Secure Firmware Upgrade for CYW207xx and CYW208xx

ModusToolbox™

About this document

Scope and purpose

This user guide describes the Secure Firmware Upgrade (SFU) for AIROC™ CYW207xx and CYW208xx devices using ModusToolbox™ software.

Reference documents

This user guide should be read in conjunction with the following documents and code examples:

[1] AIROC™ Firmware Upgrade Library. Document number: 002-19289: AIROC™ Bluetooth® SDK

[2] ota_firmware_upgrade, Document number: 002-19289: AIROC™ Bluetooth® code example

[3] hci_firmware_upgrade, Document number: 002-19289: AIROC™ Bluetooth® code example

Table of Contents

About this document **1**
Table of Contents **2**
1 Introduction **3**
2 IoT resources **4**
3 Preparing the secure firmware image **5**
4 Keys generation **6**
5 Project modification **7**
6 Source code to support SFU **8**
6.1 Application versioning 8
6.2 Changes to the GATT database for OTA firmware upgrades 9
6.3 Firmware Upgrade Library Access for OTA Firmware Upgrade 11
6.4 Firmware Upgrade Library access for HCI firmware upgrade 13
7 Building the SFU image **16**
8 Sign the SOTAFU image **17**
9 Upgrade the firmware **18**
Revision history **19**

1 Introduction

The Secure Firmware Upgrade (SFU) SDK feature uses Elliptic Curve Cryptography to implement a secure method to upgrade the firmware of devices that are based on Infineon AIROC™ CYW207xx and CYW208xx Bluetooth® chips. The Elliptic Curve Digital Signature Algorithm (ECDSA) is used to sign the image which is verified by the firmware after the download and before the new image is allowed to be executed on the chip.

The device memory is split into active and passive partitions. The new firmware image is downloaded to the passive partition and then verified. If the verification is successful, the partitions are swapped and the new image is executed during the next device startup. This two-step process ensures that interrupted or unverified downloads do not cause interruption of the service.

The SFU process also verifies that the image for the correct product has been downloaded. The application image embeds four bytes of the Product Version information which includes two bytes of a Product ID, one byte with the Major Version and one byte with the Minor Version. The Product Version information is verified during the upgrade.

The digital signature is prepared by calculating an SHA256 hash of the firmware image file and then signed using the ECDSA algorithm with the secp256r1 curve.

Verification of the new image will fail unless the following conditions are met:

- ECDSA verification must pass. This guarantees the image integrity and validates that the image was created by the authorized party which had access to the private key.

If the verification fails, the new image will not be activated and the existing application will continue to run without interruption.

Details of the firmware upgrade protocol are discussed in the AIROC™ Firmware Upgrade Library document **[1]** included in the AIROC™ Bluetooth® SDK.

2 IoT resources

Infineon provides a wealth of data at **http://www.cypress.com/internet-things-iot** to help you to select the right IoT device for your design, and quickly and effectively integrate the device into your design. Infineon provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates. Customers can acquire technical documentation and software from the Infineon Support Community website (**http://community.cypress.com**).

3 Preparing the secure firmware image

The AIROC™ Bluetooth® SDK provides tools to create a signed firmware image. Some changes are also required to the application code to support the SFU feature. The following is a summary of the required steps to make the application SFU-compatible.

1. Create a public and private key pair (see Section **4**). Save the private key in a safe location. When an updated version of the application is developed, it should be signed with the same private key.
2. Modify the application project to include a file with the public key and to link with the library supporting upgrade and memory access (see Section **5**).
3. Modify the application source code as follows (see Section **6**):
   a) Define the application Product Version information.
   b) Add the OTA Secure Upgrade service to the GATT database if using Over-the-Air (OTA) Firmware Upgrade.
   c) Process the commands and data received from the peer during the SFU process.
4. Build the application. See the kit guide for your device included with your AIROC™ Bluetooth® SDK for the description of the build process.
5. Sign the built image using the private key generated in step 1 (see Section **8**).

