Provision CYW20829/CYW89829 to SECURE LCS

About this document

Scope and purpose

This document describes how to provision the AIROC™ CYW20829 Bluetooth® LE 5.4 MCU to SECURE LCS.

***Note**: Pin locations and electrical and timing specifications of the physical connection are not a part of this document; see the device datasheet [1] for details .*

Intended audience

This document is intended for OEMs who wish to use the security features of CYW20829/CYW89829.

Table of contents

1 Overview

AIROC™ CYW20829 Bluetooth® LE 5.4 MCU supports the Arm® SWJ-DP interface for programming eFuse using the Serial Wire Debug (SWD) interface. The part can be programmed after it is installed in the system.

1.1 System security

To protect the hardware and the code (firmware) from being tampered with, you must implement security at the beginning of any new project.

The products may be hacked via the following ways:

- **Direct access to the debug port**: With the use of common debug tools and dongles, one can reprogram the firmware or access the internal data
- **Direct connection to a communication port**: Depending on the firmware, the direct connection to any of the communication ports (SPI, I2C, or a UART) may allow the firmware to be read or updated with a non-sanctioned software
- **Network connections**: The firmware could be accessed with networks such as Bluetooth®, Wi-Fi, or Ethernet
- **Third-party code**: Third-party code that is installed in the device after it has been shipped can also be used to access the firmware

1.2 Basic definitions

Table 1 Definitions

Terminology	Description
Code signing	Process of calculating a hash of the code binary and encrypting the hash with a private key and appending this to the code binary.
Debug Access Port (DAP)	Interface between an external debugger/programmer and CYW20829/CYW89829 for programming and debugging. This allows connection to one of the access ports (CM33_AP).
Digital signature	Encrypted digest (hash of a data set). For example, the encrypted hash of the user application.
eFuse	One-time programmable (OTP) memory that by default is ‘0’ and can be changed only from ‘0’ to ‘1’. eFuse bits may be programmed individually and cannot be erased.
Hash	Crypto algorithm that generates a repeatable but unique signature for a given block of data. This function is non-reversible.
Lifecycle stage (LCS)	Security mode in which the device is operating. To the user, it has only four stages of interest: NORMAL, NORMAL_NOSECURE, SECURE, and RMA.

(table continues...)

Table 1 (continued) Definitions

Terminology	Description
Public key	When using asymmetrical cryptography such as RSA or ECC, a public key is used to validate the firmware that was signed by the private key. It can be shared, but it should be authenticated or secured so it cannot be modified.
Private key	When using asymmetrical cryptography such as RSA or ECC, the private key is used to sign (encrypt the hash) of the firmware after it is built but prior to being loaded into the device. It must be kept in a secured location so that it cannot be viewed or stolen.
RMA	Return merchandise authorization
ROM	Read-only memory is non-volatile and is programmed as part of the fabrication process and cannot be reprogrammed.
ROM boot	After a reset, the CPU starts executing code that has been programmed into ROM. This code cannot be altered.
Serial Memory Interface (SMIF)	A serial peripheral interface (SPI) communication interface to serial memory devices, including NOR flash, SRAM, and non-volatile SRAM.
AES	Advance Encryption Standard; used to encrypt a block or blocks of data with the specified key.
TRNG	True random number generator: Unlike a pseudo random number generator, the output of TRNG is always random and cannot be hacked.
Table of contents 2 (TOC2)	An area in the external flash of CYW20829/CYW89829 that is used to store parameters and pointers to objects used for “Secure Boot”. Locations of one application pointer is stored here. The first pointer, Application1, must point to the first executable user code, which may be the bootloader or just the application.
OEM	Original equipment manufacturer: Although technically it means the manufacturer, in this document it can be referred as the person or entity who has control over CYW20829/CYW89829.

2 eFuse

The eFuse is a one-time programmable (OTP) eFuse array consisting of 1024 bits, which are mostly reserved for the system. However, some of the bits are available for storing security key information and hash values and can be programmed by the user for device security.

- eFuse memory can be programmed (eFuse bit value changed from '0' to '1') only once. If an eFuse bit is blown, it cannot be cleared
- Programming the eFuse bits requires the associated I/O supply to be at a specific level: The device VDDIO_1 (or VDDIO if only one VDDIO is present in the system) supply must be set to 2.5 V (±5%)
- Where eFuse needs to be programmed after the application is running, supply 3.0 V to VDDIO to allow the application to start, and then lower VDDIO to 2.5 V for eFuse programming

Table 2 Pins required for eFuse programming

Spec ID#	Parameter	Description	Min	Typ	Max	Unit	Details/ conditions
SID7E	VDDIO_1	Supply when programming eFuse	2.38	2.5	2.62	V	eFuse programming voltage

Because blowing an eFuse is an irreversible process, when provisioning to SECURE LCS at the factory, it is recommended to be done under controlled factory conditions using Infineon-provided provisioning tools.

2.1 Device lifecycle

The device lifecycle is a key aspect of the AIROC™ CYW20829/CYW89829 Bluetooth® MCU's security. Lifecycle stages follow a strict irreversible progression dictated by programming the eFuse bits (changing the value from '0' to '1'). This system is used to protect the internal device data and code at the level required by the customer. Lifecycle stages are governed by the LIFECYCLE_STAGE eFuse bits and can only be advanced to the next lifecycle state as shown in Figure 1. For example, once in the SECURE lifecycle stage, the device can never return to the normal state.

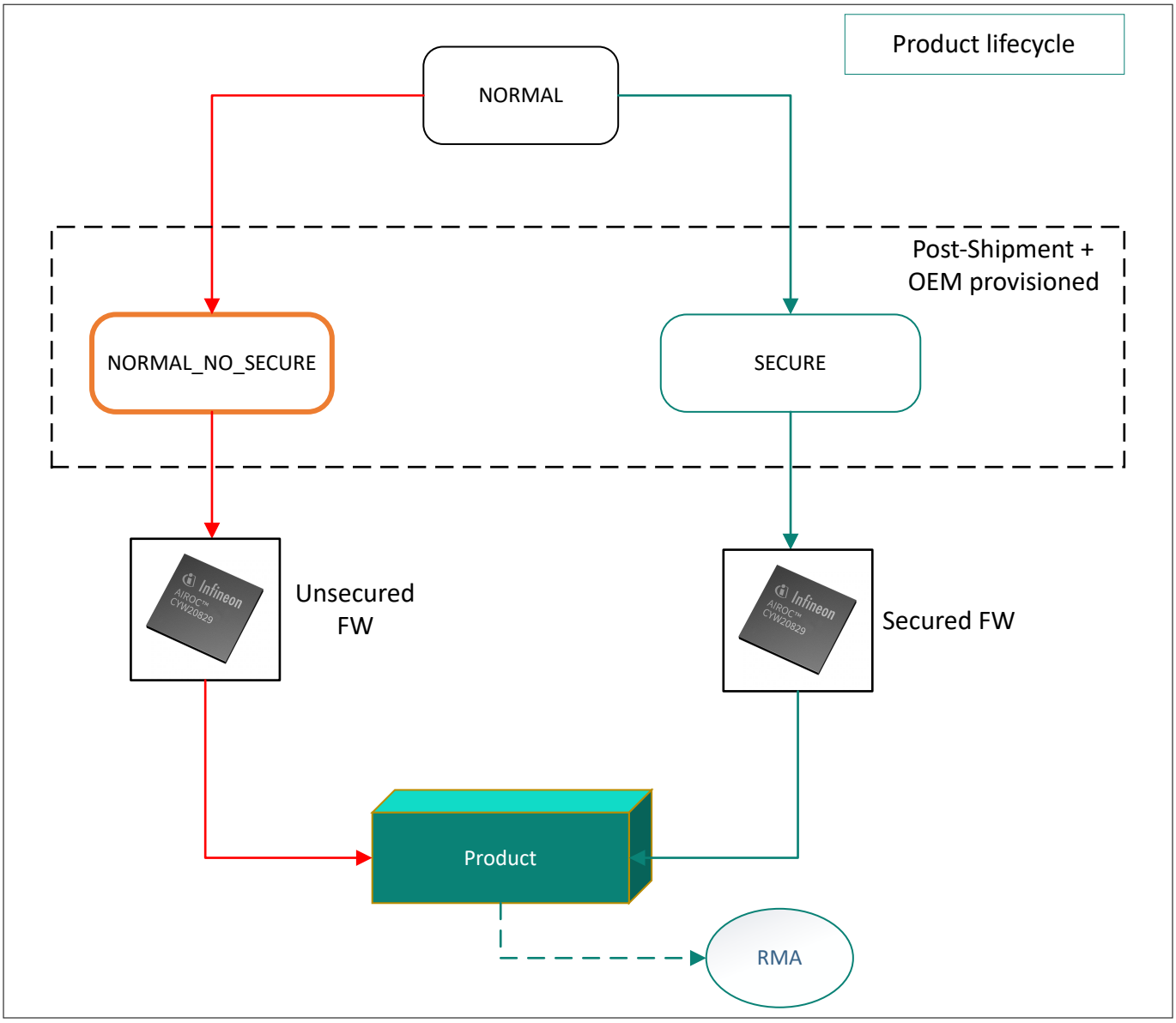

Figure 1 Device lifecycle

2.1.1 NORMAL

NORMAL is the initial lifecycle stage of the device when handed over to the OEM. In this stage, the trim values and wounding values are written into the eFuse. The device will be transitioned to a different lifecycle stage as suited by the manufacturer.

2.1.2 NORMAL_NO_SECURE

NORMAL_NO_SECURE is the lifecycle stage of a non-secured device. By default, users have full debug access and may program certain areas in the eFuse using the Infineon tool. Once a device is transitioned to NORMAL_NO_SECURE LCS, you can no longer transition it to SECURE LCS. For more details, see Provisioning to SECURE LCS.

2.1.3 SECURE

SECURE is the lifecycle stage of a secured device. To transition to SECURE LCS, complete the following tasks. Failure to do so could leave you with an inoperable device.

1. Generate the private-public key pair that can be used by the tool provided by Infineon or a third-party tool
2. Use the Infineon tool to write the public key hash into the eFuse
3. Sign the firmware image using the private key generated
4. Transition to SECURE LCS stage by using the proper policy mentioned in the Provisioning to SECURE LCS Section

In the SECURE LCS stage, the protection state is set to 'secure'. A secured device will boot only when the authentication of its flash boot and application code succeeds.

For more information on the steps, see Provisioning to SECURE LCS.

After the MCU is in the SECURE LCS stage, it is irreversible. The debug ports may be disabled depending on your preferences, which means that there is no way to reprogram or erase the device with a hardware programmer/debugger. You should have a secondary application in case there are any upgrades required in which the primary application will remain stationary and will always be verified by the BOOT_ROM and the secondary application gets upgraded (for example, MCU_Boot (primary application) can be used to upgrade the firmware (secondary application)).

***Note**: The code (user application) should be tested in the NORMAL lifecycle stages before the device moves to the SECURE lifecycle stage. This is to prevent a configuration error that could cause the part to be no longer accessible to program, thereby making it unusable.*

3 Provisioning to SECURE LCS

To transition to SECURE LCS, the following pins must be made available to be accessed by the SWD programmer.

Table 3 Pins required for eFuse programming

Pin name	Description
VDDIO_1	Supply for GPIO ports and eFuse programming. See Table 2 for power supply requirements.
P1.2	SWD_IO
P1.3	SWD_Clock
XRES	Active LOW system reset without an internal pull-up resistor

The following are needed as the prerequisite for transioning to Secure LCS:

- Python 3.12
- Infineon OpenOCD, which is included in ModusToolbox™ software. Can be downloaded from the ModusToolbox™ software webpage

With a default installation of ModusToolbox™ software, the default locations for the OpenOCD root diretory is:

- OpenOCD root directory: `C:\Users\[user_name]\ModusToolbox\tools_3.x\openocd`

For more details, see the CYW20829/CYW89829 Section in the CySecureTools webpage.

3.1 Quick start

3.1.1 Installing CySecureTools package

Install CySecureTools

To install CySecureTools, do the following:

1. Open a **Modus-shell** window by double-clicking on the `Cygwin.bat` file located in the `ModusToolbox\tools_3.2\modus-shell` directory
2. Ensure that you have the latest version of pip installed by running the following command in the **Cygwin** terminal window:

```
python -m pip install --upgrade pip
```

3. Install **CySecureTools** by running the following command:

```
python -m pip install cysecuretools
```

Update CySecureTools

To update the CySecureTools package, do the following:

- Run the following command:

```
python -m pip install --upgrade –force-reinstall cysecuretools
```

***Note**: During installation, you may see errors saying that cysecuretools requires package version X, but you have package version Y that is incompatible. In most cases, these can be safely ignored.*

Verify the Installation

To verify the installation, do the following:

- Run the following command to show the path of the installed **CySecureTools** package:

```
python -m pip show cysecuretools
```

```
The default installation path is C:\Users\[user_name]
\AppData\Local\Programs\Python\Python312\Lib\site-packages\cysecuretools
```

3.1.2 Set a path to the on-chip debugger

In the Modus shell terminal window, navigate to the Cysecuretools directory, and then run the command below to set the OpenOCD root directory.

***Note**: Specifying the path is not mandatory if you have ModusToolbox™ installed on your machine. OpenOCD from the ModusToolbox™ directory is used by default.*

```
cysecuretools set-ocd --name openocd --path <PATH_TO_OPENOCD_ROOT_DIRECTORY>
```

Make sure you provide the path to the root directory of OpenOCD (NOT the *bin* directory).

3.1.3 Define a target

This target name will be used as a `-t` option value with each command. In this case, use `-t cyw20829`.

Example command:

```
cysecuretools -t cyw20829 -p <POLICY> <COMMAND> [OPTIONS]
```

3.1.4 Create a new project

This copies the list of files required to start using the tool to the current working directory.

Example command:

```
cysecuretools -t cyw20829 init
```

3.1.5 Define the policy

Policy is a text file in JSON format that contains a set of properties for the device configuration (e.g., enabling/ disabling debug access ports, SMIF configuration, keys information, etc.)

Policy files are located in the policy directory of the user project; the path to a file is specified when almost any command is run. Select a policy file based on your needs.

For example, if you have ModusToolbox™ installed, the policy files will be located in:

```
C:\Users\[user_name]\ModusToolbox\tools_3.1\python\Scripts\policy
```

The package contains the following policy types for the CYW20829/CYW89829 target:

- SECURE – For provisioning a device with keys and then converting the device to the SECURE LCS.
  - When provisioning to Secure LCS, the CM33 debug access port will be permanently disabled by default. This means that there will be no access CYW20829/CYW89829. Future firmware update and manufacture RF testing is not possible when the CM33 debug access port is disabled
  - Users have the option to provision the CYW20829/CYW89829 to Secure LCS with the CM33 debug access port enabled. To do this, open the 'policy_secure.json' file inside the policy folder with a text editor
  - Look for the entry 'dead_ap_cm33' and set the 'value" to “Enable'. The default value is 'Permanently Disable'
  - ***Note**:* *Once you provision the 20829/89829 to Secure LCS with the CM33 debug access port enabled, it will always remain enabled which could result in security risks.*
- NO_SECURE – For provisioning a device without keys and then converting the device to the NORMAL_NO_SECURE LCS
- REPROVISIONING_SECURE – For reprovisioning an already secured device (previously provisioned with the SECURE policy type, allows the configuration of a limited set of settings. The file contains only those settings that can be changed during reprovisioning
- PROVISIONING_NO_SECURE – For reprovisioning a non-secured device (previously provisioned with the NO_SECURE policy type), allowing the configuration of a very limited set of settings. The file contains only those settings that can be changed during reprovisioning
- HCI - Allows a device to work without external memory by launching the Host Control Interface (HCI) application placed in the ROM

3.2 Key creation

1. Create an OEM key pair by specifying the key ID (‘0’ or ‘1’). The keys is generated into the files specified in the 'keys' Section of the policy. You can create both ‘0’ and ‘1’
2. Provision the device to check the policy for key paths. If key ‘0’ exists, provision it into the device. If both key ‘0’ and key ‘1’ exist, provision both
3. Sign the image by specifying the key ID (‘0’ or ‘1’). The OEM key path will be set according to the policy
4. Reprovision device. If key ‘0’ exists, create a reprovisioning packet with the OEM key ‘0’. If key ‘1’ exists, create a reprovisioning packet with the OEM key ‘0’ and ‘1’. If key ‘0’ does not exist, it indicates an error

Example command:

```
cysecuretools -t cyw20829 -p policy/policy_secure.json create-key -k 0
```

Table 4 Parameters for the 'create-key' command

Name	Optional/required	Description
`-k, --key-id [0|1]`	Optional (mutually exclusive with the `--output` option)	The OEM key ID. This ID defines the paths where the public and private keys will be saved based on the selected policy prebuild and post build sections. Depending on the selected value ‘0’ or ‘1’, `oem_key_0` or `oem_key_1` will be used.
`-o, --output [private] [public]`	Optional (mutually exclusive with the `--key-id` option)	Key pair output files. This is an alternative option and can be used to create a key in the different location from the policy location. If it is specified, the `--key-id` option will be ignored. However, using `--key-id` is preferable because it allows avoiding accidental provisioning and signing with a key from different pairs. As the option value, you must provide private and public key paths. Specify the option multiple times to create multiple key pairs.
`--aes`	Optional (mutually exclusive with the `--key-id` and `--output` options)	Indicates whether to generate an AES key for image encryption. The key will be saved by the path specified in the policy `smif_aes_key` field.
`--key-path`	Optional	Used to generate a key into a specific path. Applicable with the `--aes` or `--template` option.
`--overwrite` / `--no-overwrite`	Optional	Indicates whether to overwrite a key if it already exists.
`--template`	Optional	A JSON file containing the public key modulus and exponent. The option is used for creating a PEM file based on public key modulus and exponent.
`--hash-path`	Optional	A path to save the public key hash.

3.3 Provision the device

Use the following command to provision the device into SECURE LCS:

```
cysecuretools -t cyw20829 -p policy/policy_secure.json provision-device
```

3.3.1 Reprovision the device

Once a device has been provisioned, it can only be reprovisioned. Reprovisioning allows configuring a limited set of settings (for example, provisioning an additional OEM key, revoking `icv_pubkey_0` and `oem_pubkey_0` keys, changing the anti-rollback counter, and so on). The *policy_reprovisioning_secure.json* file contains only those settings that can be changed.

Example command:

```
cysecuretools -t cyw20829 -p policy/policy_secure.json reprovision-device -k 0
```

Table 5 Parameters for the 'reprovision-device' command

Name	Optional/required	Description
`-k, --key-id [0|1]`	Optional	The OEM private key ID used to sign the reprovisioning packet. This ID informs the tool about the key location. The tool will use the path specified in the selected policy post build section.
`--key-path`	Optional	Sets OEM private key that is used to sign the reprovisioning packet. Overrides the `--key-id` option.
`--probe-id`	Optional	Probe serial number.
`--existing-packet`	Optional	Skips reprovisioning packet creation and uses the existing packet. This may be useful when a packet with RAM application input parameters already exists and the tool does not have to generate it again.
`--signature`	Optional	Name of the file containing the signature.

3.4 Image signing

The user application can have a signature that is calculated and kept along with the application in the external flash. The application signature is generated by calculating the hash (SHA256) of the application image and generating the signature with the ECDSA NIST P256 algorithm with the OEM private key. The OEM public key is being used to verify the signature by BOOT_ROM. You can opt for only signature or signature along with encrypting the user app as explained in the following section.

Example commands:

- Sign the image without encryption:

```
cysecuretools -t cyw20829 -p policy/policy_secure.json sign-image -I image.bin -o
image_signed.bin--key-id 0 –image-type APP_PC0
```

- Sign the image with encryption:

```
cysecuretools -t cyw20829 -p policy/policy_secure.json sign-image -i image.bin -o
image_signed.bin--key-id 0 –image-type APP_PC0–encrypt –app-addr 0x8000200
```

Table 6 Parameters for the 'sign-image' command

Name	Optional/required	Description
`-i, --image`	Required	User application file. The output file format is based on the input file extension (can be *.bin* or *.hex*).
`-k, --key-id [0|1]`	Optional (mutually exclusive with the `--key-path` option)	The OEM key ID. This ID defines the key location based on the selected policy post build section. Depending on the selected value '0' or '1', `oem_priv_key_0` or `oem_priv_key_1` will be used.
`--key-path`	Optional (mutually exclusive with the `--key-id` option)	Key file path to sign the image. This is an alternative option and can be used to specify different from the policy key file location. If it is specified, the `--key-id` option will be ignored. However, using the `--key-id` option is preferable because it allows avoiding accidental provisioning and signing with keys from different pairs.
`--signature`	Optional	The name of the file containing the signature. Used to add an existing signature to the image.
`-R, --erased-val [0|0xff]`	Optional	The value that is read back from erased flash.
`-o, --output`	Optional	Signed image output file. If not specified, a copy of the input file will be created with the unsigned prefix, then the input file will be signed.
`--encrypt`	Optional	Enables the encryption feature. If present, the image will be encrypted after signing.
`--enckey`	Optional	Path to the AES key. If absent, the key path is taken from the policy `smif_aes_key` field.
`--hex-addr`	Optional	Adjusts address in *.hex* output file. The default value is '0'. Ignored for output images in *.bin* format.

(table continues...)

Table 6 (continued) Parameters for the 'sign-image' command

Name	Optional/required	Description
`--app-addr`	Optional	The address of the application to encrypt. Accepts values as hexadecimal or decimal numbers. The default value is '0'.
`-f, --image-format`	Optional	The image format defines the image header size, signing, and encryption algorithms. Values: • `bootrom_ram_app` – RAM application started by BootROM • `bootrom_next_app` – External memory application started by BootROM (e.g., MCUboot) • `mcuboot_user_app` – Application started by MCUboot. The default value is `mcuboot_user_app`.
`-H, --header-size`	Optional	Sets the image header size. Overrides the header size defined by the `--image-format` option.
`-S, --slot-size`	Optional	Sets the maximum slot size. The default value is `0x20000`.
`--pad`	Optional	Adds padding to the image trailer up to the maximum slot size. Adds padding to the 32-byte Boot Magic to the end of the trailer. Required for MCUboot image upgrade.
`--confirm`	Optional	Adds the "Image OK" byte to the image trailer. Required for MCUboot image upgrade.
`--overwrite-only`	Optional	Sets "Overwrite" mode in the MCUboot image header instead of "Swap".
`--update-key-id [0|1]`	Optional	Sets the OEM private key ID used to sign the update data packet.
`--update-key-path`	Optional	The key used to sign the update data packet. Overrides the `--update-key-id` option.
`--min-erase-size`	Optional	Sets the minimum erase size.
`--align [1|2|4|8]`	Optional	Sets the flash alignment. The default value is '8'.
`-v, --version`	Optional	Sets the image version in the image header.
`-d, --dependencies`	Optional	Add dependency on another image, format: "(`<image_ID>,<image_version>`), "... ".
`--image-id`	Optional	Image ID. The value is used to update the NV counter. The default value is '0'.

3.4.1 Image encryption

For image encryption/decryption, CySecureTool uses the AES algorithm with a direct use of a 128-bit key, which is provisioned to the device. This key must be used for image encryption. This key is provisioned into the device by storing it into eFuse. If you decide to use image encryption, the Key ID 1 space is used to store this encryption key. The user application image may consist of multiple images (for example, MCUboot application, L1 user application, and next user application). Each image may have its own TOC and application descriptor data. However, during the encryption/decryption process, the boot ROM considers the images as a single image. First, TOC2 and descriptor are used to get the image start address. All information except MCUboot TOC2 and descriptor is encrypted.

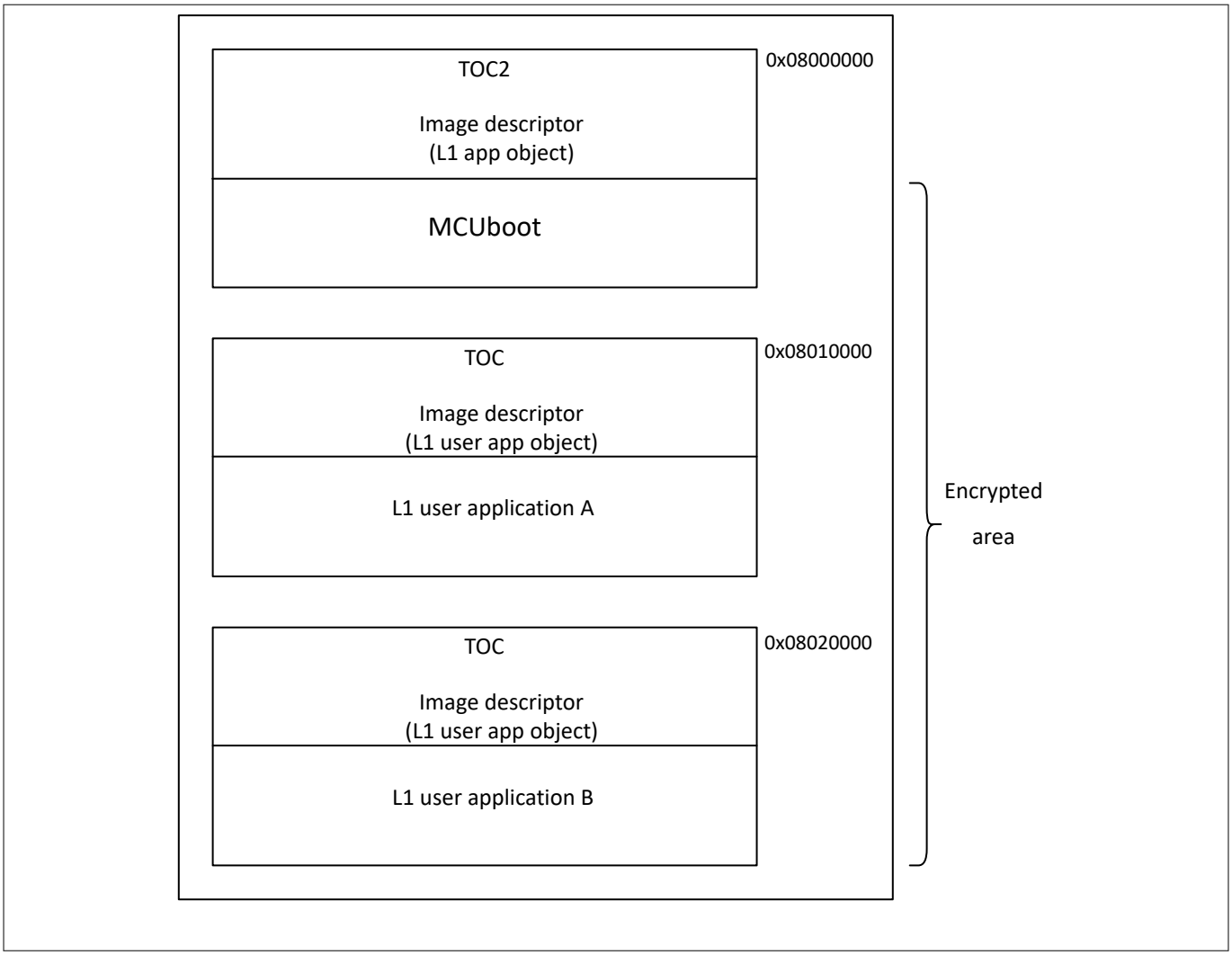

Figure 2 Image encryption

A 128-bit encryption key is provisioned as is. There is an option to have two OEM keys (`oem_pub_key_0` and `oem_pub_key_1`). However, if the encryption is used, only `oem_pub_key_0` can be used. The encryption key will be placed instead of the second OEM key hash.

3.5 Create debug certificate

The debug certificate is used by BOOT_ROM to enable the CM33-AP and/or Sys-AP when it is temporarily disabled.

***Note**: The certificate cannot enable an access port that is permanently disabled by the access restrictions. Also, the debug certificate can be used to enable/disable invasive or non-invasive debug for CM33-AP.*

Example command:

```
cysecuretools -t cyw20829 -p policy/policy_secure.json debug-certificate -t packets/
debug_cert.json -o packets/debug_cert.bin -k 0
```

The command creates a debug or RMA certificate binary based on the template provided in CySecureTools. The certificate must contain an OEM public key for further verification. If it is signed using the local private key, the

public key is extracted from the private key. If the certificate is going to be signed using the hardware security module (HSM), you must create a non-signed certificate and then specify the public key.

BOOT_ROM validates the debug certificate in boot flow; upon authentication, its starts secure debugging.

Table 7 Parameters for the 'debug certificate' command

Name	Optional/required	Description
`--non-signed`	Optional	Flag indicating that debug certificate will not be signed.
`-t`, `--template`	Optional	Path to the certificate template in JSON format.
`-k`, `--key-id [0|1]`	Optional (mutually exclusive with the `--key-path` option)	OEM private key ID used to sign the certificate. This ID defines the key location based on the selected policy post build section. Depending on the selected value '0' or '1', `oem_priv_key_0` or `oem_priv_key_1` will be used.
`--key-path`	Optional (mutually exclusive with the `--key-id` option)	Either a private key path for signing the certificate or a public key to be added to the certificate. This is an alternative option and can be used to specify different location from the policy key file location. If it is specified, the `--key-id` option will be ignored.
`--sign [cert] [signature]`	Optional	Option for signing an existing certificate using existing signature file.
`-o`, `--output`	Required	File where the debug certificate is saved.

3.6 Return merchandise authorization (RMA)

The command advances the device lifecycle stage to RMA. The customer must transition the part to the RMA lifecycle stage, when the customer wants Infineon to perform failure analysis on the part. The customer shall erase all the sensitive data prior to invoking the system call that transitions the part to RMA. If the device is in SECURE LCS, the transition process requires a certificate. Use the debug certificate with the RMA flag enabled (see Create debug certificate). The certificate must be signed with the OEM private key.

```
cysecuretools -t cyw20829 convert-to-rma --cert packets/debug_cert.bin
```

4 Nonvolatile memory subsystem

- The eFuse array can be read eight bits at a time using normal memory-mapped AHB register reads or corresponding system calls
- eFuses are programmed one bit at a time using a command register

These eFuse bytes are accessible for production programming:

- **0x040 - 0x04F**: OEM key ‘0’ hash (16 bytes hash of OEM public key)
- **0x050 – 0x05F**: OEM key ‘1’ hash (16 bytes hash of OEM public key)
- **0x064 - 0x067**: OEM key revocation and SMIF configuration

5 eFuse data mapping

See Table 8 and Table 9 for eFuse data mapping.

Table 8 eFuse data mapping

Offset	Width (bit)	Name	Description
0x000	32		System reserved
…	…	System reserved	
0x03C	32		
0x040	32	OEM_KEY_0_HASH	OEM key ‘0’ hash (16 bytes hash of OEM public key)
0x050	32	OEM_KEY_1_HASH	OEM key ‘1’ hash (16 bytes hash of OEM public key)
0x060	32	System reserved	System reserved
0x064	32	OEM_CONFIG	OEM key revocation and SMIF configuration
0x068	32	NOT_USED_1	NOT_USED_1 (32-bit unused)
0x06C	32	NOT_USED_2	NOT_USED_2 (32-bit unused)
0x070	32		System reserved
…	…	System reserved	
0x07C	32		

Table 9 eFuse bit mapping

Name	Bits	Function	Description
OEM_KEY_0_HASH	31:0	DATA32	OEM key ‘0’ hash (16-byte hash of OEM public key)
OEM_KEY_1_HASH	31:0	DATA32	OEM key ‘1’ hash (16-byte hash of OEM public key)

(table continues...)

Table 9 (continued) eFuse bit mapping

Name	Bits	Function	Description
OEM_CONFIG	11:0	SMIF_CFG	SMIF configuration (12 bits): • Chip Select (1 bit) - 0=CS0, 1=CS1 • Data width (2 bits) - 00 = 1X, 01 = 2X, 10 = 4X, 11 =8X • Data Select (2 bits) - 00 = CY_SMIF_DATA_SEL0 - 01 = CY_SMIF_DATA_SEL1 - 10 = CY_SMIF_DATA_SEL2 - 11 = CY_SMIF_DATA_SEL • Addressing Mode (1 bit) - 0 = 3-byte addressing - 1 = 4-byte addressing - SMIF crypto enable (1bit) - 0 = No encryption - 1 = L1 image is encrypted • Reserved1 (1 bit) • SMIF Configuration Index (4 bits) - 0000 = No configuration - 0001 = SFDP 1.5 and above part. Quad Enable Requirement present in SFDP. - 0010 = QER_1: Bit 1 of Status Register 2 - Write uses 2 bytes using 01h. - 0011 = QER_2: Bit 6 of Status Register 1 - Write uses 1 byte. - 0100 = QER_3: Bit 7 of Status Register 2 - Write uses 1 byte. - 0101 = QER_4: Bit 1 of Status Register 2 - Write uses 1 or 2 bytes. - 0110 = QER_5: Bit 1 of Status Register 2 - Write status uses 01h. - 0111 = QER_6: Bit 1 of Status Register 2 - Write uses 1 byte using 31h. - 1000 to 1011 = Four non-SFDP parts - 1100 to 1110 = Reserved - 1111 = ROM app
	12	OEM_KEY_REVOCATION	OEM key revocation
	31:13	UNUSED_13	19-bit unused

References

Device documentation

[1] Infineon Technologies AG: *002-31976: CYW20829 MCU datasheet*; Available online
[2] Infineon Technologies AG: *AIROC™ CYW20829 Bluetooth® LE: Designing a secure system*

Development kits

[3] *CYW920829M2EVK-02*; Available online
[4] *CYW920829B0M2P4TAI100EVK*; Available online
[5] *CYW920829B0M2P4EPI100EVK*; Available online

Tool/Tool documentation

[6] Infineon Technologies AG: *ModusToolbox™ software - The Infineon IDE for IoT designers*; Available online
[7] Infineon Technologies AG: *002-24375: Eclipse IDE for ModusToolbox™ user guide*; Available online

Revision history

Document revision	Date	Description of changes
**	2024-03-05	Initial release
*A	2024-06-25	Template update; no content change
*B	2025-05-09	Updated the procedure of installing the CySecureTools package.
*C	2025-06-02	Updated Return merchandise authorization (RMA).

Trademarks

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth® SIG, Inc., and any use of such marks by Infineon is under license.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-06-02

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-jcl1718087896580

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.