow
- Program/Erase/Verify actions in Update bit stream, if Fabric and/or eNVM/sNVM have security locks.
  This protocol is designed for temporary security unlock during programming actions. Use of the HSM allows hiding of the plain text values of the pass keys.
  SPPS automatically uses the OTPK protocol with no user interaction required.

7.4. Device Certificate of Conformance (CoC) (Ask a Question)

Upon being programming, a device can generate cryptographically signed digests for each freshly programmed bit stream component. Component digests with HSM-generated validators (message authentication codes) are together called the Certificate of Conformance (CoC). The CoC can be used as proof of device programming with a specific design.

The validators in the CoC can be verified by the Job Manager using the U-HSM.

The device generates and returns CoC, if the customer requests it upon initializing the programming bit stream (see Initialization of Initiater Programming Bitstream and Initialization of UEK1/UEK2/UEK3 Update Bitstream).

7.5. Job End Certifier Protocol (Ask a Question)

Upon job completion (see Job Completion), Job Tickets are removed from the HSM module. For every removed ticket, the HSM module returns proof of ticket deletion. This information can be validated by the Job Manager using the U-HSM. This information is cryptographically protected and cannot be modified without detection.

Job end certifier is a guarantee that the programming job cannot continue and is a part of the overall overbuild protection mechanism.

7.6. Device Authenticity Check Protocol (Ask a Question)

SmartFusion 2, IGLOO 2, PolarFire and PolarFire SoC devices have built-in device certificates that allow Microchip programing software to verify device origin. This check is enforced by FlashPro Express upon each programing operation, as a part of a check (certificate) chain operation (see FlashPro Express User Guide for details).

For any device failing Authenticity Check, a warning is issued.

8. HSM Servers (Ask a Question)

This section provides information about User and Manufacturer HSM servers: purpose, deployment scenarios, and the key management scheme.

HSM servers provide a protected security environment that allows SPPS to:

- Generate and protect user encryption and pass keys, base keys, random nonces, and so on.
  - Application keys and their associated metadata are stored as encrypted key tokens on the HSM-server mass storage device (for example, hard drive)
- Execute cryptographic algorithms and protocols making use of the protected keys:
  - Generate protocol data (For example, Authorization Code bitstream components (see Authorization Code Protocol), and so on.)
  - Verify validators generated by other HSMs or devices (For example, CoC (see Device Certificate of Conformance (CoC), Job End certifier (see Job End Certifier Protocol), and so on)
  - Securely transmit information between HSM servers (see HSM Security Environment)

Secured key handling and protocol execution is done by the SEE firmware running inside the HSM hardware module (see Use of Hardware Security Modules (HSM)).

8.1. HSM Hardware Modules used by SPPS (Ask a Question)

SPPS uses nShield Edge (Figure 8-1) and nShield Solo (Figure 8-2) hardware security modules (HSMs) manufactured by Thales. Both modules carry a FIPS140-2 Level 3 security certificate.

nShield Edge is a USB-attached module.

Figure 8-1. nShield Edge HSM Module

nShield Solo is PCIe-based, and can be installed on regular and compact size PC boxes with a PCIe port.

Figure 8-2. nShield Solo HSM Module

nShield Edge HSMs have an integrated smart card reader. The included card reader for nShield Solo HSMs is external.

In performance, nShield Solo surpasses nShield Edge. nShield Solo is optimal for use in U-HSM for performance-intensive programming Authentication Code and one-time passcode generation. nShield Edge is optimal for use in M-HSM for handling lightweight bit stream generation operations. From the software and setup perspective, both the modules are interchangeable, and module type selection is typically based on specific use conditions and the size of the SmartFustion2/IGLOO2 devices they serve.

The HSM module has standard Thales-provided cryptographic algorithms and can execute custom algorithms within the security boundaries provided by the HSM module.

