Lattice Sentry Firmware Signing Tool for Mach-NX Devices User Guide

Technical Note

FPGA-TN-02351-1.1

April 2024

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults, and all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Inclusive Language

This document was created consistent with Lattice Semiconductor’s inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice’s inclusive language FAQ 6878 for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
BMC	Baseboard Management Controller
CFG	Configuration Flash sector on Mach-NX device
CFGx	Configuration image located in either CFG0 or CFG1 of Mach-NX device
DICE	Device Identifier Composition Engine
ECDSA	Elliptic Curve Digital Signature Algorithm
FAM	Flash Address Map
FPGA	Field Programmable Gate Array
FST	Firmware Signing Tool
HSM	Hardware Security Module
IP	Intellectual Property
RISCV	Reduced Instruction Set Computer V
SFB	SoC Function Block
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
UFM	User Flash Memory

1. Introduction

This document is intended to be a guide for using keys to sign Sentry™ files for programming and updates.

The Lattice-developed Firmware Signing Tool (FST) is a collection of Python™ Scripts used to generate custom keys and sign firmware and configuration files. The FST signs the firmware .mem file and the configuration .jed and .bit files using the ECDSA keys.

In addition to providing a method to sign firmware and configuration files, the FST provides an example process of signing firmware and configuration files using standard signing interfaces. It may be adapted for use with commercial Hardware Security Module (HSM) systems or other key management systems by replacing the calls to OpenSSL with HSM interface calls, for example.

The firmware .mem file is programmed into the SPI Flash. The configuration .jed file is programmed into the Mach™-NX internal Flash sector, CFG0 and/or CFG1. The configuration .bit file can be programmed into the Mach-NX SRAM but is not used in the standard configuration flow because the SRAM is a volatile memory and will be erased upon a power cycle.

The Lattice Diamond™ Deployment Tool is a utility that can format a .jed configuration file so it can be used for an update. The output of the Diamond Deployment Tool is a specially formatted .bin file, which is then programmed into the Flash memory.

This document covers the following topics:

- Setup and installation requirements of the FST
- Complete list of files and program files created by the FST
- Generating .jed and .bit file in Diamond signed with dummy placeholder keys
- Using the FST to generate keys
- Using the FST to sign files
- Using the FST and Diamond Deployment Tool to generate a signed, formatted update .bin file
- Programming and testing the update .bin file

Note: This FST utility is to be used for firmware development and prototyping purposes only. The FST utility is not to be used for production phase for key pair generation or image signing. For production flow, a HSM or HSM service is required to generate a key pair and sign images to maintain the security of keys and signed images.

2. Software Requirements and File Descriptions

2.1. Software Requirements

The following software are required for the steps outlined in this guide:

- Bash shell environment with the following dependencies, there are several options available:
  - Linux® OS environment, such as Ubuntu 22.04
  - Windows 10 or above with Windows Subsystem for Linux (WSL) installed with an appropriate OS environment, such as Ubuntu 22.04
  - Windows 10 or above with Cygwin™64 installed. This is the configuration used in this document.
    - You can download Cygwin64 Terminal from its official website.
    - During the Cygwin64 installation, make sure to install *python3* and *openssl* by selecting the options shown in Figure 2.1 and Figure 2.2.

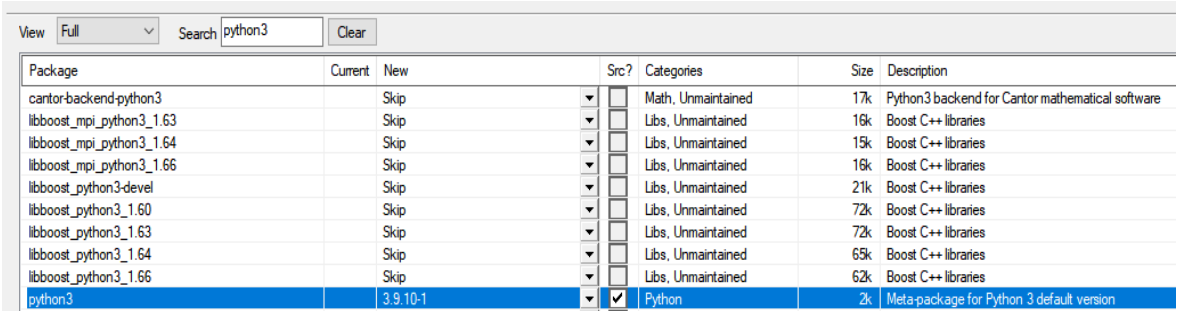

Figure 2.1. Install *python3* during the Cygwin64 Installation

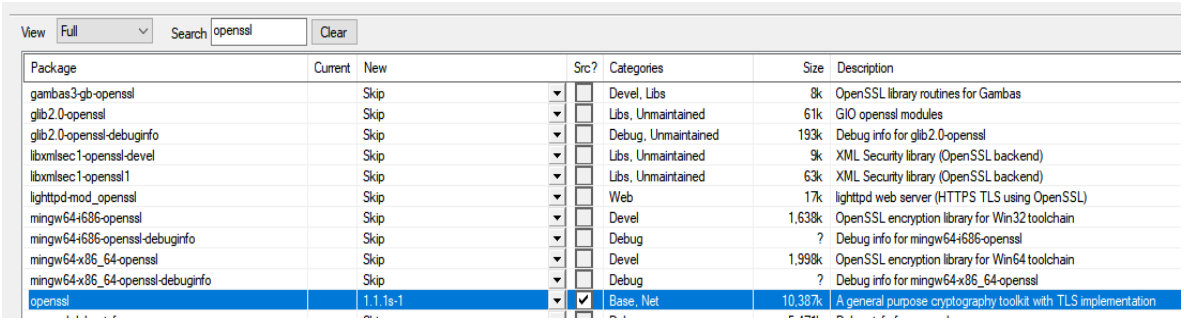

Figure 2.2. Install *openssl* during the Cygwin64 Installation

- Python™ 3.6 or above:
  - You can download Python from its official website.
- OpenSSL™ available in path (tested with OpenSSL 1.1.1f):
  - You can download OpenSSL from its official website.
- Lattice Diamond software:

Table 2.1. Lattice Diamond Software Versions

Software	Version
Lattice Diamond	3.13.0.56.2
Lattice Diamond Encryption Security Control Pack	3.13.0.56.2

- Lattice Diamond Programmer:

Table 2.2. Lattice Diamond Programmer Software Versions

Software	Version
Lattice Diamond Programmer	3.13.0.56
Lattice Diamond Patch (Required for CFGx Internal Flash Update, see Generating and Signing a Config Update File section)	3.13.0.56.2 131818
Lattice Diamond Programmer Encryption Security Patch	3.13.0.56

2.2. File Descriptions

The FST utility is available upon request. The FST utility is used to generate a public/private key pair, and to generate the signed software image for the RISCV and the signed config files.

2.2.1. Python Script Files

Python scripts are used to sign the firmware and configuration file images. This set of scripts is included with the FST utility. The file names and their descriptions are listed in Table 2.3.

Table 2.3. Python Script Files and their Descriptions

File Names	Descriptions
cfg_bin_sig_replace.py	Replace signature in full config bin and bit image
cfg_binjed_sig_replace.py	Insert .bin signature to .jed files
combine_fw_and_sig.py	Combine firmware .bin files and signature
convert_bit_to_sign_bin.py	Convert .bit files to binary files that can be signed
convert_fw_mem_to_bin.py	Convert .mem files to binary files that can be signed
convert_jed_to_bin.py	Convert .jed files to binary files that can be signed
openssl_DER_sig_to_raw.py	Convert OpenSSL output DER files to raw binary signature files

2.2.2. Bash Script and Readme Files

SH file is a script programmed for bash, a type of Unix shell. This set of files is included with the FST utility. The file names and their descriptions are listed in Table 2.4.

Table 2.4. Bash Script and Readme Files, and their Descriptions

File Names	Descriptions
sign_sentry_firmware_ecdsa.sh	Sign batch File for ECDSA
sign_sentry_firmware_hmac.sh	Sign batch File for HMAC
README_ECDSA.md	Readme File for ECDSA
README_HMAC.md	Readme File for HMAC

3. Sign .jed File with Dummy Key Pair

3.1. Diamond SoC Project

Refer to the following steps to generate .jed and .bit file in Diamond project signed with dummy placeholder keys.

1. Using the Diamond software, open the project that contains the generated CFG image file you want to sign.
2. Click on the **Tools** menu, then choose **Security Setting** as show in Figure 3.1.

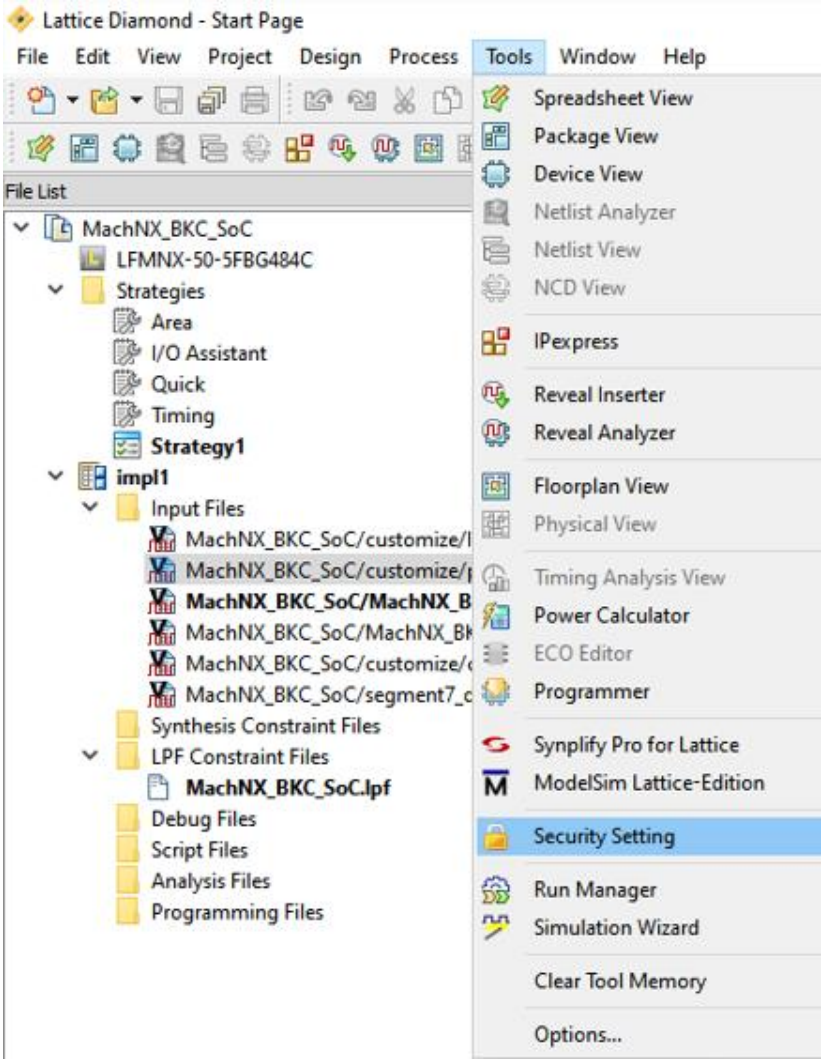

Figure 3.1. Select Security Setting

Note: Security Settings menu item requires Diamond Patch 131131. For more details about setting up the Lattice tools and environment for Sentry, refer to Lattice Sentry Demo Board for Mach-NX Walkthrough User Guide (FPGA-UG-02167).

3. Enter the password as shown in Figure 3.2. The default password to get to the Security Setting menu is *LATTICESEMI*. You may change the password if you wish. This password controls access to the key generation area in the Lattice Diamond tool.

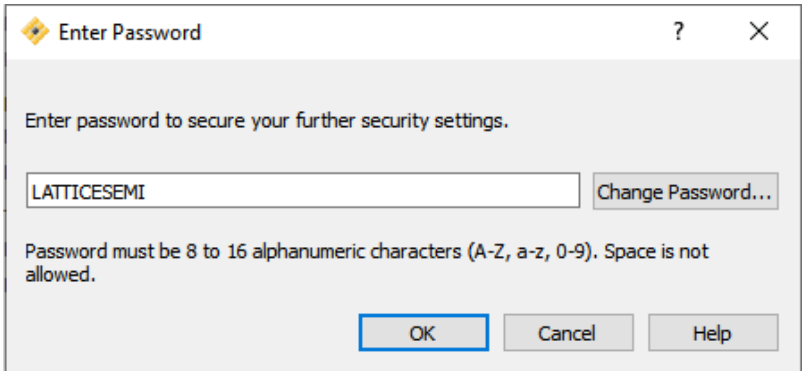

Figure 3.2. Enter the Password

4. In the Security Setting pop-up, select **ECDSA Authentication**. Click on **Auto Generated Key Pair** shown in Figure 3.3. A public/private key pair will be generated. These settings are sticky and will only need to be completed one time for this design project.

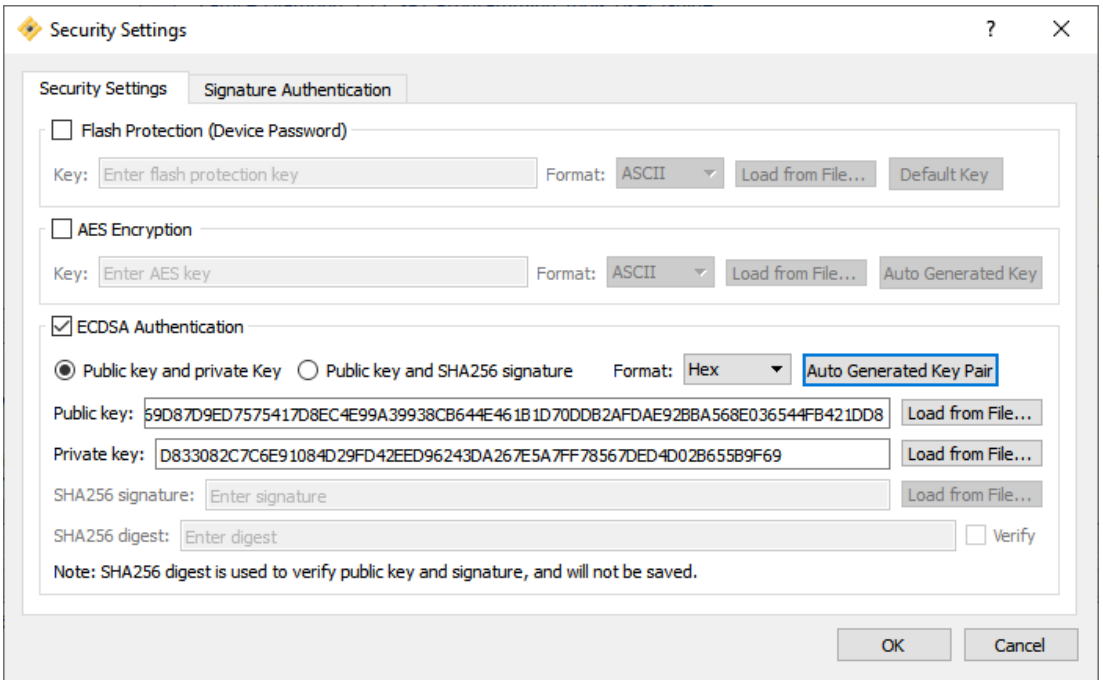

Figure 3.3. Select Auto Generated Key Pair

5. Click **OK** to close the Security Settings menu.
6. Select the **Process** tab. Under **Export Files**, click the **Bitstream File** and **JEDEC File** checkboxes as shown in Figure 3.4.
7. Double-click on the **Export Files** shown in Figure 3.4 to generate the signed .jed and .bit files. These files are signed with the auto generated key pair as a signature placeholder. This placeholder signature is later replaced with the signature generated by the customer key by the FST scripts.
8. The green check marks, shown in Figure 3.4, indicate that the processes have completed and the .jed and .bit files have been created successfully.

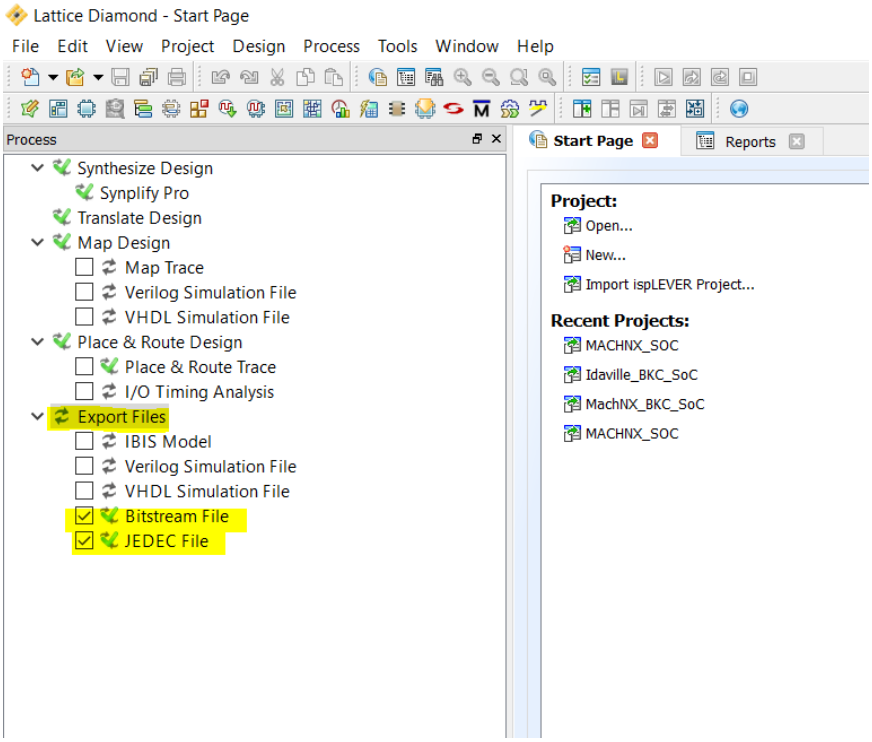

Figure 3.4. Click the Checkboxes for Bitstream File and JEDEC File

9. The .jed and .bit files can be found in the *impl1* folder in the project directory structure as shown in Figure 3.5.

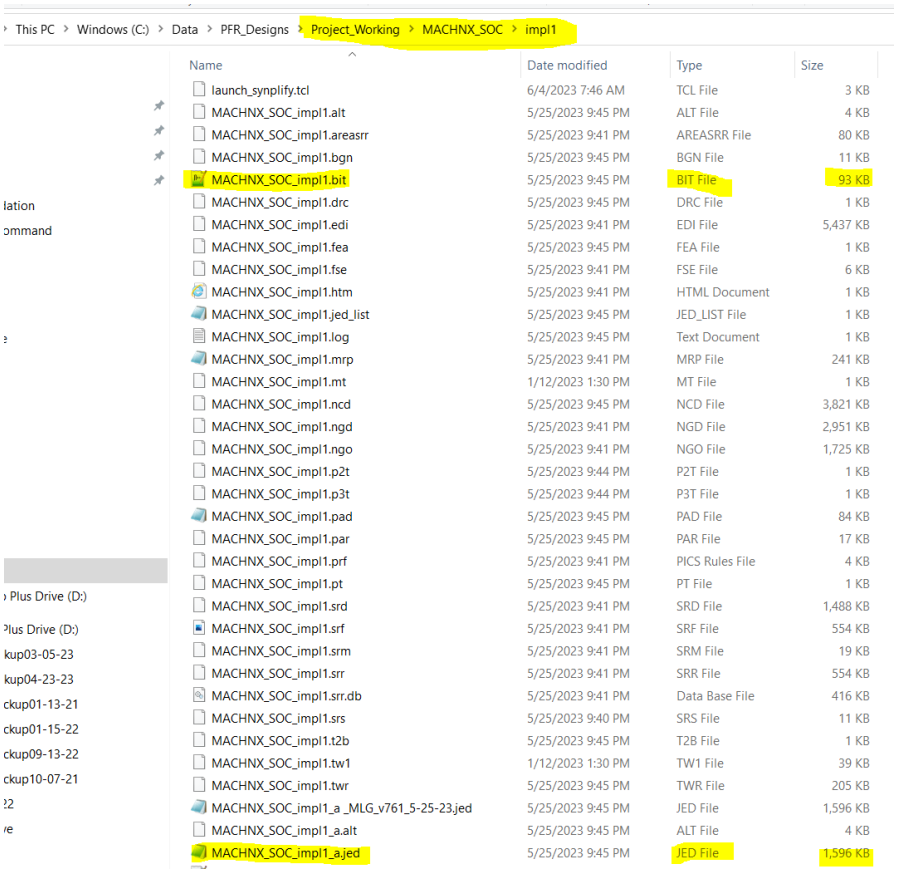

Figure 3.5. Generated .jed and .bit files are in the *impl1* Folder

The .jed file is used to program the CFG0 and CFG1 internal Flash sectors of the Mach-NX. The .bit file is used to program the SRAM cells via Diamond Programmer. SRAM is a volatile memory, so anything programmed here will be erased upon a power cycle. See Generating and Signing a Config Update File section for more information on how to generate .bit file for CFG update via firmware.

These steps could also be followed to sign the .jed and .bit files with an existing key pair. In this case, the FST utility would not need to re-sign them later. However, the firmware .mem file would need to be signed using the same key pair, which would require the FST utility. Lattice Propel™ software does not have an equivalent Security Settings menu option as Lattice Diamond software.

4. FST Script Usage

4.1. Generating New Keys

Refer to the following steps to generate new keys:

1. You can generate a fresh pair of public and private keys using the Cygwin64 Terminal, shown in Figure 4.1, or your preferred shell environment.

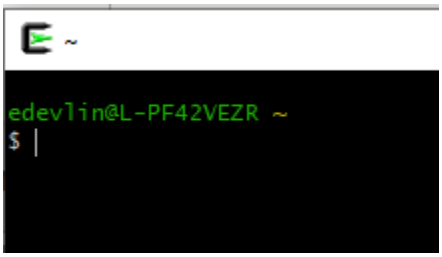

Figure 4.1. Cygwin64 Terminal

2. Navigate to the directory where the FST files are stored as shown in Figure 4.2.

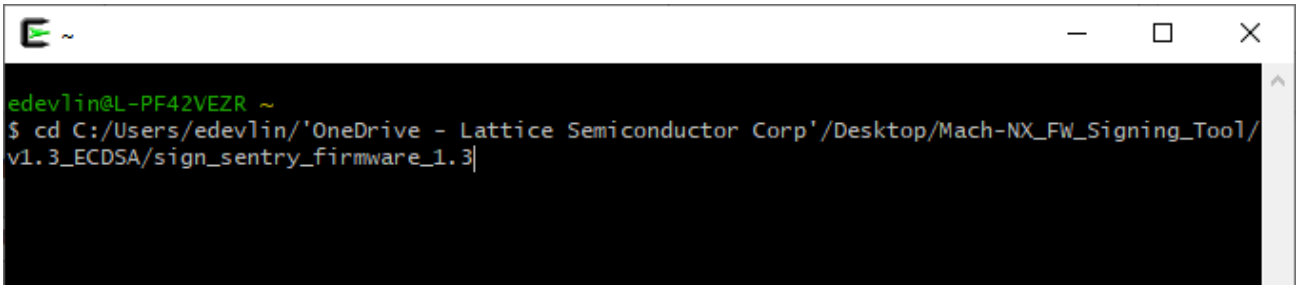

Figure 4.2. Navigate to the FST Files Directory

3. Enter the following command to the terminal shown in Figure 4.3.

Command:

```
./sign_sentry_firmware_ecdsa.sh -s key
```

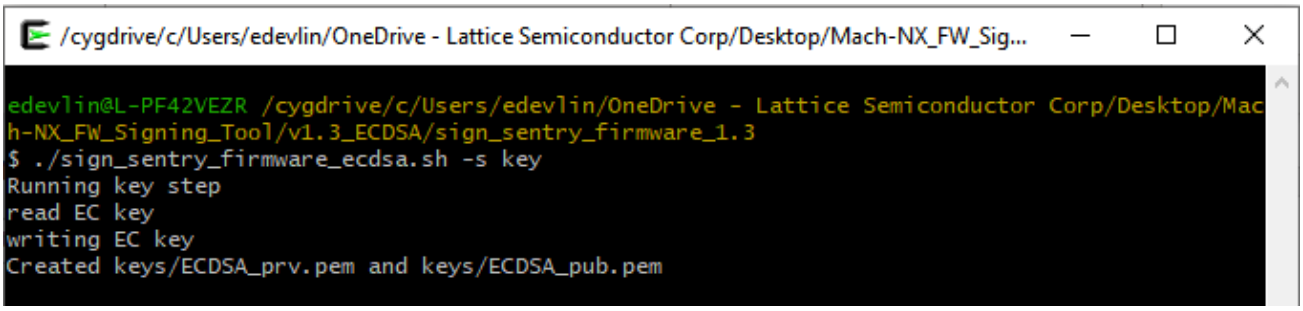

Figure 4.3. Enter the Given Command to the Terminal

4. Two new files are produced: *keys/ECDSA_prv.pem* and *keys/ECDSA_pub.pem* as shown in Figure 4.4.

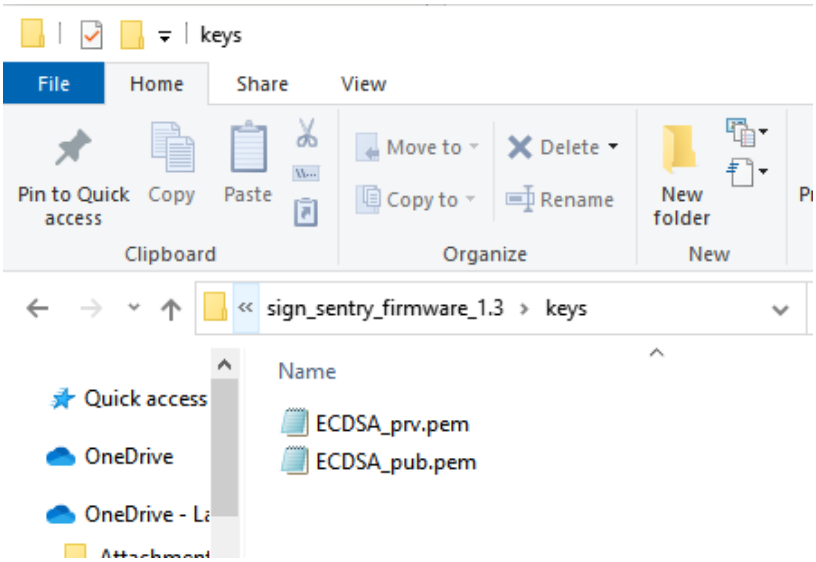

Figure 4.4. New Keys Generated

4.2. Using Existing Keys

Refer to the following steps to use an existing key pair with the FST:

1. Open the Cygwin64 Terminal and navigate to the directory where the FST is stored, as described in the Generating New Keys section.
2. Create a directory named keys using the following command as shown in Figure 4.5.

Command:

```
mkdir keys
```

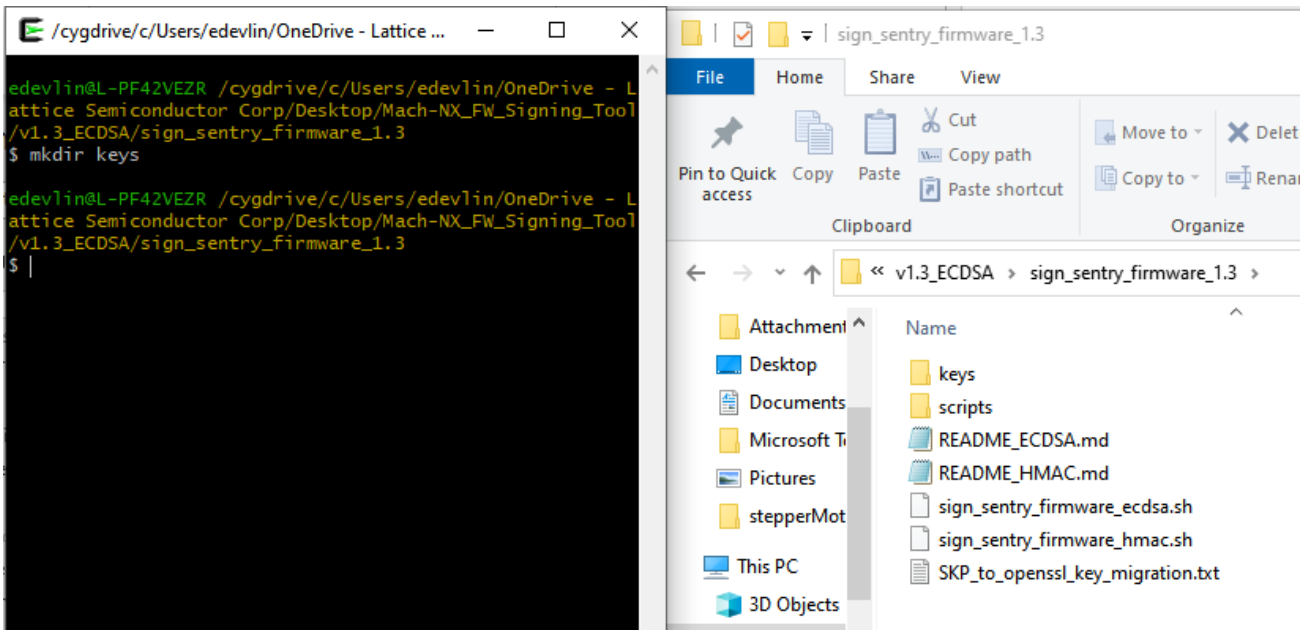

Figure 4.5. New Directory Name Keys Created

3. Enter the following commands to copy the existing keys' .pem files into the newly created directory and renaming them as file names described in Table 4.1.

Commands:

```
cp my_private.pem keys/ECDSA_prv.pem
cp my_public.pem keys/ECDSA_pub.pem
```

Table 4.1. Renamed Key .pem Files and their Descriptions

File Names	Descriptions
keys/ECDSA_prv.pem	ECDSA private key in pem format, used for signing
keys/ECDSA_pub.pem	ECDSA public key in pem format, used for verifying signature

4.3. Signing Firmware and Config Files

Refer to the following steps to sign firmware and Config files:

1. Open the Cygwin64 Terminal or your preferred shell environment and navigate to the directory where the FST is stored, as described in the Generating New Keys section.
2. Move the following three files, listed in Table 4.2, into the same directory as shown in Figure 4.6.

Table 4.2. File Names and their Descriptions

File Names	Descriptions
MACHNX_SOC_impl1_a.jed	• Generated by Diamond software, refer to Sign .jed File with Dummy Key Pair section • Programmed into CFG0 and CFG1 internal Flash sectors of MachNX
MACHNX_SOC_impl1.bit	• Generated by Diamond software, refer to Sign .jed File with Dummy Key Pair section • Programmed into SRAM cells via Diamond Programmer • Not used in the standard workflow
MACHNX_SW.mem	• Generated by Propel software, refer to Lattice Sentry Demo Board for Mach-NX Walkthrough User Guide (FPGA-UG-02167) • Programmed in external SPI Flash

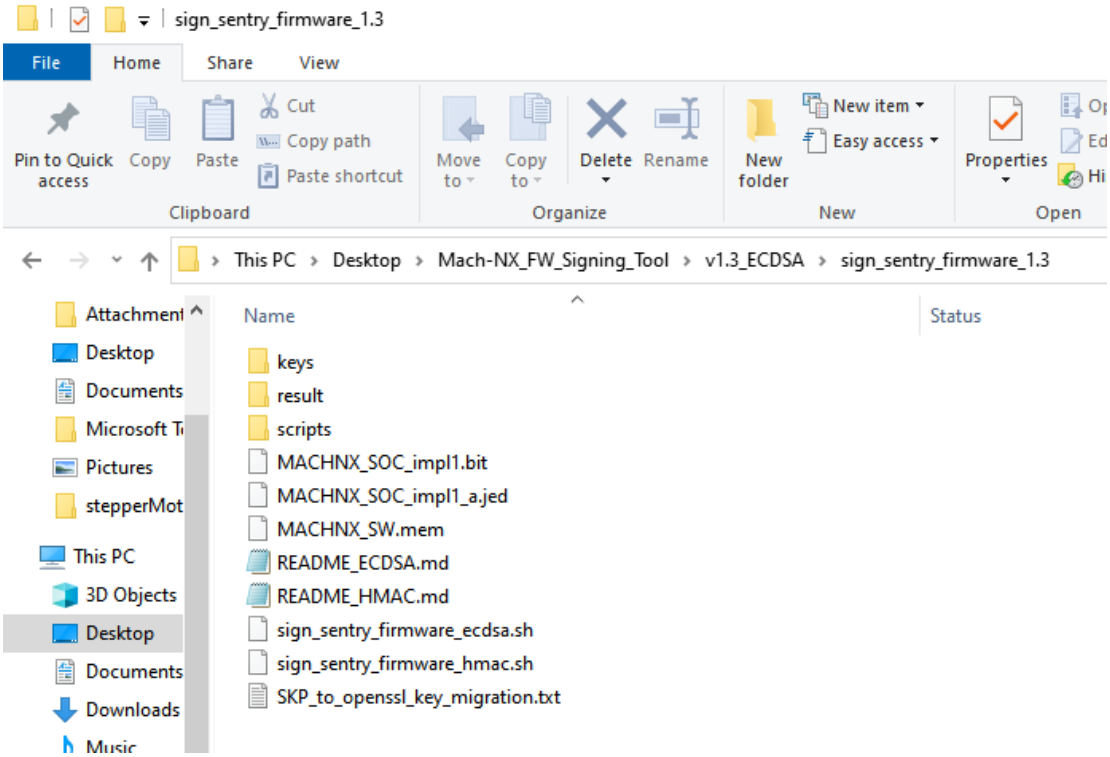

Figure 4.6. Selected Three Files in the Same Directory

3. Enter the following command as shown in Figure 4.7.

Command:

```
./sign_sentry_firmware_ecdsa.sh -s sign -m MACHNX_SW.mem -j MACHNX_SOC_impl1_a.jed -b
MACHNX_SOC_impl1.bit
```

Note: The file names may differ, depending on the project name.

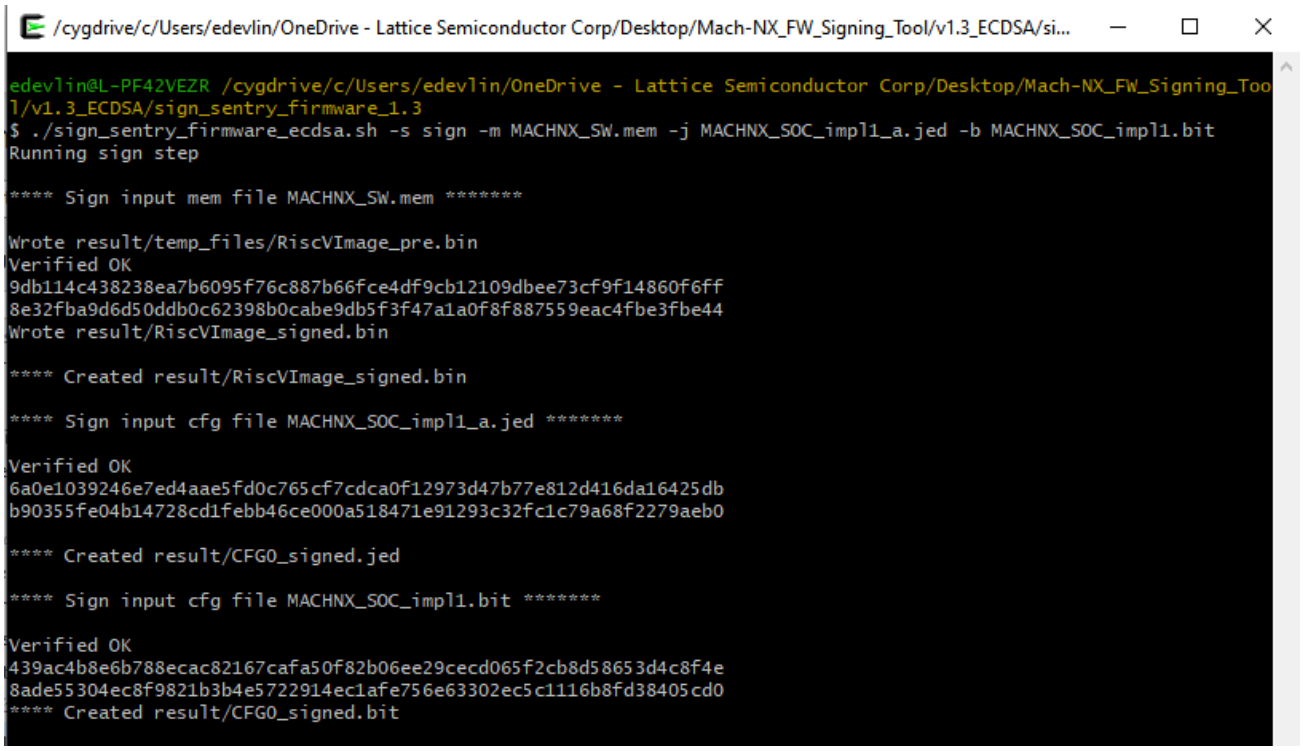

Figure 4.7. Enter the Given Command to the Terminal

4.4. Generated Files from FST

After the scripts have run, an output directory named *result* is automatically generated. The generated files and their descriptions are listed in Table 4.3.

Table 4.3. Generated Files from FST and their Descriptions

File Names	Descriptions
result/temp_files	Temporary files made during the process
result/CFG0_signed.bit	Signed Config File for Mach-NX for SRAM cell programming only via Diamond Programmer (not used in standard workflow)
result/CFG0_signed.jed	Signed Config File for Mach-NX
result/RiscVImage_signed.bin	Signed RISCV image to be programmed into Flash.

5. Generating and Signing a Config Update File

5.1. Config Firmware Update Overview

The Mach-NX device has multiple internal flash sectors that need to be programmed and most likely updated after system deployment in the data center. These internal flash sectors are CFG (configuration bitstream) and UFM (user flash memory).

In the Sentry solution, the CFG bitstream contains the SFB interface IP block in the FPGA logic. One of the SFB interface IP block functions is to process the SFB bitstream and firmware image files from the external flash to complete the boot up process for the Mach-NX PRoT device. The UFM sectors contain manifest information, DICE certificates (if enabled) and FAM information (address locations of the SFB image and FW images on external SPI Flash).

Initially, the CFG0 (primary image) and CFG1 (recovery image) are pre-programmed in the Lattice manufacturing flow or by using the Diamond Programmer tool and JTAG cable. You can use the Lattice Diamond Programmer tool and JTAG cable to program the CFG0 and CFG1 sectors with the .jed programming file. This file is generated by Lattice Diamond tool.

After field deployment of the system, the CFG sector might occasionally require an update. This update will most likely be performed by the system via an orchestration firmware event. The update file will be generated by Lattice Deployment Tool.

The Diamond tool generates .jed and .bit files. The .bit file is targeted to program the SRAM cells in the Mach-NX, not the internal CFG flash sector. This .bit file has a different format than a .bin or .jed file and should not be used to update the CFG flash sector via firmware. The .jed file is approximately 1.6 MB.

The Lattice Diamond Deployment Tool is used to generate a CFG_signed.bin file using the Diamond-generated and FST-signed .jed files as input. This .bin file is approximately 200 KB and therefore more efficient for the firmware to use for an update, compared to the jed file.

The rest of this section covers how to generate, sign, and test the .bin update file for a CFG flash sector update via firmware.

5.2. Diamond Deployment Tool Usage

Refer to the following steps to perform a CFG flash sector update:

1. Generate the .jed file in Diamond software as described in Sign .jed File with Dummy Key Pair section.
2. Sign the .jed file with the key pair, as described in Signing Firmware and Config Files section.
3. Launch the Diamond Deployment Tool from the Windows Start menu, or through Diamond Programmer by clicking the **Design** > **Utilities > Deployment Tool** from the drop-down menu as shown in Figure 5.1.

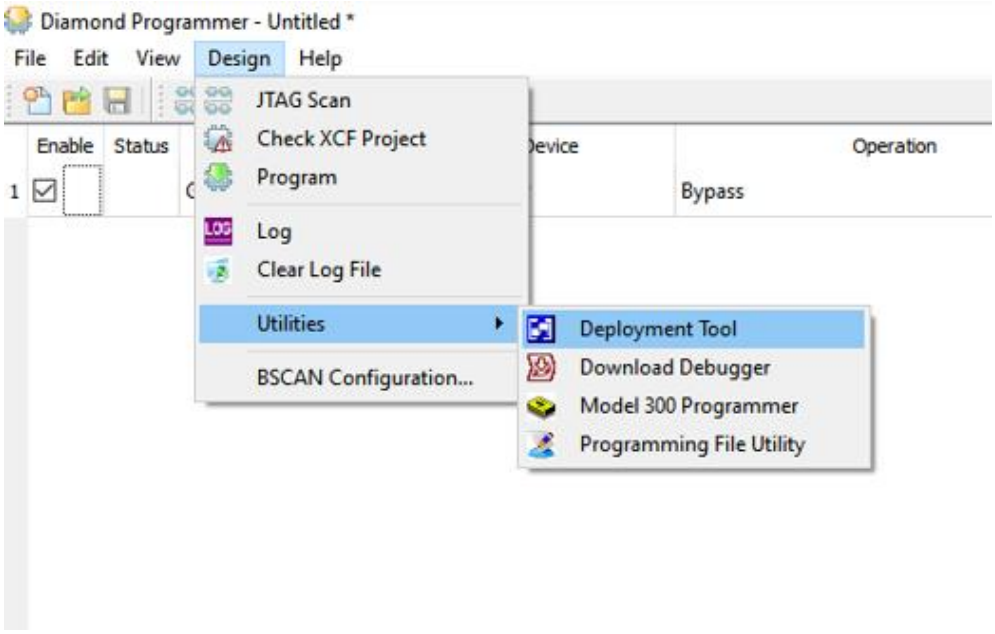

Figure 5.1. Launch the Diamond Deployment Tool through Diamond Programmer

4. Select **File Conversion** as the *Function Type* and **JEDEC to Hex** as the *Output File Type* as shown in Figure 5.2.

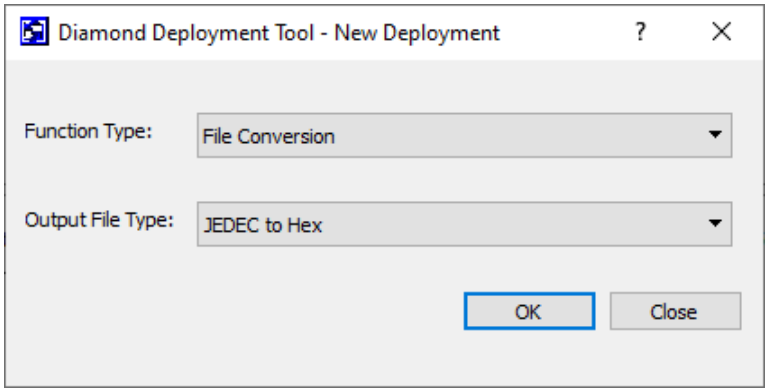

Figure 5.2. Select Function and Output File Types

5. Select the input JEDEC file and click **Next** as shown in Figure 5.3.

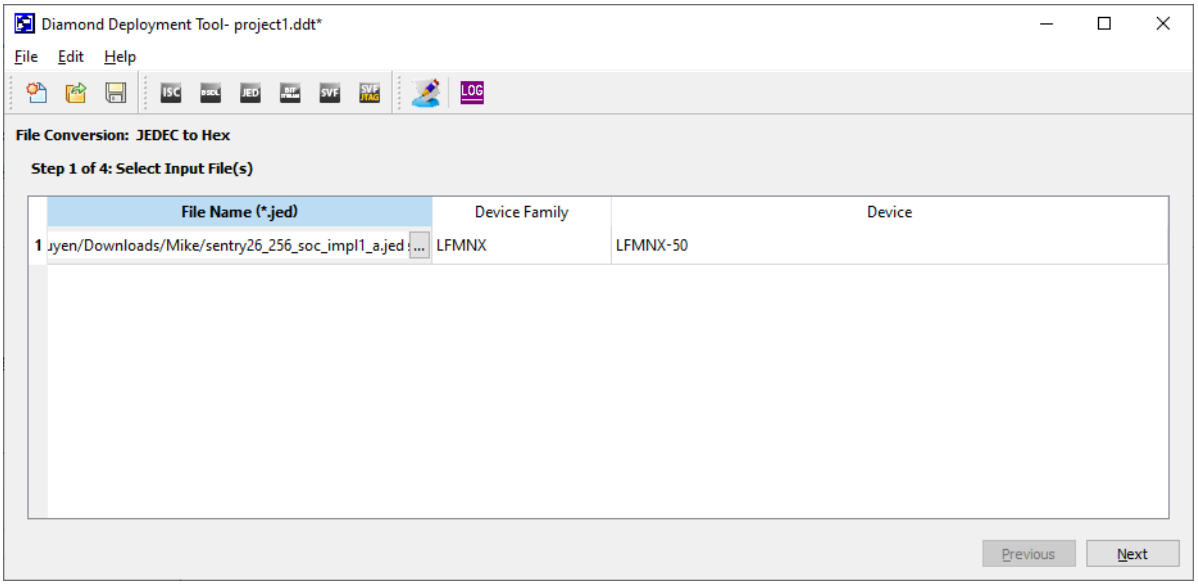

Figure 5.3. Select Input File(s)

6. Select **Binary Raw Hex (*.bin)** as the *Output Format* and click **Next** as shown in Figure 5.4.

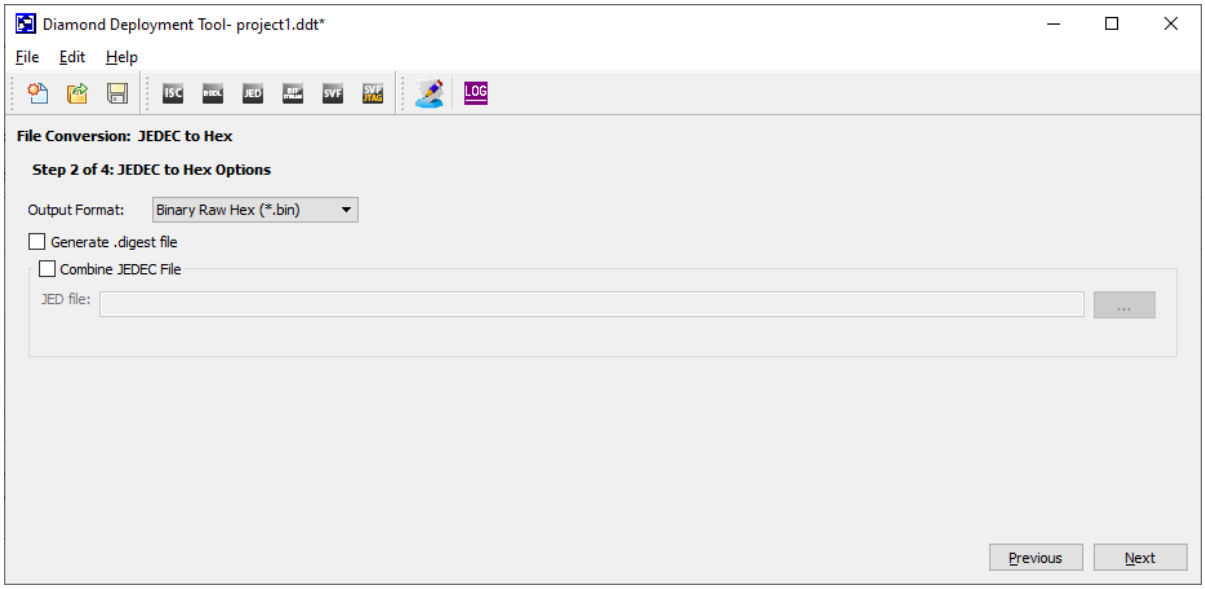

Figure 5.4. Select Output Format

7. Select the output .bin file name and location, then click **Next** as shown in Figure 5.5.

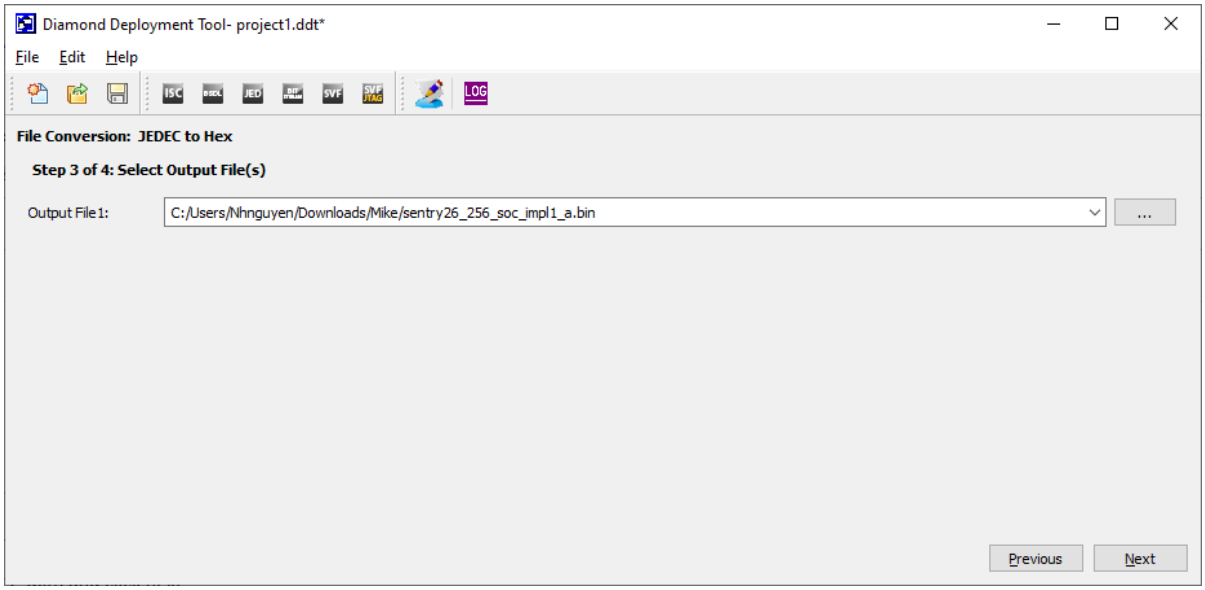

Figure 5.5. Select Output .bin File Name and Location

8. Click the **Generate** button to generate the .bin file as shown in Figure 5.6.

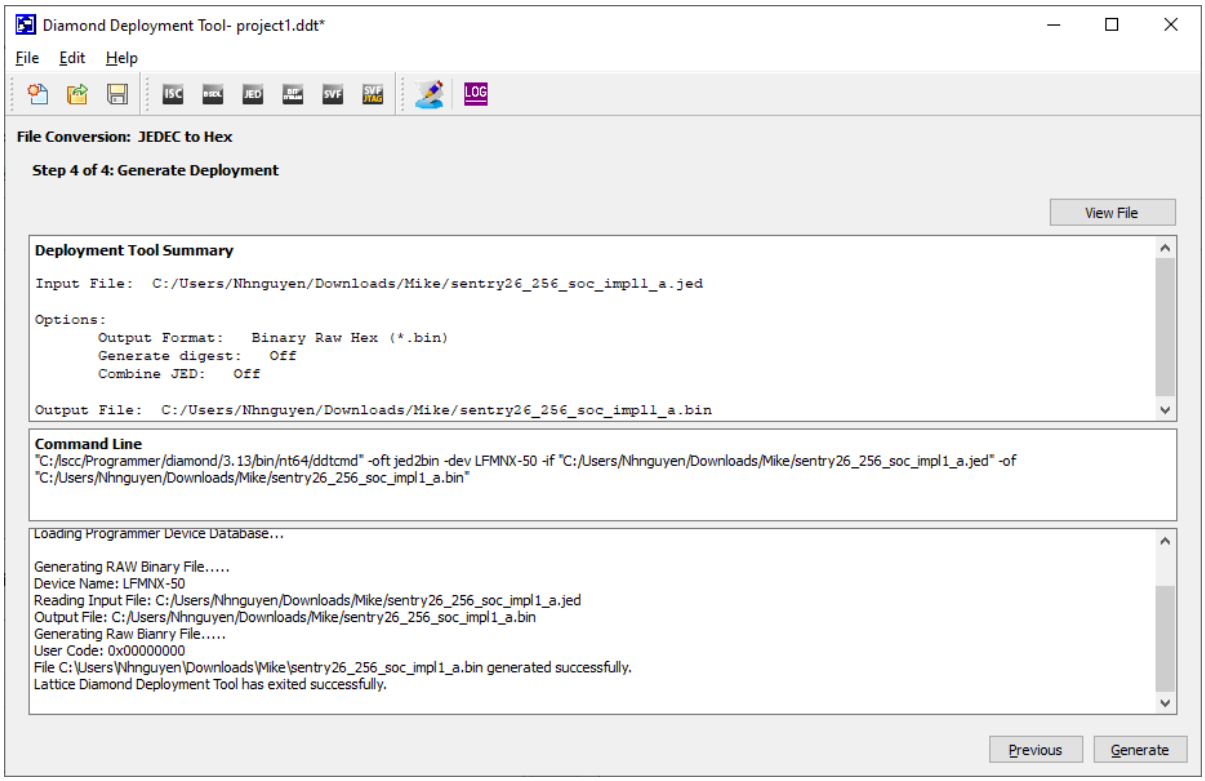

Figure 5.6. Generate the .bin File

9. The .bin file has been created and located in the directory chosen in step 7.

5.3. Programming/Testing the Config Update File

The .bin file generated by the Lattice Diamond Deployment Tool can be loaded via orchestration firmware from system memory, that is the staging area in external SPI flash, to update the Mach-NX CFG internal FLASH sector without physical access to the system. This is different to the .jed file, which is programmed directly into the CFG0/CFG1 sector of the Mach-NX using Lattice Diamond Programmer and a JTAG cable.

A firmware test is used to test this on the Lattice Sentry Demo Board for Mach-NX. The Flash address where the update file is programmed will need to be written in the firmware routine. A sample firmware routine is given in the following sections.

5.3.1. Firmware Functionality

A config update routine needs to be included in the firmware to perform a firmware update. An example firmware config update routine is shown below:

```
void cfg_isp(struct st_pfr_instance *pfr_inst, unsigned int fromAddr, unsigned char is_signed)
{
	unsigned int boot_src;
	unsigned int i, j, k;
	unsigned char QuadSPI = 0;
	unsigned char page_buff[256];
	unsigned char addr4B = 1;
	unsigned char checksum;
	unsigned int pageno = 0;
	unsigned int boot_info[3] = {0};

	QuadSPI = manifest.flash[0].flash_info & FLASH_QSPI;
	printf("\n\rStart of the function");
	print_sr_reg(pfr_inst,2);
	crypto_bootinfo_get(pfr_inst->crypto_inst, boot_info);
	boot_src = boot_info[1];

	DEBUG_PRINTF("boot_src= %d\r\n",boot_src);
	// boot from CFG1
	if(boot_src == 1) {
		uab_ufm_erase(pfr_inst->uab_inst, CFG1);
		print_sr_reg(pfr_inst,3);
		DEBUG_PRINTF("Copying\r\n");
		qspi_mon_select_flash(pfr_inst->spi_monitor_inst, 0, FLASHA_EN,
				MUXSEL_INTMASTER);
		for (i = 0; i < 49; i++) {
			//copy page at a time, 256 byte in page, 16 pages in sector
			for (j = 0; j < 16; j++) {

				if(QuadSPI) {
					qspi_quad_read_rxfifo(pfr_inst->spi_streamer_inst,
fromAddr, 256,
							page_buff, addr4B);
				} else {
					spi_read(pfr_inst->spi_streamer_inst, fromAddr, 256,
							page_buff, addr4B);
				}
				//DEBUG_PRINTF("page_buff=%x\r\n", page_buff[50]);
				for (k = 0; k < 16; k++) {
```

```
                                uab_ufm_page_write(pfr_inst->uab_inst, pageno++, CFG1,
&page_buff[k*16], &checksum);
                        }

                        fromAddr = fromAddr + 256;
                    }
                }
                print_sr_reg(pfr_inst,4);
                uab_done_set(pfr_inst->uab_inst, 1, 0);
                print_sr_reg(pfr_inst,5);
                if(is_signed) {
                    uab_done_set(pfr_inst->uab_inst, 1, 1);   // set auth done bit
                }
                print_sr_reg(pfr_inst,5);
                uab_ufm_erase(pfr_inst->uab_inst, CFG0);
                print_sr_reg(pfr_inst,6);
                DEBUG_PRINTF("Done\r\n");
            } else if(boot_src == 0) {
                // boot from CFG0
                uab_ufm_erase(pfr_inst->uab_inst, CFG0);
                printf("\n\rAfter erasing CFG");
                print_sr_reg(pfr_inst,3);
                DEBUG_PRINTF("Copying\r\n");  //From A OR B
                qspi_mon_select_flash(pfr_inst->spi_monitor_inst, 0, FLASHA_EN,
                                      MUXSEL_INTMASTER);
                for (i = 0; i < 49; i++) {
                    for (j = 0; j < 16; j++) {
                        if(QuadSPI) {
                            qspi_quad_read_rxfifo(pfr_inst->spi_streamer_inst,
fromAddr, 256,
                                    page_buff, addr4B);
                        } else {
                            spi_read(pfr_inst->spi_streamer_inst, fromAddr, 256,
                                     page_buff, addr4B);
                        }
                        //DEBUG_PRINTF("page_buff=%x\r\n", page_buff[50]);
                        for (k = 0; k < 16; k++) {
                            uab_ufm_page_write(pfr_inst->uab_inst, pageno++, CFG0,
&page_buff[k*16], &checksum);
                        }

                        fromAddr = fromAddr + 256;
                    }
                }
                printf("\n\rAfter writing CFG");
                print_sr_reg(pfr_inst,4);
                uab_done_set(pfr_inst->uab_inst, 0, 0);
                printf("\n\rAfter uab done CFG");
                print_sr_reg(pfr_inst,5);
                if(is_signed) {
                    uab_done_set(pfr_inst->uab_inst, 0, 1);
                }
```

```
            printf("\n\rAfter uab auth done CFG");
            print_sr_reg(pfr_inst,6);
            uab_ufm_erase(pfr_inst->uab_inst, CFG1);
            DEBUG_PRINTF("Done\r\n");
        }
}
```

When this routine is called by main, the firmware address is passed as the *fromAddr* input parameter.

```
        cfg_isp(pfr_inst, 0x02000000, 1);
```

Any accessible SPI Flash address area can be used.

After changing the firmware to add a firmware update routine, generate the .mem file and then follow the steps described in the Signing Firmware and Config Files section to sign the .mem file with the FST keys.

5.3.2. Programming the Config Update File

Refer to the steps below to program the config update file:

1. Program the BMC Flash sectors as described in Table 5.1. For more details about programming the Sentry Demo Board for Mach-NX, refer to Lattice Sentry Demo Board for Mach-NX Walkthrough User Guide (FPGA-UG-02167).

Table 5.1. Programming the BMC Flash Sectors

File Names	Descriptions	Flash Addresses
BMC_384_1GB.bit	BMC configuration file included in Sentry release	0x00000000
RiscVImage_signed.bin	RISCV image generated by Propel, then signed by FST	0x03690000
sfb_prod_secured.bit	SFB file included in Sentry release	0x036C0000
config_update.bin	Config .jed file generated by Diamond, signed by FST, then passed through the Diamond Deployment Tool, resulting in a .bin file	0x02000000

2. Program the CFG0, CFG1 (optional), UFM1, UFM2, and FAM sectors of the Mach-NX as described in the Lattice Sentry Demo Board for Mach-NX Walkthrough User Guide (FPGA-UG-02167).
3. Program the PUBKEY sector of the Mach-NX with the ECDSA_pub.pem key generated by the FST as shown in Figure 5.7.

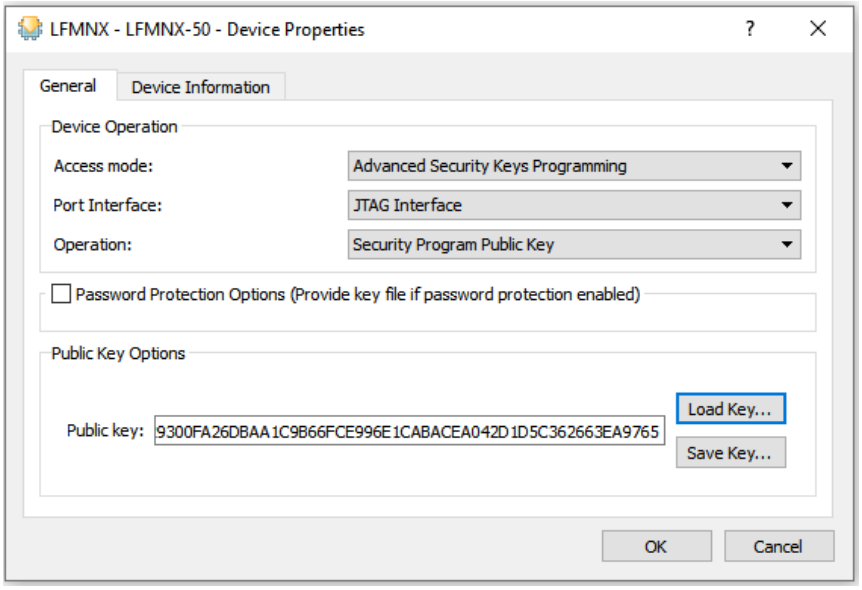

Figure 5.7. Program the Public Key in Diamond Programmer

5.4. Expected Results

After following the steps described in this guide, power cycle or reset the Sentry Demo Board for Mach-NX. The device should boot for the first time from CFG0/CFG1 (depending on settings), not the update configuration. The firmware will then execute the firmware update routine.

After the firmware performs CFGx update routine, reboot the device by power cycling the board or pressing *ProgramN*. The CFG0 or CFG1 (depending on settings) sector will be updated with the newly updated configuration image.

If the board does not reboot, this indicates that there is an error in the firmware update flow. Debugging the flow can be done by checking the CFG0 and CFG1 *Done* and *Auth Done* bits. If the firmware update is successful, the *Auth Done* bit should be *1*.

6. FST Parameters and Usage Guide

6.1. Quick Start Usage (sign_sentry_firmware.sh)

- Full usage guide:
  - Accepts parameters -m <mem file>, -j <jed file> and -b <bit file>
  - Produces result/CFG0_signed.jed, result/CFG0_signed.bit, and result/RiscVImage_signed.bin files
- To generate keys:
  - Command:
    ```
    ../sign_sentry_firmware_ecdsa.sh -s key
    ```
  - Output files: keys/ECDSA_prv.pem and keys/ECDSA_pub.pem
- To sign firmware and FPGA configuration files:
  - Command:
    ```
    ../sign_sentry_firmware_ecdsa.sh -s sign -m file.mem -j file.jed -b file.bit
    ```
  - Parameters accepted:

Table 6.1. Selectable Parameter Options

Parameters	Descriptions
-m	Select input .mem file
-j	Select input .jed file
-b	Select input .bit file

  - Output files: result/CFG0_signed.jed, result/CFG0_signed.bit, and result/RiscVImage_signed.bin

6.2. Cygwin64 Terminal (Run Scripts)

- To run scripts:
  - Command format:
    ```
    ./sign_sentry_firmware_ecdsa.sh -s <step> -m <input_mem_file> -j <input_jed_file> -b <input_bit_file>
    ```
  - Example command:
    ```
    ./sign_sentry_firmware_ecdsa.sh -s sign -m MACH_NX_SW.mem -j MACHNX_SOC_impl1_a.jed -b MACHNX_SOC_impl1.bit
    ```

References

- Lattice Sentry Demo Board for Mach-NX Walkthrough User Guide (FPGA-UG-02167)
- Mach-NX web page
- Lattice Diamond Software web page
- Lattice Propel Design Environment web page
- Lattice Sentry Solutions Stack web page
- Lattice Insights web page for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Revision 1.1, April 2024

Section	Change Summary
All	Renamed the document title from *Lattice Sentry Firmware Signing and Update Guide* to *Lattice Sentry Firmware Signing Tool for Mach-NX Devices User Guide*.
Introduction	Added a note on the use of the FST utility for firmware development and prototyping purposes only.

Revision 1.0, February 2024

Section	Change Summary
All	Initial release.

www.latticesemi.com