The following is a summary of the steps required to prepare the next version of the application:

1. If security vulnerability is being fixed, increment the major version and optionally set the minor version to zero (0). Otherwise, increment the minor version.
2. Build the new version of the application.
3. Sign the application using the same private key that was used to sign the original version of the application.

Note: *The AIROC™ Bluetooth® SDK contains source code and binaries for applications to generate ECDSA keys, sign and verify the image. See the* mtb_shared/wiced_btsdk *project in the ModusToolbox™ Project Explorer pane, under the* mtb_shared/wiced_btsdk/tools/btsdk-utils/ecdsa256 *directory.*

4 Keys generation

The AIROC™ Bluetooth® SDK includes a utility to create the private and public keys (*ecdsa_genkey.exe*). The application can be used to generate a public-private key pair or to create a public key based on a previously generated private key. In the latter case, pass the name of the file with the binary private key as a parameter.

When the application is executed, it generates the following files:

- *ecdsa256_key.pri.bin* – Contains the binary representation of the private key.
- *ecdsa256_key.pub.bin* – Contains the binary representation of the public key.
- *ecdsa256_key_plus.pub.bin* – Contains the binary representation of the public key concatenated with two bytes containing the number of zero bits in the key. This can be used when the public key is stored in the OTP area and can be tampered with.
- *ecdsa256_pub.c* – Source code containing the public key that can be included in the AIROC™ application project that will perform SFU.

5 Project modification

Copy the *ecdsa256_pub.c* file generated as per **Keys generation** to the application source folder. The ModusToolbox™ build system will automatically build all sources found under the application folder.

Modify the application *Makefile*:

1. Initialize the `OTA_FW_UPGRADE` and `OTA_SEC_FW_UPGRADE` Makefile variables to '1'.
2. Include the Firmware Upgrade Library components:

```
ifeq ($(OTA_FW_UPGRADE),1)
COMPONENTS+=fw_upgrade_lib
endif
```

You can either use the Library Manager to add the Firmware Upgrade Library to the application or manually perform similar changes. The Library Manager will "git clone" the library repository to *mtb_shared/wiced_btsdk/dev-kit/libraries* and then add a corresponding search path to the *libs/mtb.mk* file. The *deps/btsdk-ota.mtb* file is added to reference the btsdk-ota library repo.

Add code to the application to use the btsdk-ota library.

The **ota_firmware_upgrade** sample application demonstrates the use of the btsdk-ota library, and is available for download using the ModusToolbox™ Project Creator tool.

The 'hci_firmware_upgrade' sample application provides an example of firmware upgrade via HCI UART rather than OTA.

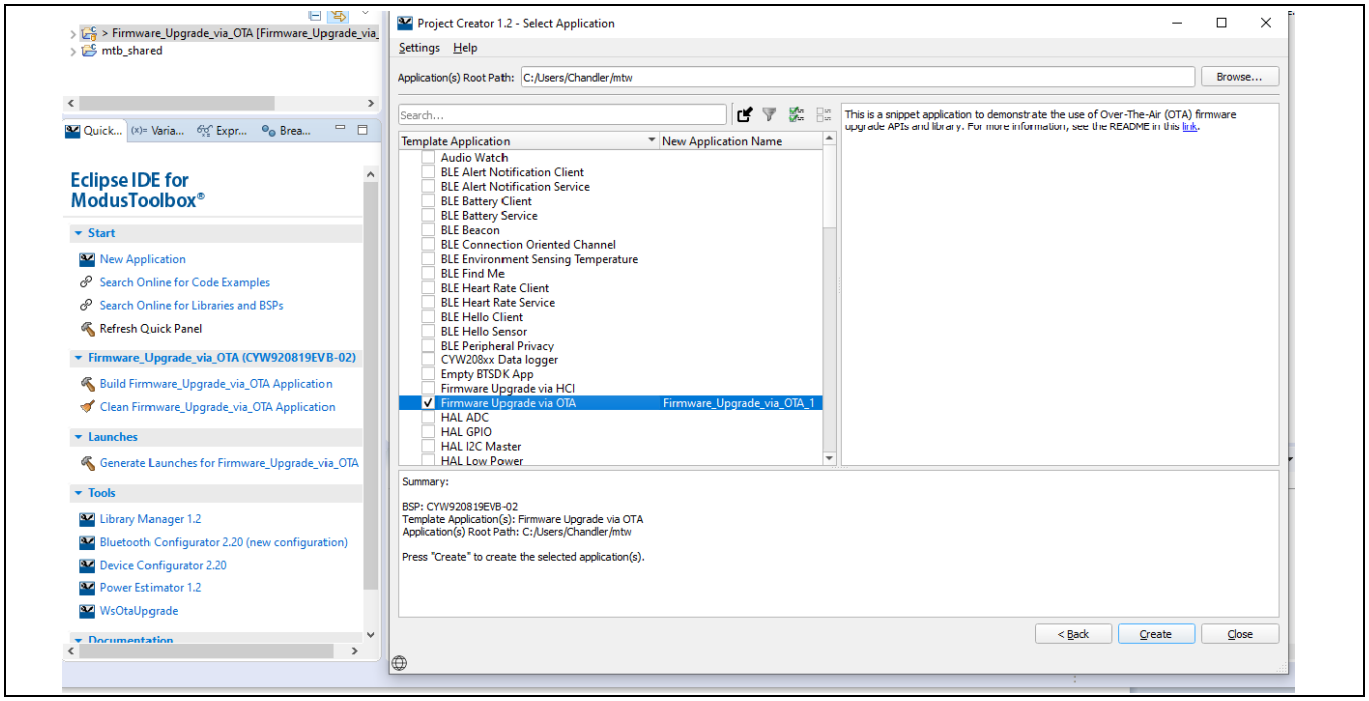


Figure 1 ModusToolbox™ Project Creator

6 Source code to support SFU

Each application that supports SFU should have the code to process SFU commands and image data (see **Firmware Upgrade Library Access**). In addition, any application supporting OTA firmware upgrade should provide a GATT database section that includes descriptions of the SFU service and characteristics (see Changes to the GATT database for OTA firmware upgrades).

6.1 Application versioning

Each application Makefile can define three values for version control. During the upgrade, the application will verify that the download has the same product ID and that the major version is not lower.

Some sample definitions are as follows:

```
APP_VERSION_APP_ID = 0x1234
APP_VERSION_MAJOR = 1
APP_VERSION_MINOR = 0
```

If not defined, these values will default to zero. If the product ID is zero, the product ID verification is skipped. These values are stored in the firmware header that is part of the current firmware and also present in the SFU download image. The value for the current firmware can be retrieved using an API from the Firmware Upgrade Library:

```
/*****************************************************************************
*
* Stored in DS header area, product id and version used to compare with upgrade image.
*
*****************************************************************************/
typedef struct
{
    uint16_t product_id;
    uint8_t major;
    uint8_t minor;
} wiced_bt_application_id_t;

/*****************************************************************************
* Function Name: wiced_get_current_app_id_and_version
*****************************************************************************//**
* \brief Retrieves application version information from nvram.
*
* \details The application calls this function to retrieve the appplication id
*          and version to compare with the downloaded image.
*
*****************************************************************************/
wiced_bool_t wiced_get_current_app_id_and_version(wiced_bt_application_id_t
*app_id_and_version);
```

The current application ID is compared with the application ID in the first packet of the SFU image by the Firmware Upgrade Library.

Do not change the product ID when a new version of an application is developed.

Increment the major version when a security vulnerability is fixed. Examples of security vulnerability include application crashes under certain conditions and unauthorized remote access to internal data. Image verification requires that the major version number be the same or greater than that of the existing version, and this prevents downgrading to a prior version that has vulnerability. Change the minor version if no critical problem was fixed.