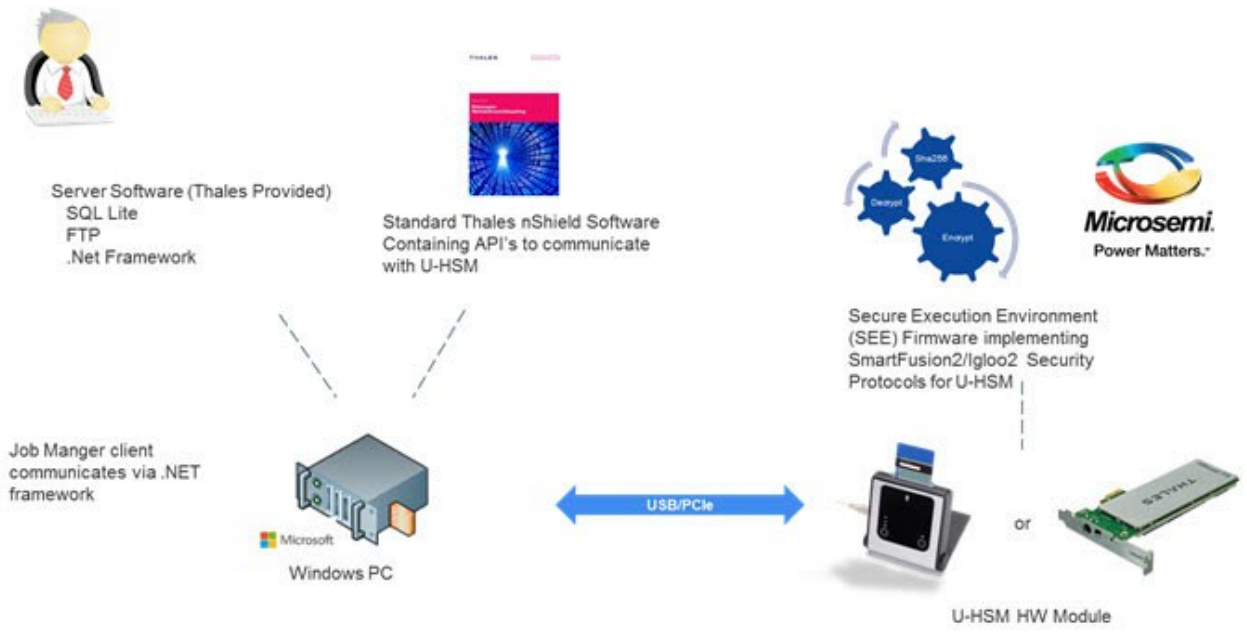
The HSM module has limited internal nonvolatile memory size for storing the module initiater key and Job Ticket information such as Ticket binding data, overbuild protection data, so on. All other information that requires protection by HSM is stored on the hard disk of the host PC.

For more information about nShield Edge and nShield modules, see the *nShield Edge Solo User Guide* from Thales.

8.2. HSM Server Architecture (Ask a Question)

The HSM module must be attached to the host PC running standard Thales nShield software. This software provides access to the HSM module. U-HSM and M-HSM servers have an architecture that is similar to the U-HSM architecture shown in the following figure, but the client software is FlashPro Express instead of the Job Manager.

Figure 8-3. HSM Server Diagram



The client application (Job Manager in this diagram) makes a request to the server. The HSM server Software running on the host PC receives the client request. Depending on request, it reads all required information from the database on the host PC and passes it through nShield software to the firmware running inside the HSM module. The HSM module decrypts received keys and processes the request. It encrypts output data and sends the response back to the HSM server. Some information is stored in the local database (for example, CoC). Client call-requested information is returned to the caller.

The SEE firmware is stored on disc of the host PC in encrypted form (see HSM Security Environment for details).

8.3. HSM Server Functionality (Ask a Question)

SPPS uses the User and Manufacturer HSM server types.

The User HSM allows the OE to generate keys and use them to generate programming bitstreams.

8.3.1. User HSM (Ask a Question)

The U-HSM provides the following functionality:

- Generated user keys
  - User encryption keys
  - User passcodes
  - Base Keys (for derivation of per-device encryption keys and passcodes, see Per-Device Protocol)
- Encrypt programming bitstreams
- Generate data for security protocols (see the Libero SoC Design Flow User Guide)
- Creation of Job Tickets
- Creation of Programming Jobs
- Validation of programming results (see Device Certificate of Conformance (CoC))
- Validation of programming job completion (see Job End Certifier Protocol)
- Preparation of device manufacturing data for the M-HSM, such as DFK DB and manufacturing keys (see the User HSM Installation and Setup User Guide)
- Test Job Execution (M-HSM function of U-HSM)
  - Complete functionality of the M-HSM
- Support for all underlying cryptographic algorithms and generation of cryptographic-quality true random numbers needed above

8.3.2. Manufacturer HSM (Ask a Question)

The M-HSM has software and firmware that is limited to job execution. The M-HSM is designed to be used by Production for the following:

- Creation of Job Tickets with binding to the physical HSM module (serving Job Requests)
  - Generation of protocol data (see the Libero SoC Design Flow User Guide)
- Device authenticity check (see Device Authenticity Check Protocol)
- Overbuild protection
- Secured initial key loading
- Providing job status
- Providing proof of job completion
- Collection of CoCs (see Device Certificate of Conformance (CoC))
  - Support for all underlying cryptographic algorithms and generation of cryptographic-quality true random numbers needed above

8.4. Deployment Scenarios (Ask a Question)

U-HSM and M-HSM servers are Windows based machines. HSM client applications, Job Manager, and FlashPro Express can be installed and run on either Linux® or Windows® systems.

On Windows platforms, client applications can be installed on the same or different physical operating systems. On Linux platforms, client applications are always installed on different systems.

There are two main HSM deployment scenarios: Server and Workstation.

For more information, see, the User HSM Installation and Setup User Guide and the Manufacturer HSM Installation and Setup User Guide.

8.4.1. Server (Ask a Question)

In the server deployment scenario, the HSM server runs on a dedicated PC. All client applications access HSM over the network.

One or more Job Manager applications can access the same U-HSM server at the same time, and one or more FlashPro Express applications can simultaneously use the same M-HSM.

Accessing HSM across the network requires certain ports to be open. For more information, see, the User HSM Installation and Setup User Guide and the Manufacturer HSM Installation and Setup User Guide.

8.4.2. Workstation (Ask a Question)

The workstation deployment scenario assumes client applications and HSM are sharing the same physical Windows system. Workstations in Operation or Production can be completely isolated from the network.

Note: U-HSM and M-HSM servers cannot be installed on the same physical Windows machine.

8.4.3. Notes about Using Virtual Machines (Ask a Question)

It is possible to run HSM (or client applications) inside virtual machines such as VMWare player. This provides the flexibility to create additional hybrid deployment scenarios. For example, Linux machine can run the Job Manager and VM with Windows system are installed for the U-HSM. This scenario enables the use of the same physical machine for both, client and server purposes. Such hybrid Workstation can be isolated from the external network.

Note: There are known issues related to using VMWare virtual systems. For information about workarounds proposed by Thales, see the *nShield Edge Solo User Guide*.

8.5. HSM Server Installation and Provisioning (Ask a Question)

U-HSM and M-HSM server installation and provisioning can be done manually or by using the setup utility provided by Microchip.

For information about obtaining components of the HSM servers and installation and provisioning instructions, see the User HSM Installation and Setup User Guide for the U-HSM and the Manufacturer HSM Installation and Setup User Guide for the M-HSM.

8.5.1. Use of the Microchip Web Portal (Ask a Question)

The Microchip Website is designed to provide SPPS customers with a convenient way to interact with Microchip Manufacturing HSM during the provisioning of the U-HSM and the M-HSM. It also allows Microchip IHP customers to initiate and control their programming jobs executed by IHP.

8.6. HSM Security Environment (Ask a Question)

This section provides an overview of the security environment used by the U-HSM and the M-HSM. It shows the key management scheme and interaction between the U-HSM, M-HSM, and Microchip Manufacturing.

The main components of the HSM security environment are:

- Security environment data (Security World)
- HSM Module connected to Security World
- Administrator Card Set (ACS)—used for administration of Security World

8.6.1. Security World (Ask a Question)

Security World is a key part of the HSM server. It is a security data domain physically residing on the disk of the host PC and used by the HSM module connected to it. Security World contains configuration data and security keys used by the HSM server.

Security World is introduced and supported by Thales. For complete documentation and descriptions of utilities for managing Security World, see the *nShield Edge Solo User Guide*. This document provides an overview about security keys utilized by the SPPS solution.

SPPS has installation and provisioning processes defined for the U-HSM (see the User HSM Installation and Setup User Guide) and the M-HSM (see the Manufacturer HSM Installation and Setup User Guide). The process includes instructions for creating a new Security World and keys used by the HSM. It also shows how to perform key exchange with other servers and how to handle DFK DB (explained in Diversified Factory Key (DFK) and DFK Database).

