PSoC™ 64 RMA mode for field failure analysis

About this document

Scope and purpose

This document guides you on how to transition the PSoC™ 64 MCU into the return merchandise authorization (RMA) life cycle stage. Devices in RMA mode can be returned to Infineon for failure analysis. The companion code example in CE237184 – PSoC™ 64 MCU: Transition to RMA demonstrates RMA mode transitioning of the PSoC™ 64 MCU.

Intended audience

This application note is intended for users who want to learn more about the RMA process on the PSoC™ 64 family of devices.

Table of contents

1 Introduction

Return merchandise authorization (RMA) is the life cycle stage of the PSoC™ 64 MCU that allows the device to return to Infineon for the purpose of evaluation or failure analysis. To place a device in the RMA stage, the internal code must successfully execute the `TransitionToRMA` system call. This system call is only valid when the device is in `SECURE_CLAIMED` stage. See the System call API for details about this system call.

To transition the device into RMA, update the RMA policy section in the policy file before the device is provisioned. This policy contains information about the key used for the RMA process as well as the flash regions that may contain sensitive code or data that needs to be destroyed before the device is transitioned into RMA. For details on how to update this policy, see RMA policy.

For information on getting started with the PSoC™ 64 MCU, see the PSoC™ 64 Secure Boot SDK User Guide.

2 Life cycle stages

PSoC™ 64 MCUs have configurable and non-volatile life cycle stages. Writing to the proper eFuse bits governs a strict, irreversible progression of life cycle stages. eFuse is a non-volatile memory area, with each bit being one-time programmable (OTP). 1 byte (8 fuses) is reserved for life cycle. For more information, see AN221111 - PSoC™ 6 MCU designing a custom secured system.

The life cycle stages can transition only from one stage to the next and cannot be reversed. For example, when the life cycle has transitioned from `SECURE_UNCLAIMED` to `SECURE_CLAIMED` stage, it cannot be transitioned back to `SECURE_UNCLAIMED` stage. It may remain in `SECURE_CLAIMED` stage or transition to RMA.

Once the device has been transitioned to the RMA stage, it can never be changed to the `SECURE_UNCLAIMED` or `SECURE_CLAIMED` stage. Therefore, Infineon recommends the user to transition the device to the RMA stage only if the device needs to be analyzed for failure analysis.

By default, the PSoC™ 64 MCU leaves the Infineon factory in the `SECURE_UNCLAIMED` stage.