6.2 Changes to the GATT database for OTA firmware upgrades

To include SFU the application GATT database shall publish a special Infineon vendor-specific service. Definitions for the handles and UUIDs are included in the *wiced_bt_ota_firmware_upgrade.h* header file.

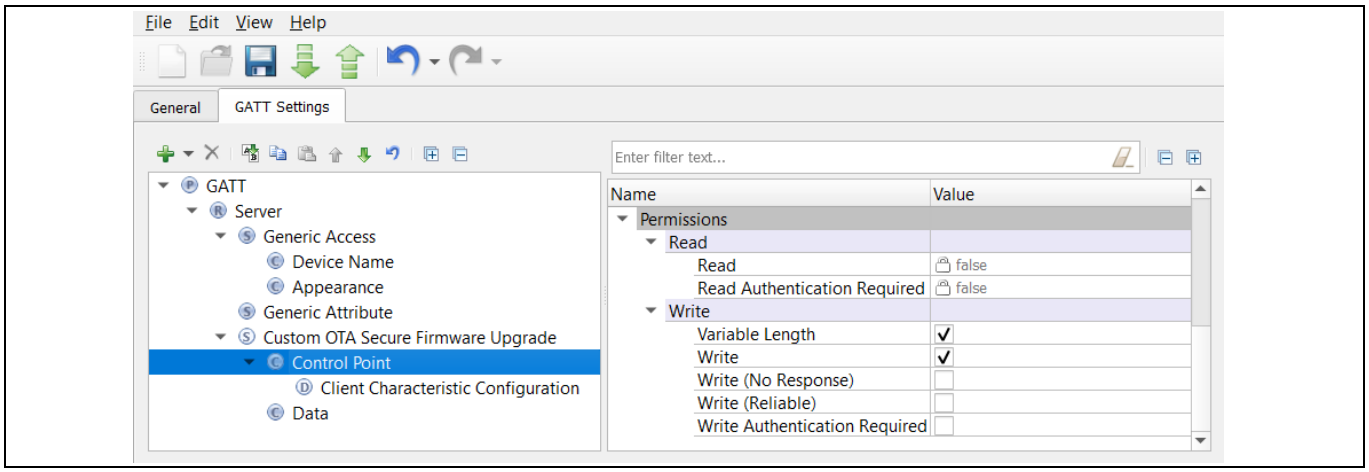
Typically, the application will need to include the following snippet in the GATT database. This can be done manually, or the Bluetooth® Configurator tool may be used to generate and/or modify the GATT database. In either case, the resultant GATT database must be registered with the stack using the `wiced_bt_gatt_db_init` API.

Do the following to add the secure OTA service using the Bluetooth® Configurator:

1. Launch the Bluetooth® Configurator tool from the Quick Launch panel in the Eclipse IDE.
2. On the **GATT Settings** tab, right-click the **GATT** > **Server** entry, and then select **Add Service** > **Custom OTA Secure Firmware Upgrade**.

   This creates the Custom OTA Secure Firmware Upgrade service.
3. (Optional) Select the **Control Point, Client Characteristic Configuration,** and **Data** entries and configure **Write Authentication Required** for each entry to optionally utilize Bluetooth® security for the service.

Note that some applications may require or disallow Bluetooth® security, depending on the application needs. This setting maps to the `LEGATTDB_PERM_AUTH_WRITABLE` flag used in the following manual snippet example.



If modifying the GATT database manually, the handles used by the service should be in sorted order. Place the OTA service after the last application service in the GATT database structure. If an application already has an existing GATT database that is registered with the `wiced_bt_gatt_db_init` API, the following snippet can be added as-is in the application code:

```
// Handle 0xff00: Infineon vendor specific Secure OTA Upgrade Service.
PRIMARY_SERVICE_UUID128
    (HANDLE_OTA_FW_UPGRADE_SERVICE, UUID_OTA_SEC_FW_UPGRADE_SERVICE),

// Handle 0xff01: characteristic Secure Firmware Upgrade Control Point,
// Handle 0xff02: characteristic value.
CHARACTERISTIC_UUID128_WRITABLE
(
    HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_CONTROL_POINT,
    HANDLE_OTA_FW_UPGRADE_CONTROL_POINT,
    UUID_OTA_FW_UPGRADE_CHARACTERISTIC_CONTROL_POINT,
    LEGATTDB_CHAR_PROP_WRITE | LEGATTDB_CHAR_PROP_NOTIFY |
    LEGATTDB_CHAR_PROP_INDICATE,
    LEGATTDB_PERM_VARIABLE_LENGTH | LEGATTDB_PERM_WRITE_REQ |
    LEGATTDB_PERM_AUTH_WRITABLE
),

// Handle 0xff03: Declare client characteristic configuration descriptor
CHAR_DESCRIPTOR_UUID16_WRITABLE
(
    HANDLE_OTA_FW_UPGRADE_CLIENT_CONFIGURATION_DESCRIPTOR,
    UUID_DESCRIPTOR_CLIENT_CHARACTERISTIC_CONFIGURATION,
    LEGATTDB_PERM_READABLE | LEGATTDB_PERM_WRITE_REQ |
    LEGATTDB_PERM_AUTH_WRITABLE
),

// Handle 0xff04: characteristic OTA firmware upgrade data
// Handle 0xff05: characteristic value
// This characteristic is used to send portions of the FW image
CHARACTERISTIC_UUID128_WRITABLE
(
    HANDLE_OTA_FW_UPGRADE_CHARACTERISTIC_DATA,
    HANDLE_OTA_FW_UPGRADE_DATA,
    UUID_OTA_FW_UPGRADE_CHARACTERISTIC_DATA,
    LEGATTDB_CHAR_PROP_WRITE,
    LEGATTDB_PERM_VARIABLE_LENGTH | LEGATTDB_PERM_WRITE_REQ |
    LEGATTDB_PERM_AUTH_WRITABLE
),
```

6.3 Firmware Upgrade Library Access for OTA Firmware Upgrade

The WICED Firmware Upgrade Library (available from the Library Manager; see **Section 5**) implements the functionality that is common for all applications that require firmware upgrade.

The application needs to initialize the library, send notifications to the library when a connection to a peer device is established or dropped, and pass to the library packets received from the connected peer and destined for the OTA Firmware Upgrade service. This includes read, write, and indication confirmation messages received from the peer device.

Note: *In the samples below the code assumes that OTA Firmware Upgrade service is the last service in the GATT database.*

6.3.1 Library initialization

The AIROC™ Firmware Upgrade Library should be initialized any time during the application execution after the stack has been initialized and before the first GATT Write request is passed to the library.

The following snippet shows library initialization that is performed during device startup:

```
void app_init(void)
{
    /* Initialize app */
    wiced_bt_app_init();

    // arguments are key, optional status callback, and optional data send callback
    if (!wiced_ota_fw_upgrade_init(&ecdsa256_public_key, NULL, NULL))
    {
        WICED_BT_TRACE("OTA upgrade Init failure!!!\n");
    }

    // Continue with the application initialization
```

6.3.2 Connection handler

The application passes connection up/down events to the library whenever it receives notification from the stack.

The following snippet of the code shows how the application should process connection events:

```
wiced_bt_gatt_status_t app_connection_status_event(wiced_bt_gatt_connection_status_t
*p_status)
{
    // Pass connection up/down event to the OTA FW upgrade library
    wiced_ota_fw_upgrade_connection_status_event(p_status);

    // Continue with the event processing
```

6.3.3 Read handler

The following snippet of the code shows how the application should process GATT Read requests from the peer:

```
wiced_bt_gatt_status_t app_read_handler(uint16_t conn_id, wiced_bt_gatt_read_t *
p_read)
{
    // if read request is for the OTA FW upgrade service, pass it to the library to
    // process
    if (p_read->handle > HANDLE_OTA_FW_UPGRADE_SERVICE)
    {
        return wiced_ota_fw_upgrade_read_handler(conn_id, p_read);
    }
    // Continue with the event processing
```

6.3.4 Write handler

The following snippet of the code shows how the application should process GATT Write requests from the peer:

```
wiced_bt_gatt_status_t app_write_handler(uint16_t conn_id, wiced_bt_gatt_write_t *
p_write_data)
{
    // if write request is for the OTA FW upgrade service, pass it to the library to
    // process
    if (p_write_data->handle > HANDLE_OTA_FW_UPGRADE_SERVICE)
    {
        return wiced_ota_fw_upgrade_write_handler(conn_id, p_write_data);
    }
    // Continue with the event processing
```

6.3.5 Indication confirmation handler

The following snippet of the code shows how the application should process GATT Indication confirmations received from the peer:

```
wiced_bt_gatt_status_t app_conf_handler(uint16_t conn_id, uint16_t handle)
{
    // if indication confirmation is for the OTA FW upgrade service, pass it to the
    // library to process
    if (handle > HANDLE_OTA_FW_UPGRADE_SERVICE)
    {
        return wiced_ota_fw_upgrade_indication_cfm_handler (conn_id, handle);
    }
    // Continue with the event processing
```

6.4 Firmware Upgrade Library access for HCI firmware upgrade

The WICED Firmware Upgrade Library should be initialized any time during the application execution after the stack has been initialized and before the first HCI command is passed to the library.

The following snippet shows library initialization that is performed during device startup:

```
void app_init(void)
{
    /* Initialize app */
    wiced_bt_app_init();

    // arguments are key, optional status callback, and optional data send callback
    if (!wiced_hci_fw_upgrade_init(&ecdsa256_public_key, status callback, NULL))
    {
        WICED_BT_TRACE("HCI upgrade Init failure!!!\n");
    }

    // Continue with the application initialization
```

6.4.1 Command interface

HCI commands with opcode type `HCI_CONTROL_DFU_COMMAND_WRITE_COMMAND` are relayed to the Firmware Upgrade Library using `hci_fw_upgrade_handle_command()`. Commands from the host such as prepare, download, verify, and abort are handled in this way, as shown in the following code example:

```
    case HCI_CONTROL_DFU_COMMAND_WRITE_COMMAND:
        WICED_BT_TRACE("Write command: %d %d len %d\n", *p_data, *(p_data+1),
            length);
        if (!hci_fw_upgrade_handle_command(dfu_conn_id, p_data, length))
        {
            if(*p_data != WICED_HCI_UPGRADE_COMMAND_ABORT)
            {
                WICED_BT_TRACE("hci_handle_command failed.\n");
                rc = HCI_CONTROL_STATUS_FAILED;
            }
        }
        break;
```

6.4.2 Data transfer interface

HCI firmware upgrade data is transferred using packets with opcode `HCI_CONTROL_DFU_COMMAND_WRITE_DATA`. The packet data is be transferred to the library by calling `hci_fw_upgrade_handle_data()`, as in the following code example:

```
    case HCI_CONTROL_DFU_COMMAND_WRITE_DATA:
        WICED_BT_TRACE("Write data %B\n", p_data);
        if (!hci_fw_upgrade_handle_data(dfu_conn_id, p_data, length))
        {
            WICED_BT_TRACE("hci_handle_data failed.\n");
            rc = HCI_CONTROL_STATUS_FAILED;
        }
        else
        {
            WICED_BT_TRACE("Send data event to HCI\n");
            wiced_transport_send_data( HCI_CONTROL_DFU_EVENT_DATA, NULL, 0 );
        }
        break;
```

6.4.3 Events

When the status callback is provided the library will call it to provide library state updates, such as started, aborted, verification started, and completed. These events may be relayed to the host. The following is an example:

```
/*
 * This callback receives status updates from the Firmware Upgrade Library.
 */
void hci_dfu_status_callback(uint8_t status)
{
    WICED_BT_TRACE("status callback %d\n", status);
    switch(status)
    {
        case HCI_FW_UPGRADE_STATUS_STARTED:
            // now prepared for download
            wiced_transport_send_data( HCI_CONTROL_DFU_EVENT_STARTED, NULL, 0 );
            break;
        case HCI_FW_UPGRADE_STATUS_ABORTED:
            // response now aborted
            wiced_transport_send_data( HCI_CONTROL_DFU_EVENT_ABORTED, NULL, 0 );
            break;
        case HCI_FW_UPGRADE_STATUS_COMPLETED:
            // operation completed
            wiced_deinit_timer(&dfu_timer);
            wiced_transport_send_data( HCI_CONTROL_DFU_EVENT_VERIFIED, NULL, 0 );
```

```
            break;
        case HCI_FW_UPGRADE_STATUS_VERIFICATION_START:
            // verification started
            wiced_transport_send_data( HCI_CONTROL_DFU_EVENT_VERIFICATION, NULL, 0 );
            break;
        default:
            WICED_BT_TRACE( "hci_dfu_status_callback unknown status %d\n", status );
            break;
    }
}
```

7 Building the SFU image

When a target application is built, a separate binary file for the upgrade is created in the build directory. For example, when the **ota_firmware_upgrade** is successfully built for the CYW920819EVB-02 platform, the *OTA_FirmwareUpgrade_CYW920819EVB-02.ota.bin* file is created in the output folder for the application.

8 Sign the SOTAFU image

Execute the ecdsa_sign utility, passing the name of the OTA upgrade image file name as a parameter.

The command below is used to sign the *OTA_FirmwareUpgrade_CYW920819EVB-02.ota.bin* file created during the build process.

Note: The following procedure should be executed in a secure environment since a private key is used. The application uses the ecdsa256_key.pri.bin file.

```
ecdsa_sign OTA_FirmwareUpgrade_CYW920819EVB-02.ota.bin
hash = f8314ab0600f3b5c6477df534710cd944939686440bb72c33d859e8200421f89
r = 771b80bbee9fdf06e28c1ff2a86cd15ef5ddc9babababaa43d98018531bb3af223
s = c85975f5757749fa76983e3714b694bc00096f43ab6ad3990f238c30d006bc97
Signed file OTA_FirmwareUpgrade_CYW920819EVB-02.ota.bin.signed
```

The signed output file can be used to perform the upgrade. During execution the application prints out the hash of the image and the signature that can be used for debugging.

9 Upgrade the firmware

The *wiced_btsdk/tools/btsdk-peer-apps-ota* directory contains the source code and the executables to perform the upgrade in Windows or Android environments. Under Windows, pair the computer with the device running the image, and execute the WsOtaUpgrade utility passing the name of the signed firmware image file as a parameter. For example:

```
<path>\WsOtaUpgrade.exe OTA_FirmwareUpgrade_CYW920819EVB-02.ota.bin.signed
```

The application displays the progress of the operation and the status when the procedure is complete.

Revision history

Document Revision	Date of revision	Description of changes
**	2017-08-23	Initial release
*A	2018-10-08	Replaced “WICED Studio” with “ModusToolbox™”. Replaced “WICED Studio SDK” with “WICED SDK with PSoC™ 6 Support”. Updated Sales page.
*B	2019-02-19	Updated to include CYW20819 Replaced “WICED SDK with PSoC™ 6 Support” with “WICED Bluetooth® SDK”
*C	2019-04-24	Removed associated part family Updated for BT SDK release
*D	2019-10-15	Updated for ModusToolbox™ 2.0
*E	2021-01-15	Updated for ModusToolbox™ 2.2 and HCI (not over-the-air) firmware update
*F	2021-02-26	Added back description of application version and ID support.
*G	2021-10-28	Updated terminology per Bluetooth® SIG, updated WICED to AIROC™

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

Edition 2021-10-28
Published by
Infineon Technologies AG
81726 München, Germany

© 2021 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?
Go to www.cypress.com/support

Document reference
002-16561 Rev. *G

IMPORTANT NOTICE
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, 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.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

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.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

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.