8.6.1.1. Security World Directory (Ask a Question)

All data related to Security World is stored in the Security World directory on the disk of the host PC. The actual path is determined during installation and defaults to:

```
C:\ProgramData\nCipher\Key Management Data\local
```

Note: "ProgramData" is a Windows system folder that is hidden by default.

8.6.1.2. Security World Data (Ask a Question)

The Security World directory contains all Security World data and can be backed up, restored, copied, or moved to another HSM as a whole. It contains HSM identity information and should be backed up periodically.

Security World data includes the following:

- Security World identity and configuration settings
- Data of HSM module(s) connected to Security World (the SPPS U-HSM and M-HSM support single module setup only)
- Data for the ACS cards associated with Security World
- Keys used by HSM internally
- Private component of the private/public key infrastructure for communication
- Public keys of the other HSMs for secure data exchange
- Microchip Manufacturing Keys (used during bitstream generation by the U-HSM and used during protocol execution by the M-HSM)

The following figure shows an example of U-HSM Security World directory content.

Figure 8-4. U-HSM Security World Directory Example

Note: The M-HSM has similar Security World data structure. The main difference between the two is the length of the UUID used by both HSM types (32 for the U-HSM UUID, 40 for the M-HSM UUID) and key name prefixes: "cu" for the U-HSM and "cm" for the M-HSM, respectively.

8.6.1.3. Security World Creation (Ask a Question)

New Security World creation instructions are provided in the User HSM Installation and Setup User Guide for the U-HSM and the Manufacturer HSM Installation and Setup User Guide for the M-HSM. This process also includes initialization of the Administrator Card Set (ACS). ACS cards are used for administrative actions performed on Security World, such as connecting new HSM modules, and so on.

Note: ACS can be set up to require a user-defined quorum of ACS cards for performing administration actions on Security World (see the *nShield Edge Solo User Guide* for details).

Information about Security World is placed into the file named "world", as shown in Figure 8-4.

Information about each card in ACS is stored in the file with prefix "card_", for example, `card_7fb2caf3494e585d4664febf7d624ab1b99e7d50_1`

8.6.1.4. Connecting HSM Module to Security World (Ask a Question)

The process of Security World creation makes use of the HSM module connected to the host PC. The HSM module is connected to Security World, and as a result, the Thales nCipher software creates a file that contains information about the module in the Security World directory. The file follows the pattern "module_<module serial number>", as shown in the example in Figure 8-4.

This HSM module can be replaced with another module at any time after Security World creation.

8.6.2. Key Management Scheme (Ask a Question)

This section explains the key management scheme used by both HSM server types in their interactions with each other, as well as the Microchip Manufacturing HSM Server.

All security keys used by Security World are stored in the Security World directory as key token files, as shown in Figure 8-4. All private data in key token files is encrypted, watermarked, and signed. These files are part of the FIPS140-2 Level 2 and 3 certified nShield key management system. For more information, see the *nShield Edge Solo User Guide*.

Security World Key access control lists (ACL) control which operations a key can be used for, if the key can be stored persistently as a key token after generation, if a key is recoverable by the Security World Security Officer, and other attributes of the key.

One of the features of SW Key ACLs is that is used in the SPPS is the SEE application key scheme. This scheme allows all crypto operations available to a key type to only be executed within a properly signed SEE application (HSM module firmware).

8.6.2.1. SEE Integrity Key (Ask a Question)

The HSM hardware module executes two types of the firmware:

- Standard nShield firmware provided by Thales.
- Custom firmware that implements Microchip security protocols and makes use of various keys created for the SPPS. This custom firmware is referred to as the SEE machine.

Standard firmware controls execution of the SEE machine. This control includes load, decryption, and security integration between SEE machine firmware, security keys used by SPPS, and data stored inside HSM module non-volatile memory (NVRAM).

- The Microchip SEE machine is signed by the private component of the SEE Integrity Key.
- The public component of the SEE Integrity Key is installed in the Security World directory upon HSM setup in the file `key_seeinteg_userdata_signer` (shown in Figure 8-4).
- All keys created and used by SPPS are cryptographically signed by the public component of the SEE Integrity Key. This signature carries the hash of the private component of the key, thus allowing standard nShield firmware to enforce ACL policy by matching signatures on the SEE machine and keys used by it.

8.6.2.2. Private/Public HSM Keys (Ask a Question)

For each HSM server, the user creates two private/public key pairs. One private/public key pair is used for encryption and decryption. These keys appear with the prefix "key_simple_g4cu_seepk" and "key_simple_g4cu_seesk" as shown in Figure 8-4. The public key is sent to the other HSM servers for encryption of the data sent back to this HSM. The HSM then uses the private key component to decrypt.

received data. The private component is a persistent key created in the Security World directory stored in the encrypted form.

The other private/public key pair is used for signing and verifying signature. These keys appear with prefix "key_simple_g4cu_seessk" and "key_simple_g4cu_seespk" as shown in Figure 8-4. The HSM uses the private key component to sign the data that is sent to other HSM servers. The public key is sent to other HSM servers for verifying the signature of data sent by this HSM, which allows checking the authenticity (that is, assurance of signatory's identity) and integrity (that is, tamper and corruption detection) of the data being imported. The private component is a persistent key created in the Security World directory stored in the encrypted form.

The public key of the HSM server carries cryptographic proof that it was created by an nShield HSM module. This binding is done through the HSM Warrant file and is checked by the other HSMs when importing this public key component. This mechanism makes it nearly impossible for the intruder to gain access to any HSM server used by SPPS by sending them a falsely generated public key.

8.6.2.3. nShield HSM Warrant (Ask a Question)

The HSM warrant is a file created by Thales. It is unique and included with every nShield Solo or Edge HSM module provided by Microchip as part of the module acquisition process.

Data inside the warrant file ensures that an nShield module with a specific module serial number (ESN) and KLF value has been certified by Thales.

8.6.2.4. Internal Security Key (ISK) (Ask a Question)

The ISK is a persistent Security World key generated by the customer. The ISK is stored in the Security World directory in a file with the default name `key_simple_g4see_isk`, as shown in the example in Figure 8-4. The actual name of the file is user-configurable per instructions provided in the User HSM Installation and Setup User Guide and the Manufacturer HSM Installation and Setup User Guide.

The ISK value is randomly generated by the HSM module and is unique within Security World. This key can only be used by the SEE application.

8.6.2.5. Import of Public Keys from Other HSMs (Ask a Question)

Before an HSM can communicate with other HSMs, the public keys need to be manually exchanged in a trusted manner. As a part of key exchange, every HSM server used by SPPS should import two public keys from each of the other HSMs. This mechanism mirrors the process explained in Private/Public HSM Keys. See the User HSM Installation and Setup User Guide and the Manufacturer HSM Installation and Setup User Guide, which explains how to import these public keys.

Public key import includes a strict check of the key generation certificate of the key being imported. This check validates that the key was generated by a valid HSM of the official SPPS application.

After the check has been done, the ISK is used to sign imported public key as proof of warrant check done upon the key import. This signature is required for the key to be used by the SEE application.

An example of the imported public keys is shown in Figure 8-4. This sample Security World has the following imported public keys:

- key_simple_g4cm-seepk-0000000000000000000000000000000000000002
  - This is the encryption public key of the M-HSM with UUID "0000000000000000000000000000000000000002"
  - Note that the M-HSM has UUID length of 40
- key_simple_g4cu-seepk-00000000000000000000000000000001
  - This is the encryption public key of this U-HSM
  - Note that the UUID length is only 32 hex symbols
  - This is to enable the M-HSM function of this U-HSM server
- key_simple_g4cm-seespk-0000000000000000000000000000000000000002
  - This is the verify public key of the M-HSM with UUID "0000000000000000000000000000000000000002"
  - Note that the M-HSM has UUID length of 40
- key_simple_g4cu-seespk-00000000000000000000000000000001
  - This is the verify public key of this U-HSM
  - Note that the UUID length is only 32 hex symbols
  - This is to enable the M-HSM function of this U-HSM server

8.6.2.6. Microchip Manufacturing Keys (Ask a Question)

These special Microchip keys are imported per device type as a part of obtaining the DFK DB. For more information, see the User HSM Installation and Setup User Guide for the U-HSM and the Manufacturer HSM Installation and Setup User Guide for the M-HSM.

Every imported key is created in a file starting with the prefix "key_simple_f4mf-" key. These keys are used by both the U-HSM and the M-HSM. They are sent to either HSM from the Microchip Manufacturing HSM. The key values are encrypted with the public key of the receiving HSM.

8.6.2.7. U-HSM Initiater Key (Ask a Question)

The U-HSM Initiater Key is created by the user. This is a symmetric key that allows the U-HSM to protect KeySet files and Job Ticket initiater keys.

The U-HSM Initiater Key encrypts the Job Ticket Initiater Key, which in turn protects all private Job Ticket data. To send ticket data to the M-HSM, the U-HSM only needs to rewrap the Job Ticket Initiater Key with the M-HSM public key.

8.6.2.8. DFK DB Handling (Ask a Question)

DFK DB is used by the M-HSM during initial key loading (see Diversified Factory Key (DFK) and DFK Database). It can also be used during a test run by the U-HSM.

DFK DB is generated by Microchip for the specific U-HSM UUID. All DFK values in the generated database are encrypted by per-device type DFK DB ticket keys. DFK DB ticket keys are protected by the U-HSM public key imported into the Microchip Manufacturing HSM during the previous steps. This scheme allows the U-HSM access to the DFK values in the received database.

To use DFK DB on the M-HSM side, the user exports the DFK DB ticket keys and sends them to the Microchip Manufacturing server for rewrap with the M-HSM public key. Rewrapped ticket keys are imported by the user back to the DFK DB. When the DFK DB has ticket keys encrypted with the M-HSM public key, it can be used by the M-HSM.

8.6.2.9. Job Handling (Ask a Question)

All private data and key material is used by the Job Manager during bit stream generation and job preparation is protected by the Ticket Initiater Key. This is a symmetric encryption key encrypted by the U-HSM Initiater Key (see U-HSM Initiater Key).

During HSM job export (Job Creation), all Job Tickets are rewrapped with the target the M-HSM public key. This grants the M-HSM access to all job data protected by the Job Ticket Initiater keys.

Other types of communication between the U-HSM and the M-HSM, such as Job Request/Reply protocol execution (Job Preparation) and returning job end sending job end certifiers from the M-HSM for validation to U-HSM, also make use of the counterpart HSM public key to protect private information.

9. Referenced Documents (Ask a Question)

This user guide references the following documents:

- Job Manager User Guide
- FlashPro Express User Guide
- Libero SoC Design Flow User Guide
- User HSM Installation and Setup User Guide
- Manufacturer HSM Installation and Setup User Guide
- UG0443: SmartFusion2 and IGLOO2 FPGA Security and Best Practices User Guide
- PolarFire Family Security User Guide
- nShield Edge Solo User Guide (Thales)

10. Acronyms (Ask a Question)

The following table lists the acronyms and definition.

Term	Definition
U-HSM	User HSM
M-HSM	Manufacturer HSM
CM	Contract Manufacturer
OE	Operation Engineer
DSN	Device Serial Number
DFK DB	Diversified Factory Key Database
IHP	In-House Programming
SPM	Security Policy Manager

11. Revision History (Ask a Question)

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision	Date	Description
M	05/2025	This document is released with Libero SoC Design Suite v2025.1 without changes from v2024.2.
L	08/2024	This document is released with Libero SoC Design Suite v2024.2 without changes from v2024.1.
K	02/2024	This document is released with Libero SoC Design Suite v2024.1 without changes from v2023.2.
J	08/2023	This document is released with Libero SoC Design Suite v2023.2 without changes from v2023.1.
H	04/2023	This document is released with Libero SoC Design Suite v2023.1 without changes from v2022.3.
G	12/2022	The following changes are made in this revision: • Updated the Programming Production section. • Updated the Main SPPS Flow section.
F	08/2022	The following changes are made in this revision: • Updated the Introduction section. • Updated the Secure Initial Key Loading section. • Updated the SPPS Functions and Services section. • Updated the Firmware Handoff section. • Updated the Authorization Code Protocol section. • Updated the Diversified Factory Key (DFK) and DFK Database section. • Updated the Factory ECC Public Key Modes section. • Updated the One Time Pass Key (OTPK) Protocol section.
E	04/2022	This document is released with Libero SoC Design Suite v2022.1 without changes from v2021.3.
D	12/2021	This document is released with Libero SoC Design Suite v2021.3 without changes from v2021.2.
C	08/2021	This document is released with Libero SoC Design Suite v2021.2 without changes from v2021.1.
B	04/2021	Editorial updates only. No technical content updates.
A	11/2020	Document formatted as per Microchip standards. Initial revision.

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-1240-4

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.