# CYW4390X: OTP Programming and Using Secure Boot and Secure Flash

**Associated Part Family: CYW4390X**

This application note describes how the secure boot and secure flash features operate with the Cypress CYW4390X family of embedded wireless system-on-a-chip (SoC) devices. This family includes the CYW43903, CYW43907, and CYW43909 devices. Secure boot and secure flash features are not required during the early product development phase. As feature functionality involves programming one-time programmable (OTP) nonvolatile memory, it is important to exercise diligent precautions before starting the process.

This document is intended for design and applications engineers.

## 1 Introduction

The CYW4390X devices are single-chip IEEE802.11 a/b/g/n devices with integrated ARM Cortex-based processors and are intended for embedded Internet-of-Things (IoT) applications. The following processes are described in this document:

- Prepare the programming environment using the appropriate WICED-SDK package.
- Set up the options for encryption and authentication shown in Programming Secure Boot, Secure Flash, and SECURE_BIT on page 2.
- Set up the options to enable secure booting shown in Enable Secure Boot on page 3.
- Set up the options to enable secure flash shown in Enable Secure Flash on page 4.
- Set up the options to enable SECURE_BIT shown in Enable Secure Bit on page 4.
- Establish secure boot and secure flash keys (see Generating Secure Keys on page 4).
- Become familiar with programming commands shown in OTP Programming Commands on page 6.

Table 1. Recommended SECI Coexistence and HOST_WAKE Pin Mapping

| CYW4390X WLBGA Pin Name | CYW4390X SECI Coexistence Mapping |
|---|---|
| SPIM_CLK | SECI_OUT |
| GPIO_1 | SECI_IN |
| SPIM_CS_N | HOST_WAKE |

Figure 1 shows the BT to WLAN SECI block interface diagram.

## 2 Cypress Part Numbering Scheme

Cypress is converting the acquired IoT part numbers from Broadcom to the Cypress part numbering scheme. Due to this conversion, there is no change in form, fit, or function as a result of offering the device with Cypress part number marking. The table provides Cypress ordering part number that matches an existing IoT part number.

Table 2. Mapping Table for Part Number between Broadcom and Cypress

| Broadcom Part Number | Cypress Part Number |
|---|---|
| BCM20730 | CYW20730 |
| BCM43903 | CYW43903 |
| BCM43907 | CYW43907 |
| BCM43909 | CYW43909 |

## 3 IoT Resources

Cypress 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. Cypress 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 Cypress Support Community website (http://community.cypress.com/).

## 4 Before You Begin

It is recommended that the users of this application note request the correct WICED-SDK package from Cypress's Customer Support Portal (CSP). If necessary, contact your Sales or Engineering support representative.

## 5 OTP Programming Preparation

The CYW4390X OTP memory is programed using remote WL over a UART connection. Remote WL is a Windows application that sends WL commands (for example, `ciswrite`, `otpraw`, etc.) over the serial connection to the `mfg_test` WICED™ application that is running on the CYW4390X ACPU. The ACPU forwards the commands to the WLAN firmware, which then programs the OTP.

The procedure details are specified below:

1. Connect the CYW4390X WICED board to a Windows PC with an FTDI USB cable then verify that UART communication is successful.
2. Compile and download the mfg_test WICED application (default is Wiced-SDK\apps\test\mfg_test) to the CYW4390X WICED board using the following target build string. Modify the build string below to match your board name. For example, for BCM943909WCD1_3 board use the following build string.

```
$Wiced-SDK> make test.mfg_test-ThreadX-NetX_Duo-BCM943909WCD1_3 download run
```

3. Power cycle the CYW4390X WICED board.
4. From the Windows PC console, change the directory to `$Wiced-SDK\libraries\test\wl_tool`, where the remote WL Windows application (`wl43909_B0.exe`) is present.
5. Execute the WL command to print the WLAN firmware version (`ver`) using the remote WL Windows application (`wl43909_B0.exe --serial <serial_port>`).

**Note:** The <serial_port> number will differ from one computer to another.

***Example:***

```
$Wiced-SDK\libraries\test\wl_tool> wl43909_B0.exe --serial 6 ver

7.16 RC99.19

wl0: Jun 18 2015 10:58:40 version 7.15.168 (TOB) (r521252 WLTEST) FWID 01-3da9c1dd
```

If successful, the WLAN firmware version will be printed as shown in the example above. This confirms that the remote WL and `mfg_test` communication infrastructure is working as expected.

**Note:** If another serial terminal application is open on the same COM port, WL will fail to open.

The remote WL application setup can be used to execute any WL command supported by the WLAN firmware.

## 6 Programming Secure Boot, Secure Flash, and SECURE_BIT

**Note:** Programming secure boot and secure flash is usually not required during the early development phase.

This section covers OTP programming details related to secure boot and secure flash.

Either secure boot or secure flash, or both, can be implemented with some flexibility in terms of using different keys as outlined in Table 3. For instance, the RSA authentication method can be used for secure boot, and HMAC_SHA256 authentication can be used for secure flash. After choosing the authentication options, simply follow the steps required to support those options and discard anything that is not required or is not applicable.

Table 3. Encryption and Authentication Key Options

| Keys | Secure Boot | Secure Flash |
|---|---|---|
| ENCRYPTION | 0: No encryption<br>1: AES-CBC-128 | 0: No encryption<br>1: AES-CBC-128 |
| SIGNATURE (Authentication) | 0: No signature<br>2: HMAC_SHA256 (256-bit)<br>3: RSA | 0: No signature<br>2: HMAC_SHA256 (256-bit) |

## 6.1 Secure Boot

During boot-up, the second stage bootloader is decrypted and authenticated by the ROM bootloader. The ROM boot loader supports the following algorithms

- Encryption: AES128-CBC
- Authentication: HMAC_SHA256 or RSA (2048-bit)

## 6.2 Secure Flash

- Flash Partitions can be marked as secure during build time, during which the contents of the partitions is signed and encrypted before being programmed to the flash. The contents of the partitions are decrypted and authenticated when they are accessed during runtime.
- By default, Secure Flash ensures that the following flash partitions are secure:
  - User application
  - File System
  - Device Configuration Table (DCT)
  - OTA2 (Over the air upgrade version2) application
  - OTA2 Failsafe application
  - Factory Reset application
- Secure flash supports AES128-CBC for encryption and HMAC_SHA256 for authentication. A 256 bit HMC_SHA256 digest is stored at the end of each flash sector.

## 6.3 Enable Secure Boot

**Note:** During development, Secure Boot can be tested without programming the keys in OTP. By default, OTP is all 0s, and if `KEYS=<keys_dir>` is not specified in the WICED build string, NULL (all 0s) keys are used.

1. Set up the programming environment using WICED-SDK.
2. Establish the encryption and authentication method based on key options shown in Table 3 on page 3.
3. Generate appropriate keys as described in Generating Secure Keys on page 4.
4. Write secure keys (AES128-CBC, HMAC_SHA256, RSA, and so forth) by using the examples shown in Writing Secure Keys on page 7.
5. Determine the secure options to use from the list below and then enable these options using the examples shown Enabling Secure Options on page 6.
   – BOOT_OPTION.BOOT_SECURE: Should be set to 1.
   – BOOT_OPTION.USE_HW_CRYPTO: Recommended: 1 (HW Crypto).

     **Note:** In HW Crypto, the second stage Bootloader size is limited to 16 KB.

   – BOOT_OPTION.ENCRYPTION_METHOD: Recommended: 1 (AES128-CBC).
   – BOOT_OPTION.SIGNATURE_METHOD: Recommended 2 HMAC_SHA256 or 3 RSA

   In this step, program only row 292/290. No other row should be programmed.
6. Build and download the WICED APP with secure boot enabled.

- To select HMAC_SHA256, build with SECURE_BOOT=1 KEYS=<keys_dir> appended to WICED build string.
- To select RSA, build with SECURE_BOOT=1 SECURE_BOOT_AUTH=RSA KEYS=<keys_dir> appended to WICED build string.

***Example:***

```
test.console-BCM943909WCD1_3 download SECURE_BOOT=1 SECURE_BOOT_AUTH=RSA KEYS=<keys_dir>
```

For additional examples, see Secure Boot and Secure Flash Build String Examples on page 9.

## 6.4 Enable Secure Flash

**Note:** During Development, Secure Flash can be tested without programming the keys in OTP. By default, OTP is all 0s, and if `KEYS=<keys_dir>` is not specified in the WICED build string, by default NULL (all 0s) keys are used.

1. Set up the programming environment using WICED-SDK. Secure Flash uses AES128-CBC for encryption and SHA256-HMAC for authentication.
2. Generate the appropriate keys as described in Generating Secure Keys on page 4. Secure flash and secure boot share the same OTP space for keys.
3. Copy the keys to `WICED-SDK\platforms\<Platform>\keys\<keys_dir>\`

   **Note:** Secure flash does not support RSA authentication.
4. Program secure keys by using the examples shown in Writing Secure Keys on page 7. When using the same set of keys as those used for secure boot, there is no need to repeat the programming steps described here.
5. Build and download WICED APP with secure flash enabled
6. Build with SECURE_SFLASH=1 keys=<keys_dir> appended to the WICED build string.

For additional examples, see Secure Boot and Secure Flash Build String Examples on page 9.

## 6.5 Enable Secure Bit

It is important to prevent any external host from reading OTP-secured keys.

**Note:** Once SECURE_BIT is set to 1, OTP and flash cannot be programmed, so this must be the last task performed after the secure boot and secure flash procedures are completed.
When the secure bit (SECURE_BIT) is set to 1, JTAG, SDIO, USB and HSIC interfaces are disabled on the device. This prevents any external host from reading secure keys stored in OTP.

1. Set up the programming environment using WICED-SDK.
2. Set the SECURE_BIT address shown below. This is identical for both version B0 and version B1.

```
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 387 1 1
```

3. Download the target application. This must be done before step 4, otherwise the JTAG port will be disabled and the application download will not be possible.
4. Power cycle the device.

# 7 Generating Secure Keys

This section describes how to generate different secure keys for secure boot and secure flash.

## 7.1 HMAC_SHA256

1. Select any 32 byte key (see Table 3 on page 3). This key must be programmed in the OTP. See the OTP address map and programming examples shown in OTP Programming Commands on page 6.
2. Copy this key to WICED source code as:

```
WICED-SDK\WICED\platform\MCU\BCM4390x\keys\<keys_dir>\boot_sha.key
```

## 7.2 AES128-CBC

1. Select any 16 byte key (see Table 3 on page 3). This key must be programmed in the OTP. See the address map and programming examples shown in OTP Programming Commands on page 6.
2. Copy this key to WICED source code as:

   `WICED-SDK\WICED\platform\MCU\BCM4390x\keys\<keys_dir>\boot_aes.key`

## 7.3 RSA

1. Use `WICED-SDK\tools\secureboot\rsa\rsa_key_gen.py` to generate an RSA key pair and a hash of an RSA public key.
   – Usage example: `rsa_key_gen.py rsa_key`

**Note:** rsa_key_gen.py requires python, openssl, and ssh-keygen tools. In a Windows system, use Cygwin to install these tools.

2. Copy `rsa_key` and `rsa_key.n` to `WICED-SDK\WICED\platform\MCU\BCM4390x\keys\<keys_dir>\`.
3. Program `rsa_key.n.hash` to OTP. See the OTP address map and programming examples shown in OTP Programming Commands on page 6.

# 8 OTP Address Maps for Secure Boot

Table 4 and Table 5 show all OTP locations that must be programmed for both B0 and B1 chip versions. Table 5 on page 5 shows boot option bit definitions.

Table 4. OTP Address Map For Secure Boot

| Start offset (16-bit Words) | Start offset (Bits) | End offset (Bits) | Number of Bits | Name | Description |
|---|---|---|---|---|---|
| 290 | 4640 | 4671 | 32 | BOOT_OPTION_REDUNDANT | Redundant copy of boot options. See Table 5 on page 5 for definitions. |
| 292 | 4672 | 4703 | 32 | BOOT_OPTION | See Table 5 on page 5 for definitions. |
| 274 | 4384 | 4511 | 128 | AES_KEY_REDUNDANT | Redundant copy of AES128-CBC |
| 282 | 4512 | 4639 | 128 | AES_KEY | AES128-CBC key |
| 242 | 3872 | 4127 | 256 | SHA_KEY_REDUNDANT | Redundant copy of HMAC_SHA256 |
| 258 | 4128 | 4383 | 256 | SHA_KEY | HMAC_SHA256 symmetric key |
| 210 | 3360 | 3615 | 256 | RSA Public key hash | HMAC_SHA256 symmetric key |
| 226 | 3616 | 3871 | 256 | RSA public key hash (redundant) | Redundant copy of HMAC_SHA256 |
| NA | 387 | 387 | 1 | SECURE_BIT | • 1: Disables JTAG, SDIO, USB, and HSIC.<br>• 0: Default. However, once 1 is programmed, this cannot be reset to 0. |

Table 5. Boot Option Bit Definitions

| Bit Number | Boot Option | Description |
|---|---|---|
| 0 | BOOT_OPTION.BOOT_SECURE | • 0: Non secure boot<br>• 1: Secure boot |
| 1 | BOOT_OPTION.USE_HW_CRYPTO | • 0: Use SW algorithm<br>• 1: Use HW Crypto engine |
| 2 | BOOT_OPTION.ENCRYPTION_METHOD | • 0: No encryption<br>• 1: AES128-CBC encryption |
| 3:4 | BOOT_OPTION.SIGNATURE_METHOD | • 0: No signature<br>• 2: HMAC_SHA256<br>• 3: RSA (2048-bit) |

# 9 OTP Programming Commands

***Example:***

Using the remote wl tool to program OTP contents through the UART interface, build, download and run the mfgtest app using: `test.mfg_test-<platform> download run`. From the host PC, use the remote-wl tool: WICED-SDK\libraries\test\wl_tool\wl43909_B0.exe.

```
.\wl43909_B0.exe -serial <serial port num> otpraw <otp bit offset> <number of bits> <value>
```

When the argument <value> is omitted, the command reads the specified number of bits, starting at the offset.

***Example:***

```
\wl43909_B0.exe -serial 5 otpraw 0 1 returns the bit value at offset 0.

\wl43909_B0.exe -serial 5 otpraw 0 1 1 writes 1 to offset 0.

\wl43909_B0.exe -serial 5 otpraw 3 8 returns the value of 8 bits, starting at offset 3.

\wl43909_B0.exe -serial 5 otpraw 24 4 0xa writes the nibble such that LSB goes to bit-offset
24, MSB in 27.

\wl43909_B0.exe -serial 5 otpraw 24 8 0xba writes the byte at byte-address 3 (bit-offset 24).

\wl43909_B0.exe -serial 5 otpraw 24 16 0x12ba writes 0xba at byte-address 3, and 0x12 at byte-
address 4.
```

The wl otpdump command can be used to output contents of the entire OTP content to the console.

```
.\wl43909_B0.exe --serial <serial_port_num> otpdump
```

## 9.1 Enabling Secure Options

### 9.1.1 Enable Secure Boot

```
/* BOOT_OPTION.BOOT_SECURE */

# .\wl43909_B0.exe --serial <serial_port_num>  otpraw 4672 1 1


/* BOOT_OPTION_REDUNDANT.BOOT_SECURE */

# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4640 1 1
```

### 9.1.2 Enable HW Crypto

```
/* BOOT_OPTION.USE_HW_CRYPTO */

# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4673 1 1


/* BOOT_OPTION.USE_HW_CRYPTO_REDUNDANT */

# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4641 1 1
```

### 9.1.3 Enable AES128-CBC Encryption

```
/* BOOT_OPTION.ENCRYPTION_ALGO */

# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4674 1 1


/* BOOT_OPTION_REDUNDANT.ENCRYPTION_ALGO */

# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4642 1 1
```

### 9.1.4 Enable HMAC_SHA256 Authentication (Signature)

```
/* BOOT_OPTION.AUTH_ALGO */
# .\wl43909_B0.exe --serial <serial_port_num>   otpraw 4675 1 0
# .\wl43909_B0.exe --serial <serial_port_num>   otpraw 4676 1 1

/* BOOT_OPTION_REDUNDANT.AUTH_ALGO */
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4643 1 0
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4644 1 1
```

### 9.1.5 Enable RSA Authentication

```
/* BOOT_OPTION.AUTH_ALGO */
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4675 1 1
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4676 1 1

/* BOOT_OPTION.AUTH_ALGO */
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4643 1 1
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4644 1 1
```

## 9.2 Writing Secure Keys

### 9.2.1 Writing HMAC_SHA256 Key

This example assumes that the HMAC key is as follows:

Text: BroadcomTestBootBroadcomTestBoot

```
ASCII   42 72 6F 61 64 63 6F 6D 54 65 73 74 42 6F 6F 74 42 72 6F 61 64 63 6F 6D 54 65 73 74
42 6F 6F 74
```

Which should be written in 32-bit words as follows:

```
616f7242 aorB
6D6F6364 mocd
74736554 tseT
746F6F42 tooB
616f7242 aorB
6D6F6364 mocd
74736554 tseT
746F6F42 tooB

/* HMAC-SHA256_KEY */
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4128 32 0x616f7242
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4160 32 0x6d6f6364
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4192 32 0x74736554
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4224 32 0x746f6f42
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4256 32 0x616f7242
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4288 32 0x6d6f6364
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4320 32 0x74736554
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4352 32 0x746f6f42

/* HMAC-SHA256_REDUNDANT */
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 3872 32 0x616f7242
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 3904 32 0x616f7242
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 3936 32 0x6d6f6364
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 3968 32 0x74736554
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4000 32 0x746f6f42
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4032 32 0x616f7242
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4064 32 0x6d6f6364
# .\wl43909_B0.exe --serial <serial_port_num> otpraw 4096 32 0x74736554
```

### 9.2.2 Writing AES-CBC-128 Key

This example assumes that the AES-CBC-128 key is as follows:

Text: BcmEncryptionKey

```
ASCII: 42 63 6D 45 6E 63 72 79 70 74 69 6F 6E 4B 65 79

/* AES-128-CBC Key */
.\wl43909_B0.exe --serial <serial_port_num> otpraw 4512 32 0x456d6342
.\wl43909_B0.exe --serial <serial_port_num> otpraw 4544 32 0x7972636e
.\wl43909_B0.exe --serial <serial_port_num> otpraw 4576 32 0x6f697470
.\wl43909_B0.exe --serial <serial_port_num> otpraw 4608 32 0x79654b6e
/* AES-128-CBC Key Redundant */
.\wl43909_B0.exe --serial <serial_port_num> otpraw 4384 32 0x456d6342
.\wl43909_B0.exe --serial <serial_port_num> otpraw 4416 32 0x7972636e
.\wl43909_B0.exe --serial <serial_port_num> otpraw 4448 32 0x6f697470
.\wl43909_B0.exe --serial <serial_port_num> otpraw 4480 32 0x79654b6e
```

### 9.2.3 Writing RSA Public Key Hash

This example assumes that the RSA public key (<key_name>.n.hash generated by rsa_key_gen.py) is:

```
#hexdump -C <key_name>.n.hash
00000000 4f 0d 59 0c 32 d6 5b 0d 39 f0 7b ee ba 7e 84 3d |O.Y.2.[.9.{..~.=|
00000010 0f 3c da 8d c9 17 d4 07 5f c0 ac ad 59 03 3e e8 |.<......_...Y.>.|
/* RSA Public Key HASH */
.\wl43909_B0.exe --serial <serial_port_num> otpraw 3616 32 0x0c590d4f
.\wl43909_B0.exe --serial <serial_port_num> otpraw 3648 32 0x0d5bd632
.\wl43909_B0.exe --serial <serial_port_num> otpraw 3680 32 0xee7bf039
.\wl43909_B0.exe --serial <serial_port_num> otpraw 3712 32 0x3d847eba
.\wl43909_B0.exe --serial <serial_port_num> otpraw 3744 32 0x8dda3c0f
.\wl43909_B0.exe --serial <serial_port_num> otpraw 3776 32 0x07d417c9
```

```
.\wl43909_B0.exe --serial <serial_port_num> otpraw 3808 32 0xadacc05f

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3840 32 0xe83e0359

/* RSA Public Key HASH Redundant */

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3360 32 0x0c590d4f

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3392 32 0x0d5bd632

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3424 32 0xee7bf039

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3456 32 0x3d847eba

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3488 32 0x8dda3c0f

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3520 32 0x07d417c9

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3552 32 0xadacc05f

.\wl43909_B0.exe --serial <serial_port_num> otpraw 3584 32 0xe83e0359
```

## 10 Secure Boot and Secure Flash Build String Examples

```
/* Enable Secure boot, Use AES128-CBC and HMAC-SHA256  */

test.console-BCM943909WCD1_3 download SECURE_BOOT=1 KEYS=<keys_dir>

/* Enable Secure boot, Use AES128-CBC and RSA */

test.console-BCM943909WCD1_3 download SECURE_BOOT=1 SECURE_BOOT_AUTH=RSA KEYS=<keys_dir>

/* Enable Secure flash, Use AES128-CBC and HMAC-SHA256  */

test.console-BCM943909WCD1_3 download SECURE_SFLASH=1 KEYS=<keys_dir>

/* Enable Secure boot and Secure flash, For Secure boot use AES128-CBC and HMAC-SHA256, for
Secure Flash use AES128-CBC and HMAC-SHA256  */

test.console-BCM943909WCD1_3 download SECURE_BOOT=1 SECURE_SFLASH=1 KEYS=<keys_dir>

/* Enable Secure boot and Secure flash, For Secure boot use AES128-CBC and RSA, for Secure
Flash use AES128-CBC and HMAC-SHA256 */

test.console-BCM943909WCD1_3 download SECURE_BOOT=1  SECURE_BOOT_AUTH=RSA  SECURE_SFLASH=1
KEYS=<keys_dir>
```

## 11 References

The references in this section may be used in conjunction with this document.

**Note:** Cypress provides customer access to technical documentation and software through its Customer Support Portal (CSP) and Downloads and Support site.

For Cypress documents, replace the “xx” in the document number with the largest number available in the repository to ensure that you have the most current version of the document.

| Document (or Item) Name | Number | Source |
|---|---|---|
| OTP Programming and NVRAM Development | 4390X-AN10x | Cypress CSP |
| WICED™ IEEE 802.11 b/g/n SoC with an Embedded Applications Processor | 43907-DS10x-R | Cypress CSP |
| WICED™ IEEE 802.11 b/g/n SoC with an Embedded Applications Processor | 43909-DS10x-R | Cypress CSP |

## Document History Page

| **Document Title: AN214842 - CYW4390X: OTP Programming and Using Secure Boot and Secure Flash** | | | | |
|---|---|---|---|---|
| **Document Number: 002-14842** | | | | |
| **Rev.** | **ECN No.** | **Orig. of Change** | **Submission Date** | **Description of Change** |
| ** | – | – | 07/31/2015 | 4390X-AN200-R:<br>Initial release |
| *A | 5460329 | UTSV | 10/04/2016 | Updated in Cypress template |
| *B | 5699618 | VIPA | 12/19/2017 | