



XAPP1333 (v1.1) May 28, 2021

External Secure Storage Using the PUF

Author: Nathan Menhorn

Summary

To store data in non-volatile memory (NVM) using a Zynq® UltraScale+™ device, data must be stored externally and should be encrypted if it is confidential. All Zynq UltraScale+ devices have a built-in physically unclonable function (PUF), which can generate a cryptographically strong, device-unique encryption key that can be used in combination with the built-in advanced encryption standard (AES) cryptographic core. This key cannot be read by a user, allowing for a heightened level of key security. Only if a Zynq UltraScale+ device is provisioned to store the PUF configuration information in eFUSES and if Rivest-Shamir-Adleman (RSA) Authentication is registered and enabled in eFUSES, then the PUF's device-unique encryption key can be used to encrypt and decrypt user data, which can then be stored and read from external non-volatile memory.

Download the [reference design files](#) for this application note from the Xilinx® website. For detailed information about the design files, see [Reference Design](#).

Introduction

The PUF takes advantage of silicon variations unique to Zynq UltraScale+ devices to generate a device-unique encryption key that cannot be read by anyone, including the user. Along with generating a unique encryption key, the PUF also generates the required helper data so that the PUF can exactly regenerate the encryption key later. The details of the PUF are described in the *Zynq UltraScale+ MPSoC: Technical Reference Manual* (UG1085) [Ref 1]. Normally, the PUF's encryption key, referred to as the Key Encryption Key (KEK), is used for encrypting a user's plain-text red key so that a user's red key can be stored encrypted in black key form in either eFUSES or the boot header. The black encryption key is then decrypted using the PUF's KEK to generate the red key, which in turn is used for decrypting the boot information during secure boot. This use of the PUF is shown in [Figure 1](#).

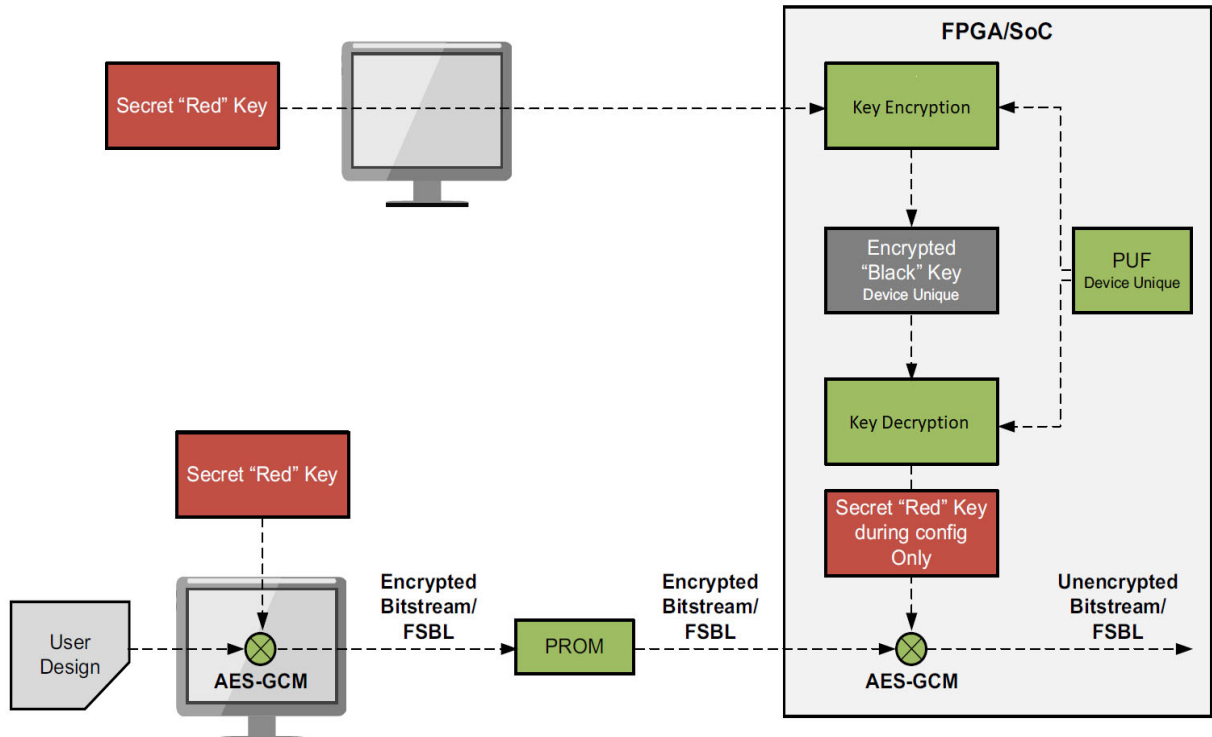


Figure 1: Encrypting and Decrypting the Device Key Using PUF

When the PUF is registered in eFUSES and RSA authentication is enabled in eFUSES, documented in *Programming BBRAM and eFUSES* (XAPP1319) [Ref 2], the PUF's device-unique encryption key can be used to encrypt and decrypt any user data. This encrypted data can then be stored externally to the Zynq UltraScale+ device, which is the focus of this application note. The RSA authentication settings cannot be stored in the boot header when using the PUF to encrypt and decrypt user data.



IMPORTANT: When the `RSA_ENABLE` eFUSES are programmed, boot header authentication is no longer permitted.

The process of using the PUF to encrypt user data is shown in Figure 2 and works as follows: a user generates data that must be encrypted and appends an optional ID. This optional ID can be used to validate that the correct version of data that is being used, such as when the data consists of encryption key information or configuration data, and is useful in preventing replay attacks. Even though the ID is optional, Xilinx highly recommends using it to ensure a more secure system. The optional ID enables key/data revocation as the user data packet can be revoked by burning one of the 256-bit user eFUSES. Each of the 256-bit user eFUSES can be mapped to 256 different 8-bit user IDs. Keep in mind that user eFUSES are a shared resource as the fuses could be used for Enhanced Key Revocation software, a tamper log (see *Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices* (XAPP1323) [Ref 3]), or any other user function.

Next, the PUF is enabled to regenerate the PUF's device-unique encryption key, which is loaded into the AES cryptographic core to encrypt the data. Xilinx recommends minimizing the use of the PUF's key by keeping the user data small or implementing an advanced key-rolling

architecture where the PUF's device-unique key is only used to encrypt the first portion of a larger sized data, thereby minimizing its exposure. This helps to avoid differential power analysis (DPA) attacks. After the encrypted data is written to external memory, the data is read back and decrypted to verify the process using the GCM authentication tag. If the data is authenticated, the user selected ID is safe to use. Conversely, if the data verification fails, a revocation penalty can take place, such as burning an associated user eFUSE.

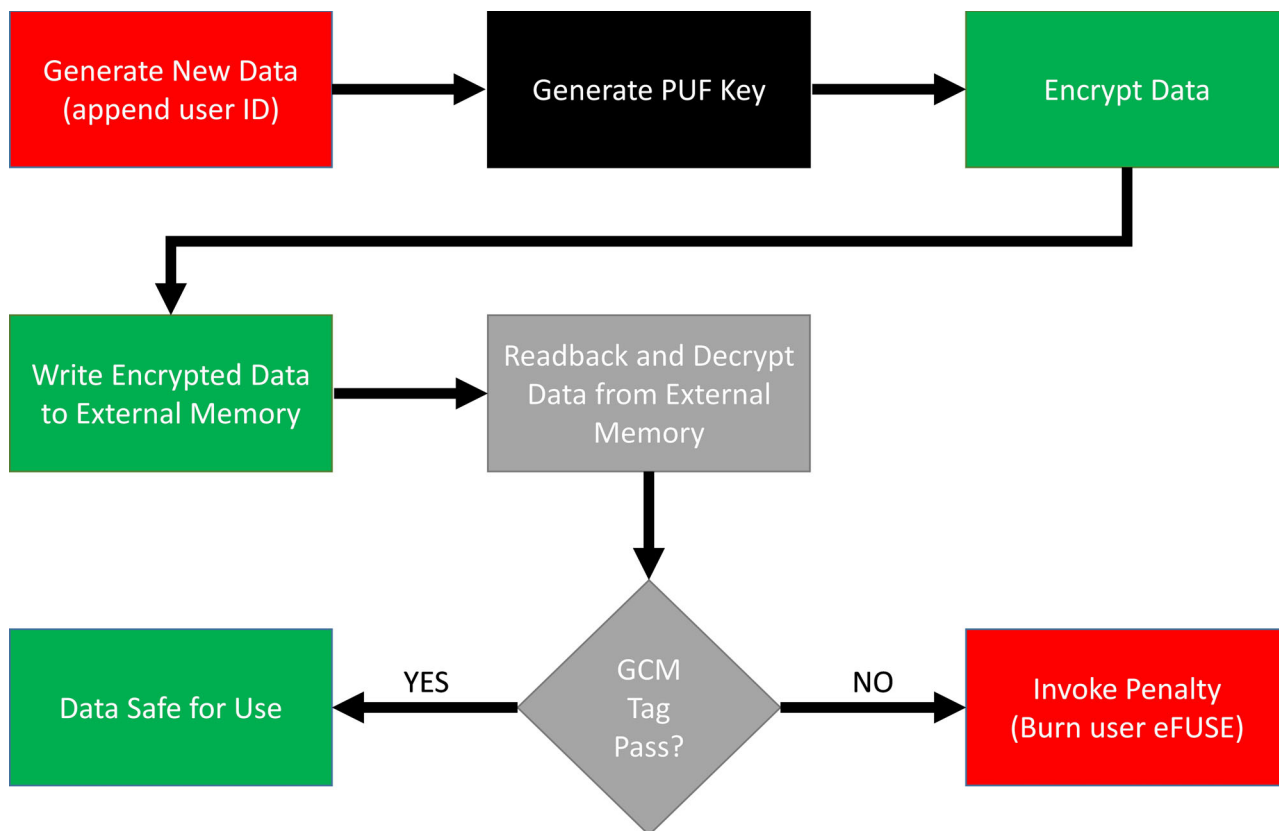


Figure 2: Normal Encryption Process Using PUF

Decrypting external data using the PUF is shown in Figure 3 and works as follows: the encrypted data packet is read from the external memory location followed by regeneration of the PUF decryption key. The data is then decrypted and authenticated via the GCM tag. If authentication passes and if the ID from the decrypted data has not been revoked in user eFUSES, then the data is valid and can be used. Conversely, if the GCM tag authentication fails, then a penalty can be invoked and the decryption process could be stopped to avoid side channel attacks such as DPA. Furthermore, if the decryption process authenticates but the data's ID has been revoked in user eFUSES, the data is invalid and should not be used.