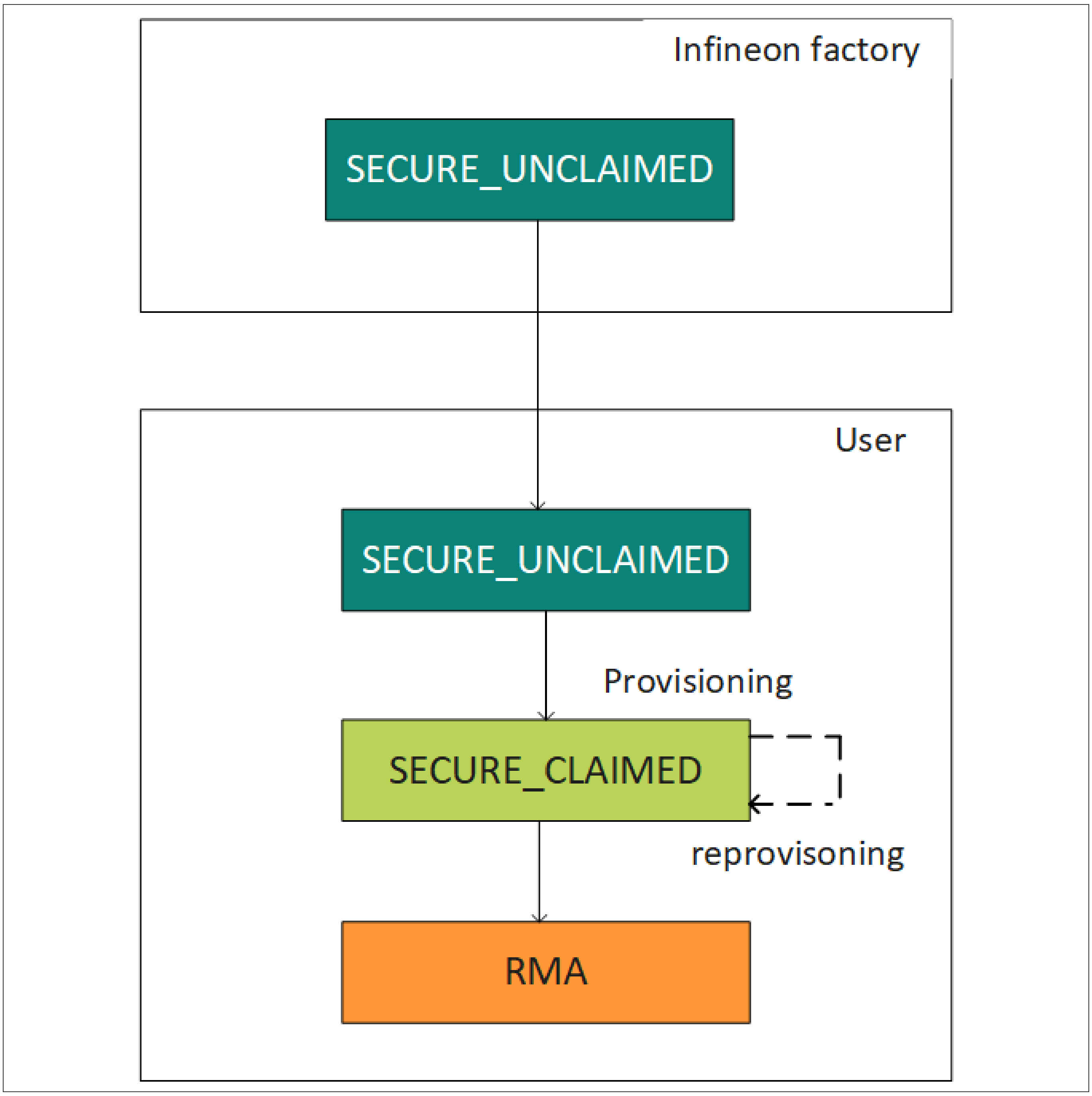

Figure 1 **Life cycle stages**

3 RMA process

The RMA process involves the user transitioning the device into the RMA stage and shipping the device to Infineon, along with the signed JWT package. Ensure that the device does not contain any user-sensitive information at the RMA stage, as it will be erased, as per the user-defined RMA policy, before moving the device to RMA.

After the device is transitioned into the RMA stage, when it boots, it does not attempt to execute any code in user flash. Instead, it waits for an `OpenRMA` system call. The `OpenRMA` system call requires the same signed JWT packet as a parameter that was used to place the device in the RMA stage. When the part is sent to Infineon for failure analysis, it must include the signed JWT packet that contains the device `DIE_ID`. The JWT packet will only work for the part with the corresponding `DIE_ID`.

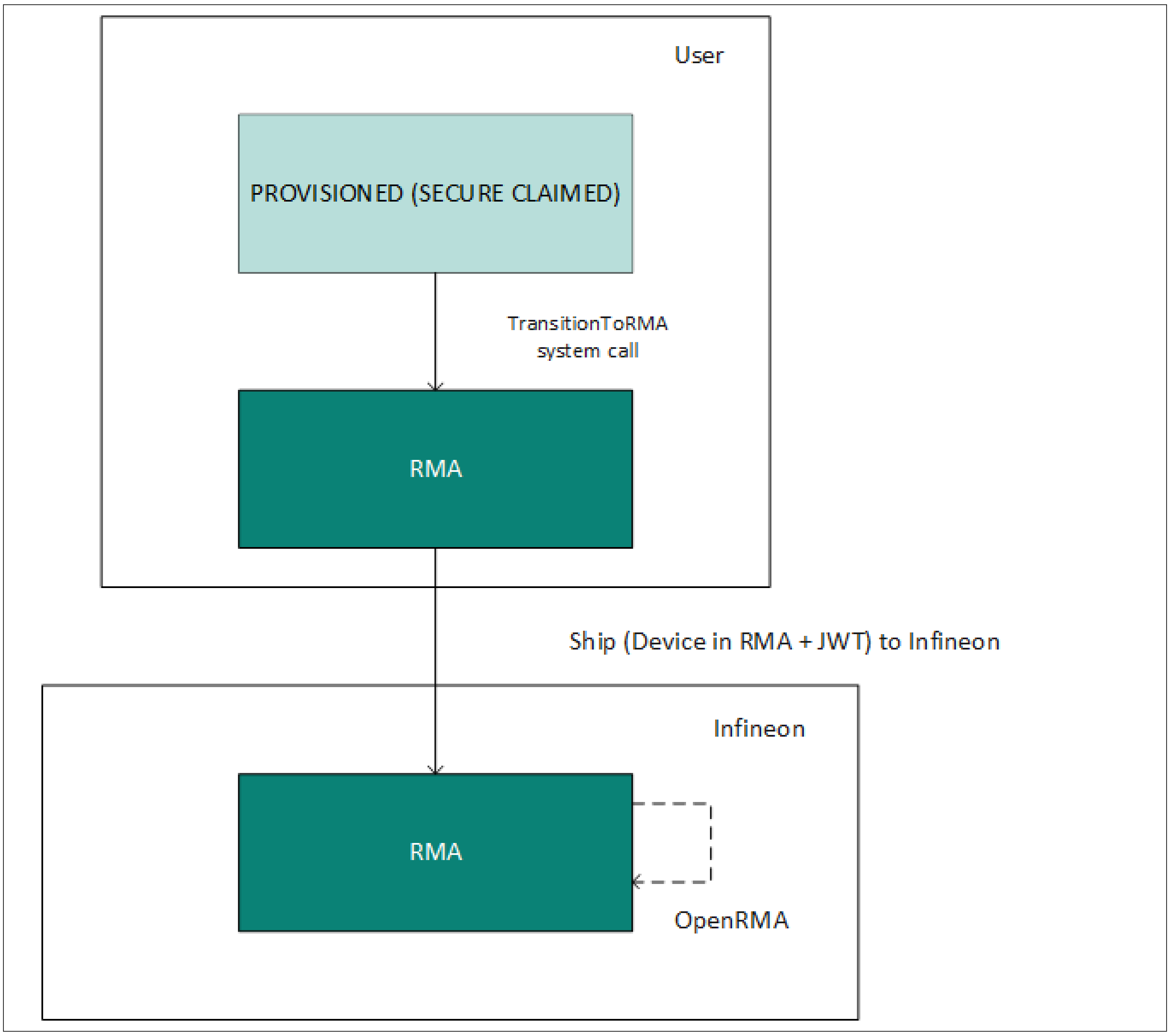
User
PROVISIONED (SECURE CLAIMED)
TransitionToRMA system call
RMA
Ship (Device in RMA + JWT) to Infineon
Infineon
RMA
OpenRMA

Figure 2 **RMA process**

Use any of the following options to perform the `TransitionToRMA` system call:

- **Option 1:** The Cortex®-M0+ or Cortex®-M4 user application may be designed to perform the `TransitionToRMA` system call upon a user-defined trigger, such as a button press.
  - An example of a device that uses the Cortex®-M4 application and button press to perform `TransitionToRMA` can be found in the Code example.

> *Note:* *If the device failure occurs during the booting of the user application, you cannot perform `TransitionToRMA` using this option.*

- **Option 2:** The system call may be performed through the SYS-AP; to be able to achieve this, the SYS-AP should not be in a disabled state. Ensure that SYS-AP access is enabled in the policy file before provisioning the device. See more details about the debug port policies in the PSoC™ 64 Secure Boot SDK User Guide.

Follow these steps to ensure support for future RMA transitions:

1. Before provisioning the device, ensure that the `"rma"` policy fields are updated in the provisioning packet, as described in the RMA policy.
2. If you plan to use the SYS-AP to perform the RMA transition, ensure that the SYS-AP ports are enabled as per the policy described in the PSoC™ 64 Secure Boot SDK User Guide.
3. Generate the JSON web token (JWT) signed by the key specified in the `"rma"` policy. This JWT must be passed as a parameter to execute the `TransitionToRMA` system call.
4. Optionally, design your application to support the `TransitionToRMA` system call and preload the JWT packet when required to be used.
5. Provision and program the device as described in the PSoC™ 64 Secure Boot SDK User Guide.

Follow these steps to perform the RMA transition on parts requiring failure analysis:

1. To transition the device life cycle to RMA, execute the `TransitionToRMA` system call. The system call can only be executed in the `SECURE_CLAIMED` life cycle stage.

   The system call can be performed from a supported user application or via the debug access port (DAP) port if it is enabled.

   > *Note:* *As this transition performs eFuse programming, VDDIO0 should be set to 2.5 V for successful operation. For VDDIO0 supply, see the "Pinouts" section in the PSoC™ 6 MCU: datasheet.*

2. Once the device is in the RMA stage, it may be shipped to Infineon along with the same JWT packet that was used to transition the device into RMA.
3. Infineon will perform an `OpenRMA` system call on the returned device to evaluate or perform a failure analysis.

The JWT required at the time of the RMA transition must have the following features:

- At the time of provisioning the device, the “`rma`” policy must specify a key that signs the JWT packet and contains the `DIE_ID`.
- The user has the option to pregenerate this JWT packet and store it in flash or to provide a method to generate and transfer it to the device through a user-defined interface.

> *Note:* *The cysecuretools package supports the `transit-to-rma` and `open-rma` commands.*
>
> *For details on cysecuretools, see the PSoC™ 64 Secure Boot SDK User Guide.*

4 System call API

The process of transitioning the device into the RMA stage is implemented as a system call. System calls can be performed by Cortex®-M0+ and Cortex®-M4, or via the debug access port (DAP). Each has a reserved IPC structure through which the user can request the execution of a system call. The opcodes (8-bit numbers) are used to identify different system calls. The caller sends an IPC structure pointer via IPC message to be written to the IPC data register. The IPC structure pointer points to the SRAM address that holds the LCS parameters. This results in an NMI interrupt in Cortex®-M0+, where the system call is executed. When the system call execution is complete, the IPC structure is released, and the caller can read a return code in the data register of the IPC structure. For more information about PSoC™ 64 system calls, see the device-specific documents in the PSoC™ 6 reference manuals.

The `TransitionToRMA` system call transitions the parts from the SECURE to the RMA life cycle stage. Before transitioning to the RMA stage, the system call destroys the eFuse and erases flash content five times as specified in the RMA policy. A JWT packet that contains the `DIE_ID` that is signed by a specified installed private key is passed as a parameter to the system call. The `TransitionToRMA` system call uses indirect parameter passing.

The system call return code is a 32-bit value. The return code and output data are passed in a similar way as input parameters. The data register of the IPC structure can contain either the return code or a pointer to the SRAM, where the return code and output data are stored (if they are provided by the system call).

Table 1 IPC structure data registers

Address	Value to be written	Description
	IPC_STRUCT.DATA register	
Bits[31:0]	SRAM_SCRATCH_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
	SRAM_SCRATCH	
Bits[31:24]	0x3B	Opcode for TransitionToRMA
Bits[23:0]	0xXXXXXX	Not used
	SRAM_SCRATCH + 0x04	
Bits[31:0]	SRAM_SCRATCH2_ADDR	SRAM address where the API parameters are stored. This must be a 32-bit aligned address.
	SRAM_SCRATCH2	
Bits[31:0]	Length	Length of JWT packet
	SRAM_SCRATCH2 + 0x04	
Array[Length]	JWT packet	The JWT packet should contain an "auth" object including the valid DIE_ID range. It should be signed by the private key mentioned in the *Debug Policy/RMA/Key*.

Table 2 TransitionToRMA system call

Address	Return value	Description
Bits[31:28]	0xA = SUCCESS 0XF = ERROR	Status code
Bits[27:0]	Error status	See the "System Call Status" section in device-specific documents in the PSoC™ 64 reference manuals for more details.

For more details about the JWT format, see JSON web token.

5 RMA policy

A policy, in the context of a PSoC™ 64 device, is a JSON file that defines the device's operation modes, debug access, code updates, and so on. The policy is injected into the device during provisioning and consists of a collection of JSON tokens defined by Infineon and a number of tokens defined by the user.

Update the `"rma"` section of the debug policy in the *provisioning JWT* file before provisioning the device. Set the RMA permission to *"allowed"* and update the key used to sign the JWT. This policy should include any areas of flash that may contain proprietary code or sensitive data.

Code Listing 1 RMA policy structure

```
"rma" : {
    "permission" : "STRING VALUE ",
    "destroy_fuses" : [
        {
            "start" : Integer Value,
 "size" : Integer Value
        }
    ],
    "destroy_flash" : [
        {
            "start" : Integer Value,
            "size" : Integer Value
        },
        {
            "start" : Integer Value,
            "size" : Integer Value
        }

    ],
    "key" : Integer Value
}
```

Table 3 RMA policy fields

Object	Description	Range of valid values
rma: permission	Specifies if RMA is allowed	• **"disabled":** RMA is not allowed • **"allowed":** The RMA stage is available and can be entered by presenting a JWT to a system call API. The system will destroy fuse and flash contents as specified in `<destroy_fuses>` and `<destroy_flash>` before transitioning to the RMA stage.
rma: destroy_fuses: start	Starting fuse bit number for region	0~65535. Check the device-specific datasheet for the permitted eFuse address.

(table continues...)

Table 3 (continued) RMA policy fields

Object	Description	Range of valid values
rma: destroy_fuses: size	Number of fuse bits in region	0~65535. Check the device-specific datasheet for the permitted eFuse size.
rma: destroy_flash: start	Starting byte address of region (will be rounded down to nearest program/erase boundary)	0~0xFFFFFFFF. Check the device-specific datasheet for the permitted flash address.
rma: destroy_flash: size	Size in bytes of region (will be rounded up so the region is an integral number of program/erase units)	0~0xFFFFFFFF. Check the device-specific datasheet for the permitted flash size.
rma: key	The key slot number of the key used to validate authorization to enter the RMA stage.	The key ID must be >3, pointing to the key provisioned in the *custom_pub_key* field.

Table 4 Sample RMA policy

JSON field	Description
"rma": {	Defines RMA behavior
"permission" : "allowed",	RMA mode is allowed
"destroy_fuses" : [	Indicates the eFuse region to be destroyed if entering the RMA mode
{	
"start": 888,	Starting bit number of the eFuses to be destroyed
"size":136	Size in bits to be destroyed
}],	
"destroy_flash" : [	Indicates the flash region to be destroyed if entering the RMA mode
{	
"start": 268435456,	Start address of the flash to be destroyed (0x10000000)
"size": 512	Size in bytes to be destroyed
}],	
"key": 5	KeyID used to validate an RMA request
}	

For information on the PSoC™ 64 provisioning policies, see the PSoC™ 64 Secure Boot SDK User Guide.

6 JSON web token

The JSON web token (JWT) is a standard (RFC7519) URL-safe way of storing and transmitting any JSON object. A JWT packet is digitally signed to authenticate a sender and ensure the integrity of the payload.

A JWT is represented as a sequence of URL-safe parts separated by a period ('.'). Each part contains a `base64url-encoded` value. PSoC™ 64 supports JWTs that consist of three parts: header, payload, and signature. The JWT header format is as follows:

```
{
"alg": "ES256",
"typ": "JWT"
}
```

The *"alg"* field defines the digital signature algorithm used for packet signing. *"ES256"* is the only supported value, which is an `ECDSA256` digital signing algorithm. The *"typ"* field defines the token type for transmitting any JSON object. Here, we are using the "JWT" token for the transmission.

A JWT payload is a `base64url-encoded` JSON object. For the RMA use case, the payload consists of the `auth` object, which includes the valid `DIE_ID` range.

A JWT signature is a `base64url-encoded` digital signature produced by the `ECDSA256` algorithm.

The following example showcases the JWT token utilized in the context of the RMA transition:

```
eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdXRoIjp7ImRpZV9pZCI6eyJtYXgiOnsiZGF5IjoyNTUsImxvdCI6MTY
3NzcyMTUsIm1vbnRoIjoyNTUsIndhZmVyIjoyNTUsInhwb3MiOjI1NSwieWVhciI6MjU1LCJ5cG9zIjoyNTV9LCJtaW4iOns
iZGF5IjowLCJsb3QiOjAsIm1vbnRoIjowLCJ3YWZlciI6MCwieHBvcyI6MCwieWVhciI6MCwieXBvcyI6MH19fX0.z6ePvuJ
TcY0z3azJFGpzcq0-4bxxpgfL7H-E4V-Dg6UGpwpLqf8pFXdMIXNXbQKCYW1Pq5HM7npZXNTUDtgEEw
```

The following table represents the fields of the JWT token:

Table 5 JWT token

JWT parts	Encoded	Decoded
Header	`eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCJ9`	{ "alg": "ES256", "typ": "JWT"}

(table continues...)

Table 5 (continued) JWT token

JWT parts	Encoded	Decoded
Payload	`eyJhdXRoIjp7ImRpZV9pZCI6eyJtYXgiOnsiZGF5IjoyNTUsImxvdCI6MTY3NzcyMTUsIm1vbnRoIjoyNTUsIndhZmVyIjoyNTUsInhwb3MiOjI1NSwieWVhciI6MjU1LCJ5cG9zIjoyNTV9LCJtaW4iOnsiZGF5IjowLCJsb3QiOjAsIm1vbnRoIjowLCJ3YWZlciI6MCwieHBvcyI6MCwieWVhciI6MCwieXBvcyI6MH19fX0`	{ "auth": { "die_id": { "max": { "day": 255, "lot": 16777215, "month": 255, "wafer": 255, "xpos": 255, "year": 255, "ypos": 255 }, "min": { "day": 0, "lot": 0, "month": 0, "wafer": 0, "xpos": 0, "year": 0, "ypos": 0 } } } }
Signature	`z6ePvuJTcY0z3azJFGpzcq0-4bxxpgfL7H-E4V-Dg6UGpwpLqf8pFXdMIXNXbQKCYW1Pq5HM7npZXNTUDtgEEw`	, KEY)

To generate the JWT from a JSON file, use the `sign-cert` command provided by the cysecuretools package (see README_PSOC™ 64.md for usage detail).

For details on cysecuretools, see the PSoC™ 64 Secure Boot SDK User Guide.

7 Code example

The CE237184 - PSoC™ 64 MCU: Transition to RMA CE demonstrates the process of transitioning the PSoC™ 64 MCU into RMA mode. In this example, the system call is performed by the Cortex®-M4 application on the Cortex®-M0+ processor. It contains a pregenerated JWT and executes the `TransitionToRMA` system call upon a button press by the user. For more information about this code example, see the "README" section in the CE237184 – PSoC™ 64 MCU: Transition to RMA.

8 Abbreviations

Term	Description
API	Application programming interface
DAP	Debug access port
IPC	Inter-process communication
JSON	JavaScript object notation
JWT	JSON web token
MCU	Microcontroller unit
NMI	Non-maskable interrupt
OTP	One-time programmable
RMA	Return merchandise authorization
SRAM	Static random-access memory
URL	Uniform resource locator

Revision history

Document version	Date of release	Description of changes
**	2023-09-26	• Initial release

Edition 2023-09-26

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-ijf1690351183858