IMPORTANT: *The PUF KEK isn't a FIPS legal key for storing data outside a cryptographic boundary. However, you can create a FIPS-legal KEK, encrypt the FIPS-legal KEK with the PUF KEK, store the encrypted FIPS-legal KEK in eFUSES, and subsequently use the FIPS-legal KEK to store data outside the cryptographic boundary.*

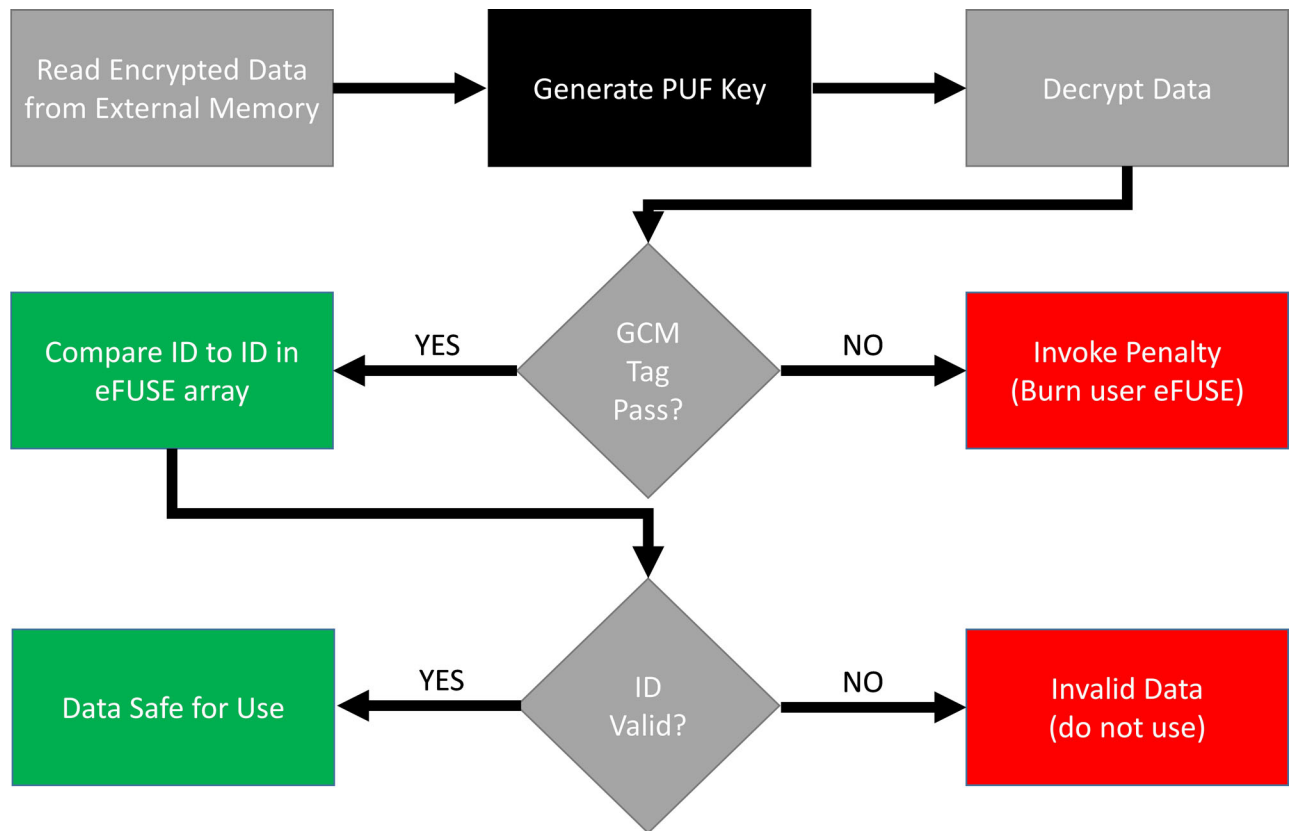


Figure 3: Using the PUF for Decryption

Hardware and Software Requirements

The hardware and software requirements for the reference systems are as follows:

- ZCU102 Evaluation Board
- AC power adapter (12 VDC)
- USB type-A to USB mini-B cable x2
- Optional Platform JTAG hardware and associated cables
- Secure Digital (SD) card formatted using the FAT file system
- Xilinx Software Development Kit (SDK) 2018.1
- Required design files, which can be downloaded [here](#).



IMPORTANT: Programming any of the noted eFUSE settings preclude Xilinx test access. Consequently, Xilinx may not accept return material authorization (RMA) requests.

Create a New Embedded Project for the Zynq UltraScale+ MPSoC

Perform the following steps to create a new embedded project for the Zynq UltraScale+ MPSoC. A brief description is covered in this section. Step-by-step instructions can be found in [Appendix A](#). For detailed elaboration on each step, refer to the *UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [\[Ref 4\]](#) for further details.

1. Open up Vivado® Design Suite and create the hardware design required for the Zynq UltraScale+ ZCU102 Evaluation Board. The PL is not required for this lab so all the PS-PL interfaces are disabled and no bitstream is exported.
2. Export the hardware and launch Xilinx SDK from within the Vivado Design Suite.
3. Create a first stage boot loader project called **FSBL** and its associated Board Support Package named **A53_BSP** running on the ARM Cortex-A53 processor.
4. Build the **FSBL** project by right-clicking on the project and select **Build Project**. This may have already been completed if the **Build Automatically** setting is enabled in Xilinx SDK.
5. Create a **HelloWorld** project to verify the hardware and software setup before proceeding.

Key Generation

Key generation is covered in detail in the *UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [\[Ref 4\]](#) so only a summary pertaining to this application note is documented here.

AES Key Generation

Create a new directory in Xilinx SDK's workspace root directory called `Keys`. The SDK root directory can be found the same level as the `HelloWorld` folder. Generate a device key and its associated IV, an operational key, and one partition block key and its associated IV. Combine these keys and IVs into a file named `multiple_keys.nky`. Alternatively, copy the `Keys` folder found in the reference design documents to use for this lab or, if desired, use them as a template and insert your own key and IV values.

```
Device zcu9eg;
Key 0 0123456789012345678901234567890123456789012345678901234567890123;
IV    01DBD60260A7EC34DE5F6A494;
Key Opt E070C542B6680A855724793A75222391E663CBD35F45D070F22F703A5CA31B45;
Key 1 0000000100000001000000010000000100000001000000010000000100000001;
IV 1 000000010000000100000001;
```

Encrypting the boot image is not required to use the PUF for encrypting user data. However, Xilinx highly recommends doing so, which is used throughout this application note.



IMPORTANT: Be sure to use your own AES keys and associated IVs for operational devices. The keys provided in this lab are for demonstration purposes and are not cryptographically strong.

RSA Asymmetric Key Generation

For this application note, generate a pair of RSA keys called **psk0.pem** and **ssk0.pem**. Alternatively, these keys are provided in the design documents in the `Keys` folder. RSA authentication is required to use the PUF for encrypting and decrypting user data. While this application note does not require the use of a secondary key set, Xilinx highly recommends doing so in an operational application.



IMPORTANT: *Be sure to use your own RSA key pairs in an operational device. The keys provided in this lab are for demonstration purposes and are not cryptographically strong.*

Generate SHA3 of Public RSA Asymmetric Key

Generate the associated SHA3 hash of the RSA PPK and name the output file **sha3.txt**. Alternatively, this hash can be found in the design documents in the `Keys` folder.

PUF eFUSE Configuration



IMPORTANT: *THESE INSTRUCTIONS MODIFY THE EFUSES ON THE ZCU102 DEVELOPMENT BOARD AND MAY LIMIT FUTURE USE OF THE DEVELOPMENT BOARD FOR NON-SECURE TESTING AND DEBUGGING!*



IMPORTANT: *Programming any of the noted eFUSE settings preclude Xilinx test access. Consequently, Xilinx might not accept return material authorization (RMA) requests.*

PUF eFUSE Settings

PUF registration is covered in detail in *Using the PUF* in the *UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [Ref 4] so only a summary pertaining to this application note is documented here.

1. Right-click the **A53_BSP** project and click **Board Support Package Settings**.
2. Click **Overview**.
3. In the **Supported Libraries**, select **xilsecure** and **xilskey**.
4. Click **OK** to close the window.
5. Right-click the **A53_BSP** project and click **Re-generate BSP Sources**.
6. Expand the **A53_BSP** project and double-click the **system.mss** file to open it.
7. Scroll to the bottom of the file and click **Import Examples** for the **xilskey** library.
8. Check the **xilskey_puf_registration** example and click **OK**. This adds the associated project to your workspace.

9. Open the **xilskey_puf_registration.h** file in the **src** folder under **A53_BSP_xilskey_puf_registration_1** in the **Project Explorer** tab.
10. Change the definition of **XSK_PUF_INFO_ON_UART** to **TRUE**. This setting is extremely important to verify the PUF registration completed successfully.
11. Ensure the definition of **XSK_PUF_PROGRAM_EFUSE** is set to **TRUE**.
12. Change the definition of **XSK_PUF_PROGRAM_SECUREBITS** to **TRUE**.
13. Change the definition of **XSK_PUF_SYN_WRLK** to **TRUE**.
14. Set the **XSK_PUF_AES_KEY** to the **Key 0** value in the **aes_key.nky** file.
15. Set the **XSK_PUF_IV** to a value that is user choice. This IV is not related to the IV created in **aes_key.nky** and can be any user generated value. This IV is used by encryption when encrypting the red key with the PUF's KEK.
16. Create a file named **puf_iv.txt** with the ASCII-HEX string of the PUF IV used in **XSK_PUF_IV** as this is needed during boot. Alternatively, use the one provided in the design documents in the **Keys** folder.
17. Verify all the required changes are made before continuing as shown in [Figure 4](#). The **xilskey_puf_registration.h** file with the example keys, shown in [Figure 4](#), is included in the reference design in the **puf_registration** folder.

```

xilskey_puf_registration.h
/***** Macros (Inline Functions) Definitions *****/
/* Following parameters should be configured by user */

#define XSK_PUF_INFO_ON_UART      TRUE
#define XSK_PUF_PROGRAM_EFUSE    TRUE
#define XSK_PUF_IF_CONTRACT_MANUFACTURER  FALSE

/* For programming/reading secure bits of PUF */
#define XSK_PUF_READ_SECUREBITS   FALSE
#define XSK_PUF_PROGRAM_SECUREBITS TRUE

#if (XSK_PUF_PROGRAM_SECUREBITS == TRUE)
#define XSK_PUF_SYN_INVALID       FALSE
#define XSK_PUF_SYN_WRLK         TRUE
#define XSK_PUF_REGISTER_DISABLE FALSE
#define XSK_PUF_RESERVED         FALSE
#endif

#define XSK_PUF_AES_KEY "0123456789012345678901234567890123456789012345678901234567890123"
#define XSK_PUF_IV       "012345678901234567890123"

#define XSK_PUF_REG_MODE    XSK_PUF_MODE4K
                          /**< Registration Mode
                          * XPUF_MODE4K */

```

Figure 4: PUF Registration File Required for eFUSE

PUF Registration into eFUSES

To register the PUF into the eFuse, perform the following steps:

1. Right-click the **A53_BSP** project and click **Board Support Package Settings**.
2. In the **Board Support Package Settings** window, expand the Overview tree, then click **standalone** as shown in [Figure 5](#).

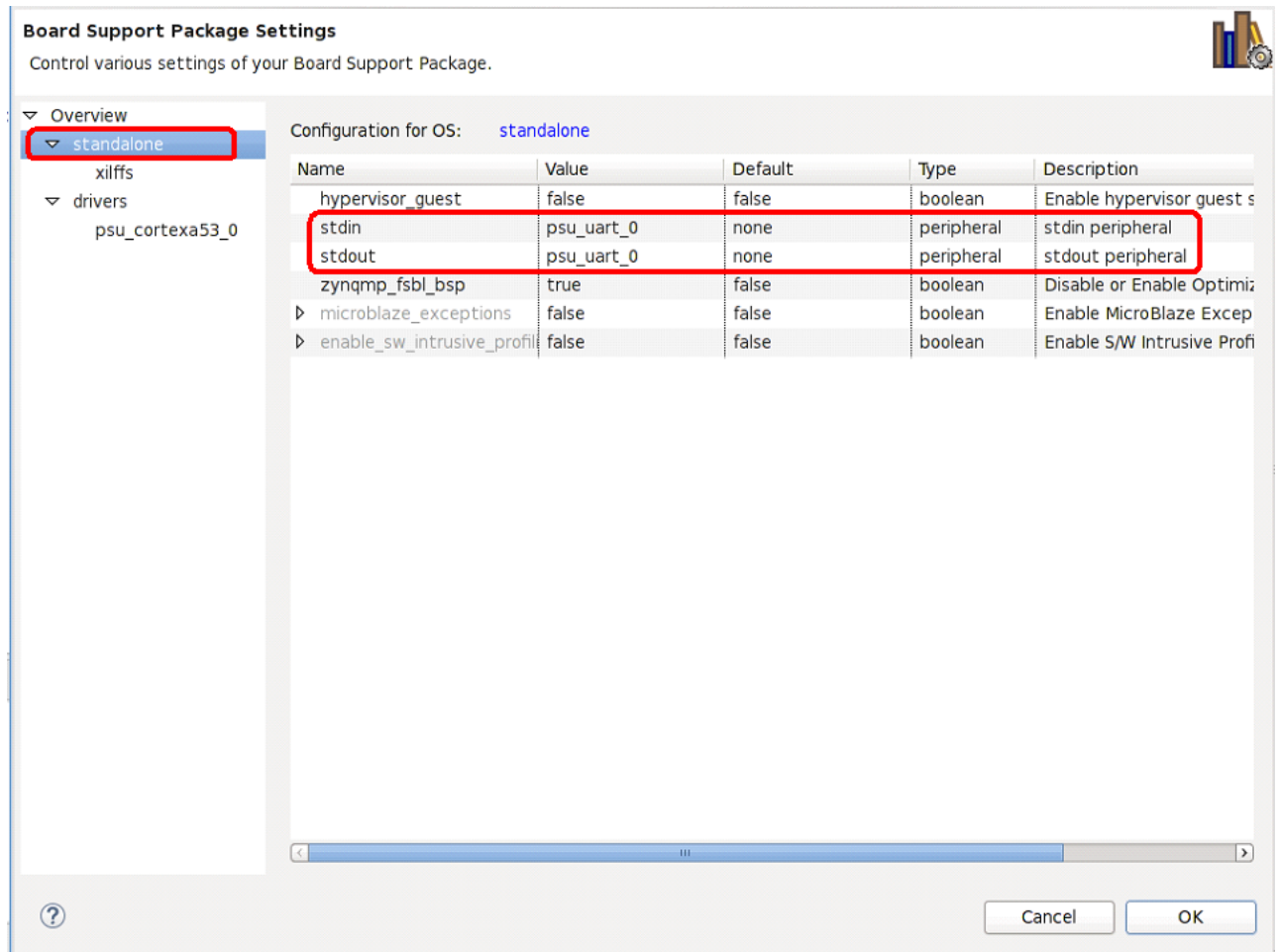


Figure 5: Setting Up the UART Output Using the BSP Settings

3. Ensure the **stdin** and **stdout** functions are mapped to `psu_uart_0` and click **OK**.
4. In Xilinx SDK, right-click **A53_BSP_xilskey_puf_registration_1** and select **Build Project**. This may have already been completed if your SDK environment is set up to build automatically.
5. Turn power off to the ZCU102 board.
6. Connect either the USB JTAG connector J2 to the ZCU102 development board and then a computer or connect the Platform JTAG to the ZCU102 and the associated hardware to a computer.
7. Connect a USB cable from the USB Serial port connector J83 on the ZCU102 board to a computer and note which COM port was enumerated with the Silicon Labs Quad CP2108 USB to UART Bridge: Interface 0.

8. Open a terminal program such as PuTTY or Tera Term and connect to the COM port listed above at 115,200 baud. Enable terminal logging and select a file name and location.
9. On the ZCU102 development board, set the dip switch SW6 to configure the board for JTAG boot mode as shown in [Figure 6](#).

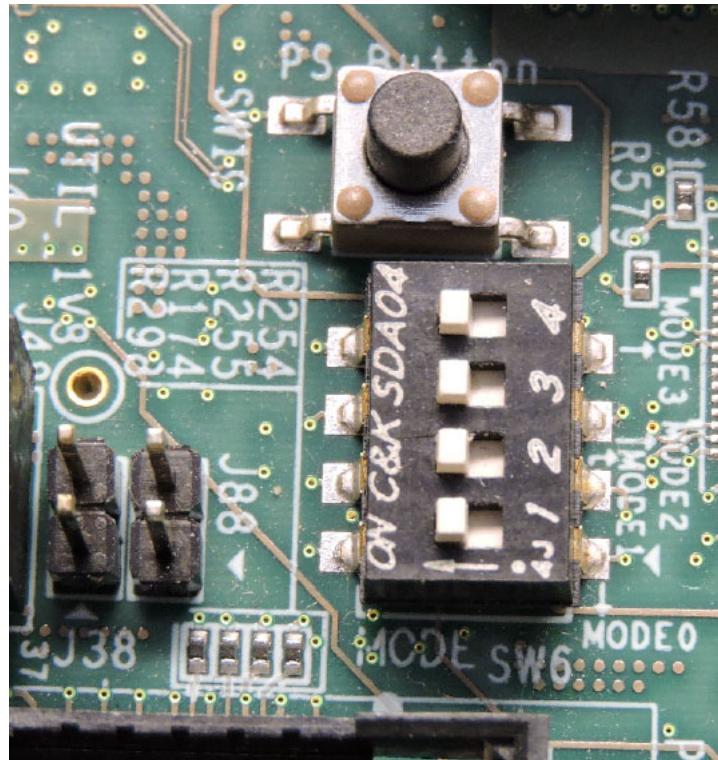


Figure 6: ZCU102 JTAG Boot Mode Switch

10. Power on the ZCU102 board using switch SW1.

- Right-click **A53_BSP_xilskey_puf_registration_1** > **Run As** > **Launch on Hardware (System Debugger)** (System Debugger) as shown in Figure 7.

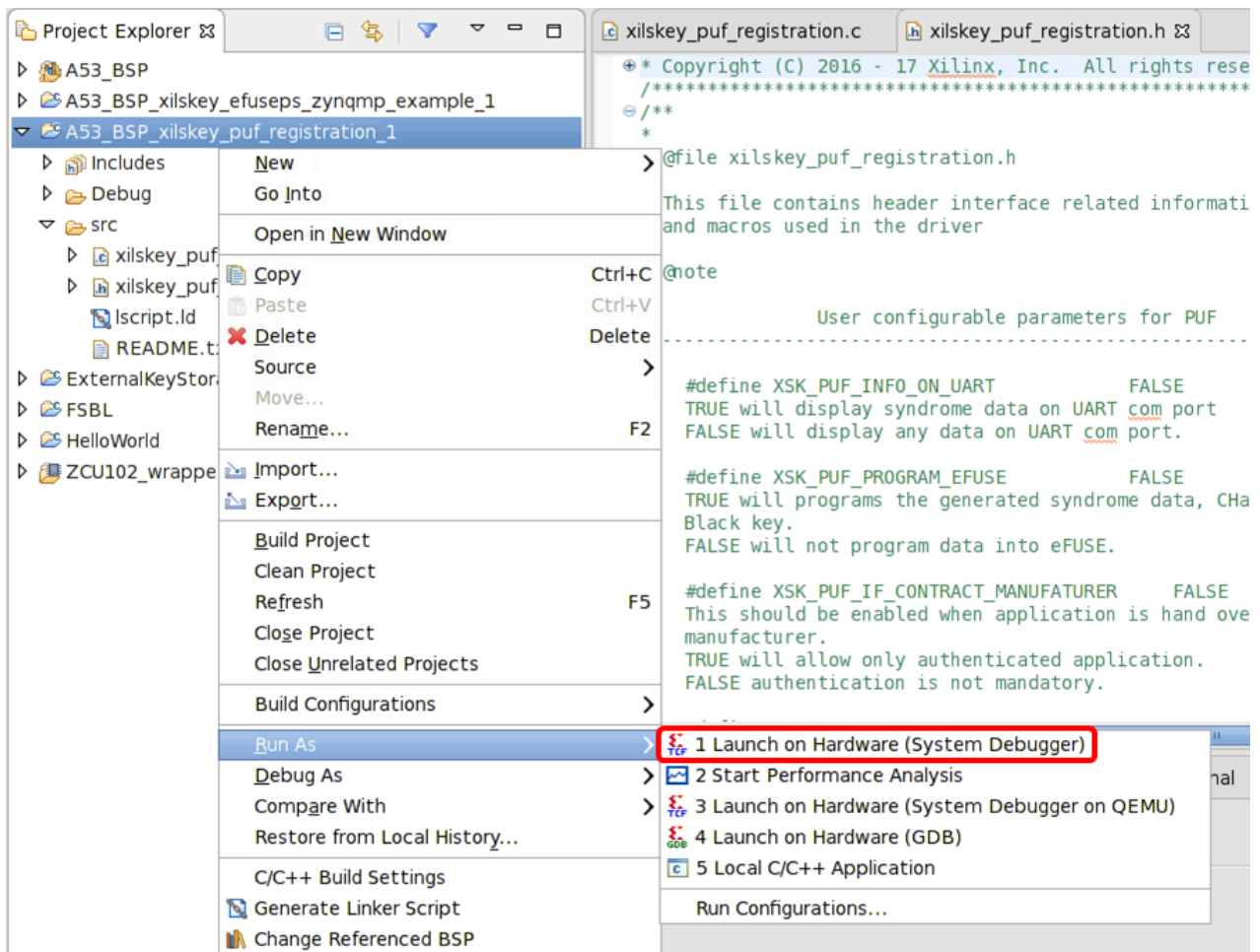


Figure 7: Running the PUF Registration on the ZCU102 Board


```

25 App: Raw Data[16]:02B00000
26 App: Raw Data[17]:02F00000
27 App: Raw Data[18]:02800000
28 App: Raw Data[19]:02A00000
29 App: Raw Data[20]:02A00000
30 App: Raw Data[21]:02900000
31 App: Raw Data[22]:02C00000
32 App: Raw Data[23]:02900000
33 App: Raw Data[24]:02A00000
34 App: Raw Data[25]:02B00000
35 App: Raw Data[26]:02D00000
36 App: Raw Data[27]:02C00000
37 App: Raw Data[28]:02C00000
38 App: Raw Data[29]:02C00000
39 App: Raw Data[30]:02800000
40 App: Raw Data[31]:02800000
41 App: Raw Data[32]:02800000
42 App: Raw Data[33]:02C00000
43 App: Raw Data[34]:02B00000
44 App: Raw Data[35]:00000000
45 App: Debug 2 result end!!
46 App: Red key - 0123456789012345678901234567890123456789012345678901234567890123
47 App: Black key IV - 012345678901234567890123
48 App: DCache disabled
49 App: DMA config
50 App: DMA config initialize
51 App: AES initialize
52 App: AES encryption
53 App: Encrypted key generated
54 App: Black key - 45708B9872F7381850F81A5152EF349E35A7F906FA359D542E4E5ED51F67E82B
55 App: Programming eFUSE
56 App: Formatted syndrome data start!!!
57 0BED582E1B6A265677D4DA923D4ACBB4273967147FD0EC2D9F4FB6A7C8D56AF0693664380AD697D69A6B28
104433DD066F5E452452975CE373F0F835C1E01169E9A073FF767755041AFF2989A9DB98BB426225BCC2D9
F80781F657202E37321E48ACBC51D58D97DAB7215FC003D977A25BBA9FBECB8DE92836958CC60F1C64938
A1BB5F690551FDE3620F7612C253866C9BC12DBE5A4AAF7C82F5BDA0FD93E69BFC9A9ACD2FED6E7A895806
2BFF4A9D90260F8A2887E075A8B8A326D9302AE8CB1EE03DDDE00DE6E89E19AC94AE3F72128C82ABB44E69
43F35035E6235ECAC826D6864DEFA437C96FF9F0FB219801560F421C0FDED8F72FD68AF396CFEC60EC5B5
2180A446865B0E403730F130BE6B4767162ECD739542741E4E2EFD46FBBA687EF5408A61B1D682A08A3A6
BEF3F004A779BE35483D25C8837E72058D2D88D2EEF0A46098D618C1B9F923D02C6054778A45FDA694335C
951D6F32501A1FC2920C4600897F9EFA6664989FA9CA7A575DB70222E7A99AEDDF9646B2A40DB895DE12B1E
05CB81E630C1AC546ADEC0DA6C084029072275612C41338F0BE06CC7252A474C84C20BA504C03B89406D3F
66DFF7B500F2AB70BACB2B2F0E6D272E4A9B6D3A6201C4C6B28E750B89AFE3EA55F807144DE168DD786F09
3A4788F71978FA3A1D43396658AB66FF52CFFBCAF42FE6C231EDE1B4DDE1882D3D300
58 App: Formatted syndrome data End!!!
59 App: Syndrome write successful
60 App: Aux write successful
61 App: CHASH write successful
62 App: Black key writing is passed
63 App: CRC caclulated on black key is = 572287EF
64 Black key CRC check is passed
65 App: Writing eFUSE PUF secure bits
66 Status:00000000

```

Figure 9: Terminal Output Registering PUF to eFUSES - 2

12. Verify line 46 of the UART output is the Red Key that was configured in **XSK_PUF_AES_KEY** in **xilskey_puf_registration.h**.
13. Verify line 47 of the UART output is the Black Key IV that was configured in **XSK_PUF_IV** in **xilskey_puf_registration.h**.

Line 54 of the UART output is the Black Key generated by the AES encryption engine using the PUF as a KEK.

Line 55 shows that the Black Key was burned into eFUSES.

Line 57 of the UART output is the required syndrome data that the PUF uses to regenerate its device-unique encryption key. It is the data that is being programmed into the eFUSES.

Lines 65 shows that the PUF information has been burned into eFUSES.

14. Power off the ZCU102 development board.

RSA eFUSE Configuration



IMPORTANT: THESE INSTRUCTIONS MODIFY THE EFUSES ON THE ZCU102 DEVELOPMENT BOARD AND MIGHT LIMIT FUTURE USE OF THE DEVELOPMENT BOARD FOR NON-SECURE TESTING AND DEBUGGING!



IMPORTANT: Programming any of the noted eFUSE settings preclude Xilinx test access. Consequently, Xilinx might not accept return material authorization (RMA) requests.

RSA eFUSE Settings

RSA eFUSE registration is covered in detail in *Programming eFUSES for AES and RSA Cryptographic Functions* in the *Programming BBRAM and eFUSES* [Ref 2], so only a summary pertaining to this application note is covered here.

1. Expand the **A53_BSP** project and double-click the **system.mss** file.
2. Scroll to the bottom of the file and click Import Examples for the **xilskey** library.
3. Check the **xilskey_efuseps_zynqmp_example** project and click **OK**. This adds the associated project to your workspace.
4. Open the **xilskey_efuseps_zynqmp_input.h** file in the `src` folder under `A53_BSP_xilskey_efuseps_zynqmp_example_1` in the **Project Explorer** tab.
5. Change the definition of **XSK_EFUSEPS_RSA_ENABLE** to **TRUE**. This permanently forces the use of RSA authentication.
6. Change the definition of **XSK_EFUSEPS_PPK0_WR_LOCK** to **TRUE**. This prevents any modifications to the PPK0 hash stored in eFUSES.

The first set of settings are shown in [Figure 10](#).

```

/**
 * Following is the define to select if the user wants to program
 * Secure control bits
 */
#define XSK_EFUSEPS_AES_RD_LOCK          FALSE
#define XSK_EFUSEPS_AES_WR_LOCK          FALSE
#define XSK_EFUSEPS_ENC_ONLY              FALSE
#define XSK_EFUSEPS_BBRAM_DISABLE         FALSE
#define XSK_EFUSEPS_ERR_DISABLE           FALSE
#define XSK_EFUSEPS_JTAG_DISABLE          FALSE
#define XSK_EFUSEPS_DFT_DISABLE           FALSE
#define XSK_EFUSEPS_PROG_GATE_DISABLE     FALSE
#define XSK_EFUSEPS_SECURE_LOCK           FALSE
#define XSK_EFUSEPS_RSA_ENABLE            TRUE
#define XSK_EFUSEPS_PPK0_WR_LOCK          TRUE
#define XSK_EFUSEPS_PPK0_INVLD            FALSE
#define XSK_EFUSEPS_PPK1_WR_LOCK          FALSE
#define XSK_EFUSEPS_PPK1_INVLD            FALSE
#define XSK_EFUSEPS_LBIST_EN              FALSE
#define XSK_EFUSEPS_LPD_SC_EN             FALSE
#define XSK_EFUSEPS_FPD_SC_EN             FALSE
#define XSK_EFUSEPS_PBR_BOOT_ERR          FALSE

```

Figure 10: Settings for RSA Authentication When Using eFUSEs - 1

7. In the next section of the configuration, change the definition of **XSK_EFUSEPS_WRITE_PPK0_HASH** to **TRUE**.
8. Change the definition of **XSK_EFUSEPS_PPK0_HASH** to the value stored in **sha3.txt** that was created by `bootgen` (or copied from the `Keys` directory) from the previous section.

The second set of settings are shown in [Figure 11](#). These settings using the examples keys are included in the design files in the **xilskey_efuseps_zynqmp_input.h** file in the `rsa_registration` folder. The second RSA authentication key (PPK1) is not written for this application note but it can be done by changing the value of `XSK_EFUSEPS_PPK1_WR_LOCK` and `XSK_EFUSEPS_PPK1_HASH`.

```

/**
 * Following is the define to select if the user wants to select AES key,
 * User Fuses, PPK0 Sha3 hash, PPK1 Sha3 hash and SPKID for Zynq MP
 */
/* For writing into eFuse */
#define XSK_EFUSEPS_WRITE_AES_KEY      FALSE
#define XSK_EFUSEPS_WRITE_PPK0_HASH    TRUE
#define XSK_EFUSEPS_WRITE_PPK1_HASH    FALSE
#define XSK_EFUSEPS_WRITE_SPKID        FALSE

#define XSK_EFUSEPS_WRITE_USER0_FUSE   FALSE
#define XSK_EFUSEPS_WRITE_USER1_FUSE   FALSE
#define XSK_EFUSEPS_WRITE_USER2_FUSE   FALSE
#define XSK_EFUSEPS_WRITE_USER3_FUSE   FALSE
#define XSK_EFUSEPS_WRITE_USER4_FUSE   FALSE
#define XSK_EFUSEPS_WRITE_USER5_FUSE   FALSE
#define XSK_EFUSEPS_WRITE_USER6_FUSE   FALSE
#define XSK_EFUSEPS_WRITE_USER7_FUSE   FALSE

/**
 * Following defines should be given in the form of hex string.
 * The length of AES_KEY string must be 64, PPK hash should be 96/64 based on
 * SHA3/SHA2 selection and and for USER_FUSES, SPK ID must be 32.
 */
#define XSK_EFUSEPS_AES_KEY             "0000000000000000000000000000000000000000000000000000000000000000"
#define XSK_EFUSEPS_PPK0_IS_SHA3        TRUE
#define XSK_EFUSEPS_PPK0_HASH           "38B29D670EECD2678105A9C51B203F50A0557B0614B33"

```

Figure 11: Settings for RSA Authentication When Using eFUSES - 2

Programming RSA eFUSES

Program the RSA eFUSES by performing the following steps:

1. Right-click **A53_BSP** project and click **Board Support Package Settings**.
2. In the **Board Support Package Settings** window, expand the **Overview** tree and then click **standalone**, as shown in Figure 5 in step 2 of PUF Registration into eFUSES.
3. Ensure the **stdin** and **stdout** functions are still mapped to `psu_uart_0` and click **OK**.
4. In Xilinx SDK, right-click **A53_BSP_xilskey_efuseps_zynqmp_example_1** and select the **Build Project** option. This may have already been completed if your SDK environment is set up to build automatically.
5. Power off the ZCU102 board.
6. Connect either the USB JTAG connector J2 to the ZCU102 development board and then a computer or connect the Platform JTAG to the ZCU102 and the associated hardware to a computer.
7. Connect a USB cable from the USB Serial port connector J83 on the ZCU102 board to a computer and make note of which COM port was enumerated with the *Silicon Labs Quad CP2108 USB to UART Bridge: Interface 0*.

8. Open a terminal program such as PuTTY or Tera Term and connect to the COM port listed above at 115,200 baud. Enable terminal logging and select a file name and location.
9. On the ZCU102 development board, set the dip switch SW6 to configure the board for JTAG boot mode as shown in [Figure 6](#).
10. Power on the ZCU102 board using switch SW1.
11. Right-click **A53_BSP_xilskey_efuseps_zynqmp_example_1 > Run As > Launch on Hardware (System Debugger)**.
12. The RSA eFUSE application starts running and outputs information to the terminal as shown in [Figure 12](#). An example log of the writing the RSA eFUSEs is included in the design files in the *Logs* folder called **write_rsa_enable_log.log**.
13. Verify line 12 from the output terminal matches the SHA3 output that was generated and stored in the **sha3.txt** file.
14. Notice in line 18 from the terminal output that the "AES CRC check failed." This is because the Black Key was programmed during the PUF registration and the definition **XSK_EFUSE_PS_WRITE_AES_KEY** was set to false in the **xilskey_efuseps_zynqmp_input.h** file.

Line 31 confirms that RSA authentication is enabled and now required for use because this was burned into the eFUSEs.

Line 32 shows that the PPK0 eFUSE has been programmed and the PPK0 SHA3 value cannot be changed.
15. Power off the ZCU102 development board.


```

1 DNA:4000000011497C34C412145keys read from cache
2 User0 Fuse:00000000
3 User1 Fuse:00000000
4 User2 Fuse:00000000
5 User3 Fuse:00000000
6 User4 Fuse:00000000
7 User5 Fuse:00000000
8 User6 Fuse:00000000
9 User7 Fuse:00000000
10
11
12 PPK0:38B29D670EECD2678105A9C51B203F50A0557B0614B3368E688469D31CBC867EE7C38D287E670D2E4
    FEF602D0EA9F7DE
13
14 PPK1:000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    0000000000000000
15
16 Spkid: 00000000
17
18 AES CRC check is failed
19
20 Secure and Control bits of eFuse:
21
22 AES key CRC check is enabled
23 Programming AES key is enabled
24 All boots must be encrypted with eFuseAES key is disabled
25 BBRAM key is not disabled
26 Error output from PMU is enabled
27 Jtag is enabled
28 DFT is enabled
29 PROG_GATE feature is enabled
30 Reboot from JTAG is enabled
31 RSA authentication is enabled
32 Locks writing to PPK0 eFuse
33 Revoking PPK0 is disabled
34 writing to PPK1 efuses is not locked
35 Revoking PPK1 is disabled
36 LBIIS is in disabled state
37 PBR boot error halt is disabled
38 Zeroization of registers in Low Power Domain (LPD) during boot is disabled
39 Zeroization of registers in Full Power Domain (FPD) during boot is disabled
40
41 User control bits of eFuse:
42 Programming USER_0 fuses is enabled
43 Programming USER_1 fuses is enabled
44 Programming USER_2 fuses is enabled
45 Programming USER_3 fuses is enabled
46 Programming USER_4 fuses is enabled
47 Programming USER_5 fuses is enabled
48 Programming USER_6 fuses is enabled
49 Programming USER_7 fuses is enabled
50 Reserved 1 bits are not programmed on eFUSE
51 Reserved 2 bits are programmed on eFUSE
    
```

Figure 12: Terminal Output While Writing the RSA Settings to eFUSES.

PUF Encryption and Decryption

PUF Encryption Decryption Demo Application

The PUF can now be used for encrypting and decrypting user data because the ZCU102 development board has been provisioned. Specifically, this section uses a reference design to show how to encrypt and decrypt user generated AES keys that are stored on an SD card.



IMPORTANT: Although the user generated encryption keys are being written to an external SD card, any type of user data can be encrypted/decrypted and written/read to any type of external non-volatile memory accessible by the Zynq UltraScale+ device.

1. To write to the SD card, modifications of the **xparameters.h** file located in **A53_BSP > psu_cortexa53_0 > include** is necessary. Comment out line 1406, `#define FILE_SYSTEM_READ_ONLY`, shown in [Figure 13](#). This modification is required to make the SD card writable and readable.

```
/* Xilinx FAT File System Library (XilFFs) User Settings */
#define FILE_SYSTEM_INTERFACE_SD
//#define FILE_SYSTEM_READ_ONLY
#define FILE_SYSTEM_NUM_LOGIC_VOL 2
#define FILE_SYSTEM_USE_STRFUNC 0
#define FILE_SYSTEM_SET_FS_RPATH 0
#define FILE_SYSTEM_WORD_ACCESS
#define XPAR_XILPM_ENABLED
```

Figure 13: Modifying xparameters.h File to Make the SD Readable and Writable

2. In Xilinx SDK, click **File > New > Application Project**.
3. Type in **ExternalKeyStorage** in the Project name:
4. Select **Use default location**.
5. Leave the OS Platform as **standalone**.
6. Leave the Hardware Platform to the name of the hardware exported from Vivado Design Suite.
7. Leave the Processor as **psu_cortexa53_0**.
8. Leave the Language selected to **C**.
9. Leave the Compiler set at **64-bit**.
10. Leave the Hypervisor Guest as **No**.
11. Change the Board Support Package to **Use existing: A53_BSP**.

These settings are shown in [Figure 14](#).

12. Select **Next**.
13. Select **Empty Application**.
14. Click **Finish**.
15. Expand the **src** folder in **ExternalKeyStorage** of the Project explorer window.
16. Right-click **src** and select **Import**.
17. Expand **General**, select **File System**, and click **Next**.
18. Navigate to the `ExternalKeyStorage/src` folder in the reference design file directory and check all ".c" and ".h" files and then click **Finish** as shown in [Figure 15](#).

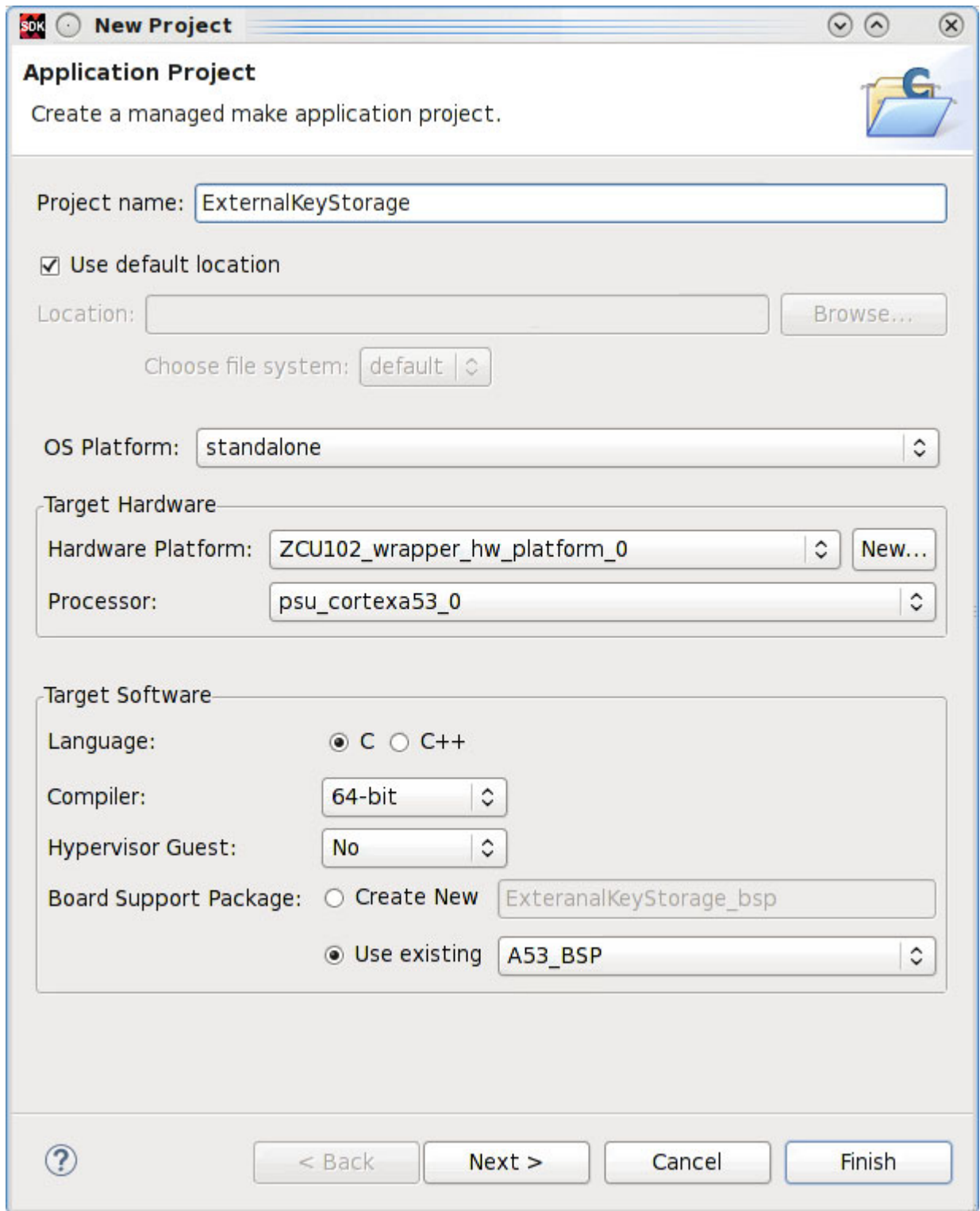


Figure 14: Creating the ExternalKeyStorage Project in Xilinx SDK

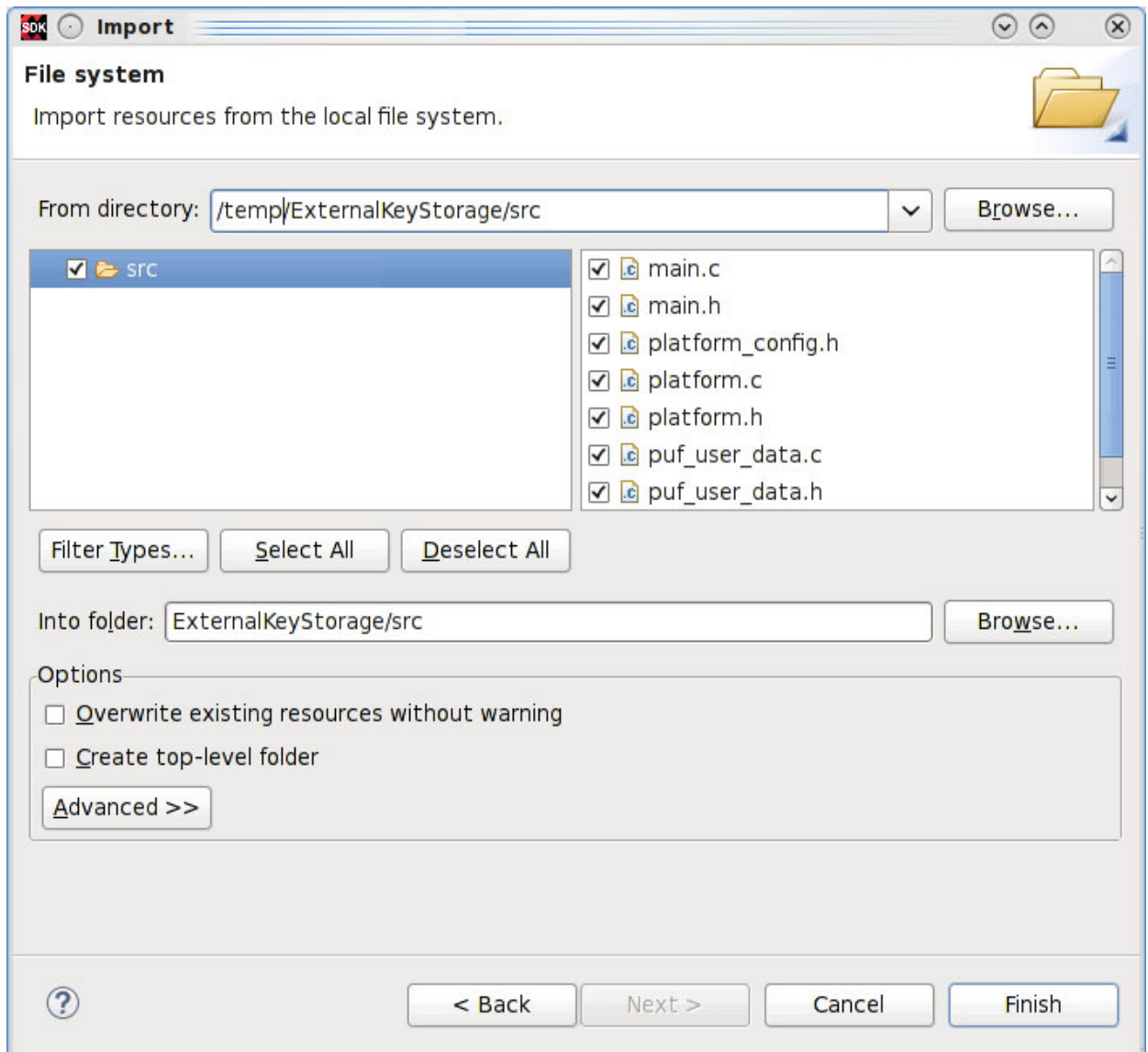


Figure 15: Importing Files from the Reference Design into the ExternalKeyStorage Project

19. Create a new file called **ExternalKeyStorage.bif** in the ExternalKeyStorage folder. This file is also included with the design files and can be copied into the project folder but the paths must be updated to point to the correct folders. Manual creation of the BIF file is necessary to use the Black Key during boot as the Create Boot Image tool within Xilinx SDK does not currently support this feature. Future revisions of Xilinx SDK may support this feature.
20. Update the contents of the file to the contents shown in Figure 16 using the correct paths.

```
1 //arch = zynqmp; split = false; format = BIN
2 the_ROM_image:
3 {
4   [pskfile] ../Keys/psk0.pem
5   [sskfile] ../Keys/ssk0.pem
6   [auth_params] spk_id = 0; ppk_select = 0
7   [aeskeyfile] ../Keys/multiple_keys.nky
8   [keysrc_encryption] efuse_blk_key
9   [bh_key_iv] ../Keys/puf_iv.txt
10  [fsbl_config] puf4kmode, shutter = 0x0100005E, opt_key
11  [bootloader, destination_cpu=a53-0, encryption = aes, authentication = rsa]
12    ./FSBL/Debug/FSBL.elf
13  [encryption = aes, authentication = rsa, destination_cpu = a53-0]
14    ./Debug/ExternalKeyStorage.elf
15 }
```

Figure 16: ExternalKeyStorage.bif File

21. Build the ExternalKeyStorage project in Xilinx SDK.
22. From the command prompt in the ExternalKeyStorage folder run the following command:

```
bootgen -p zcu9eg -arch zynqmp -image ExternalKeyStorage.bif -w -o BOOT.bin
```
23. Power off the ZCU102 board.
24. Copy **BOOT.bin** to a blank SD card.
25. Load the SD card into the J100 SD slot on the ZCU102 development board.
26. Connect a USB cable from the USB Serial port J83 on the ZCU102 board to a computer and make note of which COM port was enumerated with the *Silicon Labs Quad CP2108 USB to UART Bridge: Interface 0*.
27. Open a terminal program such as PuTTY or Tera Term and connect to the COM port listed above at 115,200 baud. Enable terminal logging and select a file name and location.

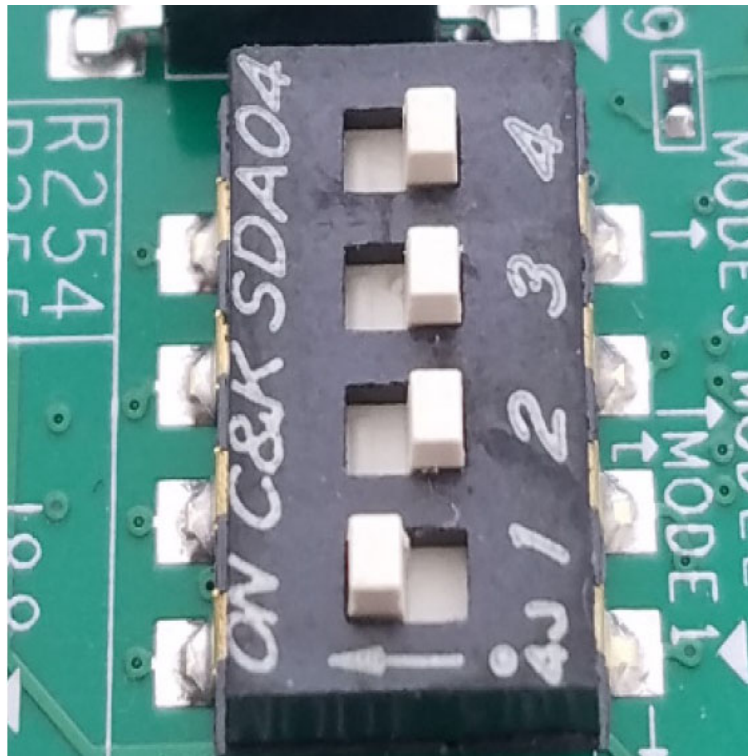
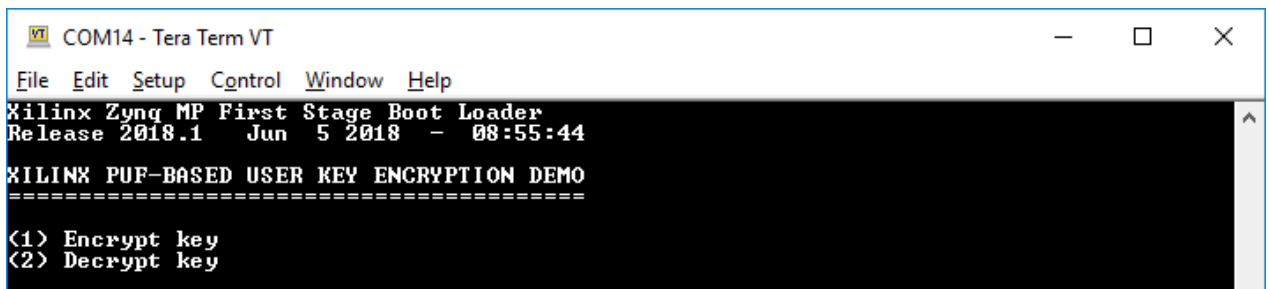


Figure 17: ZCU102 SD Boot Mode Switch Setting

28. On the ZCU102 development board, set the dip switch SW6 to configure the board for SD boot mode as shown in [Figure 17](#).
29. Load the SD card into the J100 SD slot on the ZCU102 development board.
30. Power on the ZCU102 board using switch SW1.

In the terminal program, a menu appears as shown in [Figure 18](#).



```

COM14 - Tera Term VT
File Edit Setup Control Window Help
Xilinx Zynq MP First Stage Boot Loader
Release 2018.1 Jun 5 2018 - 08:55:44
XILINX PUF-BASED USER KEY ENCRYPTION DEMO
=====
<1> Encrypt key
<2> Decrypt key

```

Figure 18: Main Menu of External Key Storage Demo

31. Press **1** to encrypt a user key and to save the encrypted key to the external SD card and follow the prompts, as illustrated in [Figure 19](#).
 - a. Enter a 96-bit IV.
 - b. Enter an 8-bit key ID. Use an ID of 42 for this key.

An ID of 0 is mapped to user eFUSE 0 bit 0, an ID of 1 is mapped to user eFUSE 0 bit 1, ... , an ID of 255 is mapped to user eFUSE 7 bit 31.

- c. Enter a 256-bit AES key.
 - d. Enter a file name including a file extension (for example, Key1.key) for the key up to 16 characters long and then press enter when complete.
32. After entering the file name, the program displays the unencrypted key blob which consists of the IV, Key's ID, and the key itself. Afterwards, the ID and AES key are encrypted using the PUF's device-unique KEK, the entire 61 byte encrypted key blob is displayed, and the entire encrypted key blob is written to the SD card.
 33. Repeat the entire encryption process and encrypt another key and IV, using [step 31](#). However, select an ID that is equal to 0xFF and create a unique key file name (e.g., Key2.key).
 34. Power off the ZCU102 board.
 35. Remove the SD card and insert the card into a SD card reader on a computer.
 36. Using a browser or the command line, display the contents of the SD card.
 37. Make sure both key files generated in [step 31](#) and [step 33](#) appear on the SD card as shown in [Figure 20](#).

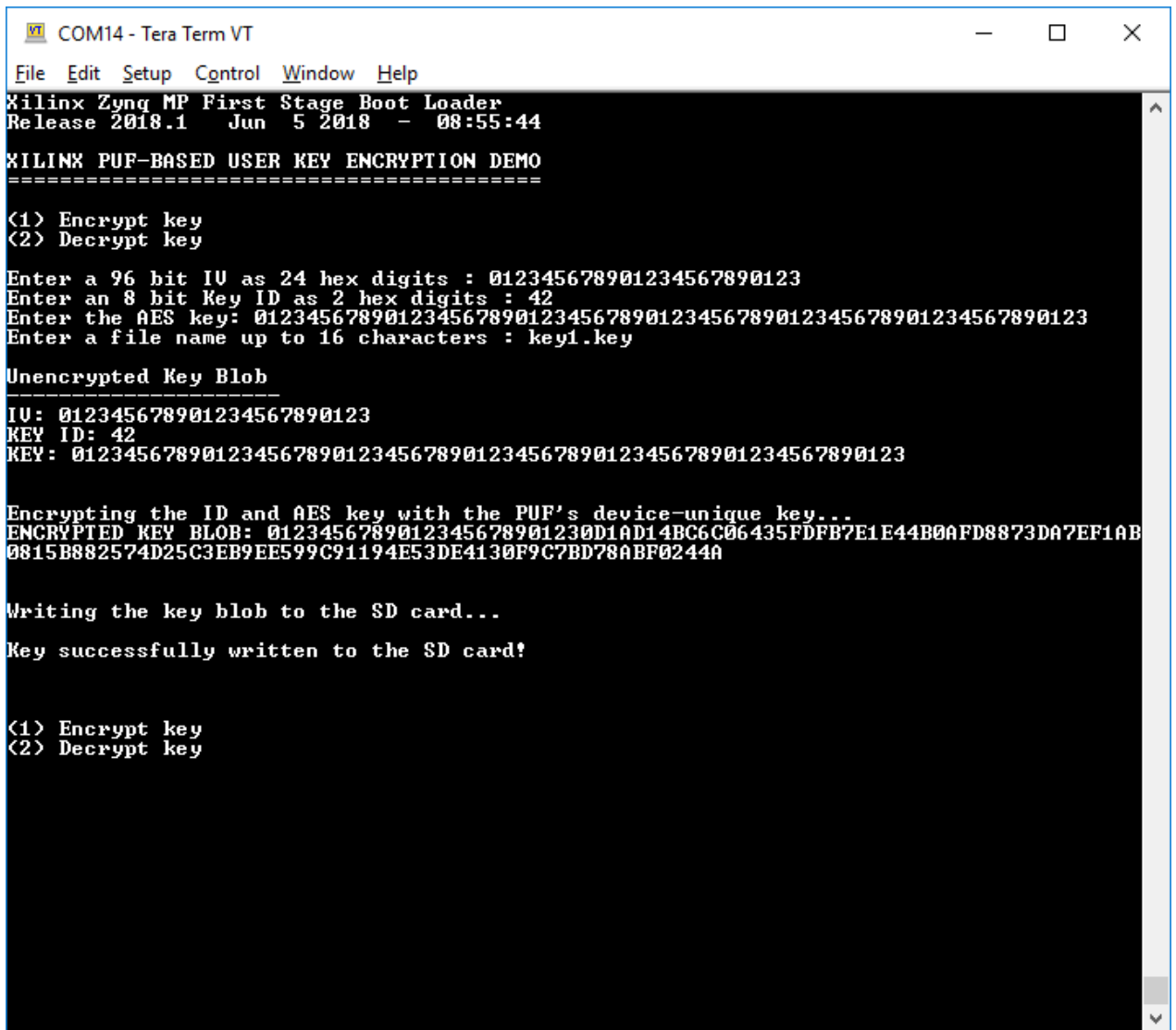


Figure 19: External Key Storage Encryption

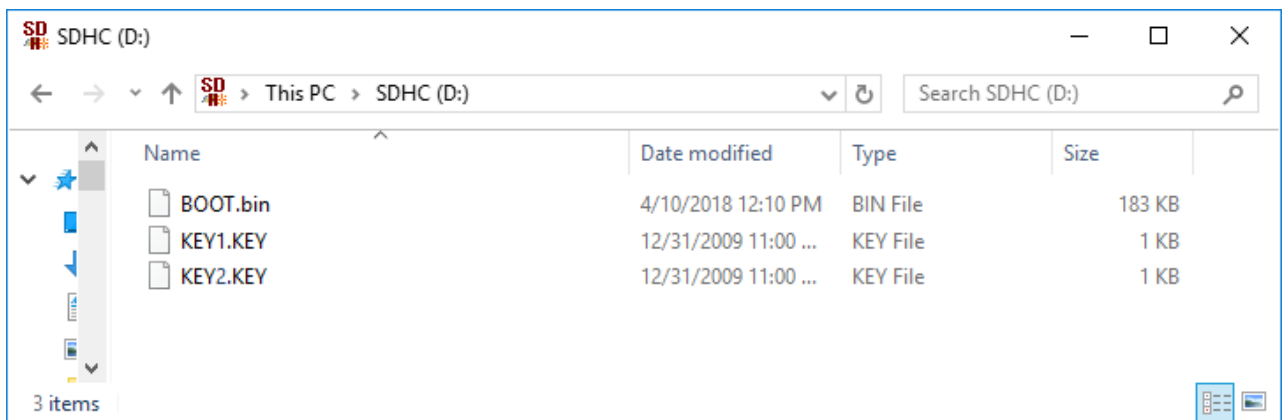
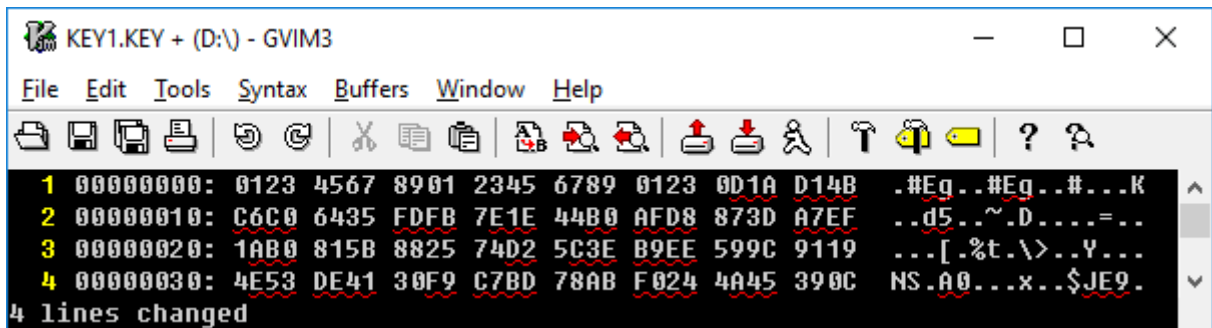


Figure 20: Directory Contents of the SD Card after Writing the Encrypted Key

- Open both keys in a hex editor and confirm that they match the encrypted key blobs displayed in the user application. KEY1.KEY is shown in [Figure 21](#) and matches the output generated in [Figure 19](#).



```

KEY1.KEY + (D:\) - GVIM3
File Edit Tools Syntax Buffers Window Help
1 00000000: 0123 4567 8901 2345 6789 0123 0D1A D14B .#Eg..#Eg..#...K
2 00000010: C6C0 6435 FDFB 7E1E 44B0 AFD8 873D A7EF ..d5..~.D....=..
3 00000020: 1AB0 815B 8825 74D2 5C3E B9EE 599C 9119 ...[. %t.\>..Y...
4 00000030: 4E53 DE41 30F9 C7BD 78AB F024 4A45 390C NS.A0...x..$JE9.
4 lines changed

```

Figure 21: Encrypted Key Data Stored in KEY1.KEY Read from the SD Card

39. Remove the SD card from the computer and insert the card into the ZCU102 development board.
40. Apply power to the ZCU102 development board.

The menu shown in [Figure 18](#) appears.

41. Press **2** to decrypt the data that is stored externally on the SD card.
42. Type in the name of the key and the file extension used in [step 31](#) (Key1.key).

The key is read from the SD card and placed into OCM for processing.

The encrypted key blob is displayed.

The decryption process of the key blob takes place and the decrypted information is displayed showing the IV, key ID, and key.

The decrypted GCM tag is compared to the GCM tag stored in the encrypted key blob and the software indicates if they match.

Lastly, the key ID is mapped to and compared to the associated bit stored in the user eFUSEs and the software indicates if the IDs match. In this case, the IDs match. An ID of 0 is mapped to user eFUSE 0 bit 0, an ID of 1 is mapped to user eFUSE 0 bit 1, ... , an ID of 255 is mapped to user eFUSE 7 bit 31.

43. Repeat the process and decrypt the second key that was created in [step 33](#).
44. All of the same information from [step 42](#) is displayed and the key is decrypted and passes authentication. However, the software simulates ID 255 being revoked and should not be used. When ID 255 is read from a decrypted key file, the software replaces the actual value read in from User eFUSE Seven, 0x0000,0000, with a simulated value of 0x8000,0000. Since bit 31 of User eFUSE Seven is now set and appears to be burned, this simulates ID 255 as being revoked. Refer to [Figure 3](#). Decrypting the two test keys is shown in [Figure 22](#).

Conclusion

This application note guides a user on how to use the PUF's device-unique encryption key in conjunction with the AES-GCM hardware in order to encrypt user generated data and store the encrypted data externally. The encrypted data can then be read from external storage and decrypted using the AES-GCM hardware in conjunction with the PUF's device-unique key. In addition, this application note shows how to perform data validation of decrypted data packets by utilizing values stored in the user programmable section of eFUSEs.

Reference Design

Download the [reference design files](#) for this application note from the Xilinx website.

[Table 1](#) shows the reference design matrix.

Table 1: Reference Design Checklist

Parameter	Description
General	
Developer Name(s)	Jim Wesselkamper, Nathan Menhorn
Target Devices	Zynq UltraScale+ devices
Source code provided?	Yes
Source code format (if provided)	C
Design uses code or IP from existing reference design, application note, 3rd party or Vivado software? If yes, list.	
Simulation	
Functional simulation performed	No
Timing simulation performed?	No
Testbench provided for functional and timing simulation?	No
Testbench format	N/A
Simulator software and version	N/A
SPICE/IBIS simulations	N/A
Implementation software tool(s) and version	SDK 2018.1
Static timing analysis performed?	No
Hardware Verification	
Hardware verified?	Yes
Platform used for verification	ZCU102 evaluation board

Appendix A

Creating the Zynq UltraScale+ ZCU102 Evaluation Board Hardware Design

1. Open Vivado Design Suite.
2. In the **Quick Start** tab click **Create Project**.
3. Click **Next** in the **Create a New Vivado Project** page.
4. Enter **ZCU102** in the Project name.
5. Enter or select an appropriate working directory in the Project location.
6. Click **Next** on the **Project Name** page.
7. In **Project Type**, select **RTL Project** and uncheck **Do not specify sources at this time**.
8. Click **Next** on the **Project Type** page.
9. Click **Next** on the **Add Sources** page.
10. Click **Next** on the **Add Constraints (optional)** page.
11. On the Default Part page, click the **Boards** tab.
12. Type in **ZCU** in the Search.
13. Click the **Zynq UltraScale+ ZCU102 Evaluation Board**.
14. Click **Next** on the **Default Part** page.
15. Click **Finish** on the **New Project Summary** Page and wait while the project is being created.
16. In the Project Manager tab located on the left of the Vivado workspace, click **IP INTEGRATOR > Create Block Design**.
17. When the **Create Block Design** window appears, type in **ZCU102** in **Design**. Leave everything else set to default.
18. Click **OK** and wait while the design is created.
19. In the **Diagram** section of the workspace, located on the top right, click the + button to add IP.
20. When the Search box appears, type in **ZYNQ**.
21. Double-click **Zynq UltraScale+ MPSoC** and wait while the part is added to the design.
22. Click **Run Block Automation** at the top of the **Diagram** window.
23. After the **Run Block Automation** window appears, select **All Automation** and **Apply Board Preset**, click **OK** and wait while the automation takes place.
24. Double-click the Zynq UltraScale+ part in the Diagram window.
25. Click **Page Navigator > PS-PL Configuration** located on the left of the Zynq UltraScale+ (3.2) window.

26. Click **PS-PL Interfaces** located in the PS-PL Configuration window.
27. Click **Master Interface** and uncheck the **AXI HPM0 FPD** and **AXI HPM1 FPD** parameters.
28. Click **OK** to close the window.
29. Press **F6** to validate the design.
30. Click **OK** when the **Validate Design** window opens indicating the validation was successful.
31. In the **BLOCK DESIGN** window, click the **Sources** tab in the upper left-hand corner.
32. Right-click **ZCU102** and select **Create HDL Wrapper**.
33. When the **Create HDL Wrapper** window opens, let Vivado manage wrapper and auto-update, then click **OK** and wait while the design sources is created.
34. In the **BLOCK DESIGN** window in the upper left corner on the **Sources** tab, expand the **ZCU102_wrapper**.
35. Right-click **ZCU102_i: ZCU102** and select **Generate Output Products**.
36. Leave the default settings in the **Generate Output Products** window. Click **Generate** and wait while the IP is being generated.
37. Click **OK** when the **Generate Output Products** window displays **Out-of-context module run was launched for generating output products**.

Exporting the ZCU102 Hardware and Launching Xilinx SDK

1. In the main Vivado Design Suite toolbar select **File > Export > Export Hardware**.
2. When the **Export Hardware** window opens, leave the **Include bitstream** option unchecked and the `<Local to Project>` selected in the **Export to** option.
3. Click **OK**.
4. In the main Vivado Design Suite toolbar select **File > Launch SDK**.
5. When the Launch SDK window opens, leave Exported location set to `<Local to Project>` and choose a location for where the SDK workspace is created or leave the setting `<Local to Project>`.
6. Xilinx SDK is launched and wait while the hardware gets imported.

After importing the hardware, you should see a project named **ZCU102_wrapper_hw_platform_0** that was automatically created based upon the ZCU102 evaluation board.

Creating the First Stage Boot Loader (FSBL) and Board Support Package (BSP)

1. In the main Xilinx SDK toolbar, select **File > New > Application Project**.
2. In the Application Project window, change the following three configuration items:
 - a. Project name: **FSBL**
 - b. Processor: **psu_cortexa53_0**
 - c. Board Support Package: **Create New: A53_BSP**
3. Click **Next**.
4. On the Template page of the New Project window, select **Zynq MP FSBL**.
5. Click **Finish**.

The workspace is now updated with the FSBL project and the A53_BSP board support package.

6. By default, in Xilinx SDK all the projects are built automatically. You can disable this from the main toolbar by selecting **Project > Build Automatically** to disable this feature.

Validate the Hardware and Software with the Hello World Application

1. In the main Xilinx SDK toolbar, select **File > New > Application Project**.
2. In the Application Project window change the following three configuration items:
 - a. Project name: **HelloWorld**
 - b. Processor: **psu_cortexa53_0**
 - c. Board Support Package: **Use Existing: A53_BSP**
3. Click **Next**.
4. On the **Template** page of the **New Project** window, select **Hello World**.
5. Click **Finish**.
6. Right-click the **A53_BSP** project and click **Board Support Package Settings**.
7. In the **Board Support Package Settings** window expand the **Overview** tree and then click **standalone**.
8. Make sure the **stdin** and **stdout** functions are mapped to `psu_uart_0` and click **OK**.
9. Right-click the `HelloWorld` project and select **Build Project**.
10. Connect either the USB JTAG connector J2 to the ZCU102 development board and then to a computer or connect the Platform JTAG to the ZCU102 via J8 and the associated hardware to a computer.

11. Connect a USB cable from the USB Serial port connector J83 on the ZCU102 board to a computer and make note of which COM port was enumerated with the *Silicon Labs Quad CP2108 USB to UART Bridge: Interface 0*.
12. Open a terminal program such as PuTTY or Tera Term and connect to the COM port listed above at 115,200 baud.
13. On the ZCU102 development board set the dip switch to configure the board for JTAG boot mode as shown in [Figure 6](#).
14. Right-click the HelloWorld project and select **Run As > Launch on Hardware (System Debugger)**.
15. Verify that "Hello World" is output on the terminal screen. The hardware and software is properly configured and is now ready for use.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the [Design Hubs](#) page.

Note: For more information on Documentation Navigator, see the [Documentation Navigator](#) page on the Xilinx website.

References

1. *Zynq UltraScale+ MPSoC Device: Technical Reference Manual* ([UG1085](#))
2. *Programming BBRAM and eFUSES* ([XAPP1319](#))
3. *Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices* ([XAPP1323](#))
4. *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* ([UG1209](#))
5. *Zynq UltraScale+ MPSoC PUF Characterization Report* (RPT236)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/28/2021 Version 1.1	
Introduction	<ul style="list-style-type: none"> Added a note for further clarity about boot header permissibility Added a note about the PUF Key
06/26/2018 Version 1.0	
Initial Xilinx release.	N/A

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

© Copyright 2018–2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included here in are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. All other trademarks are the property of their respective owners.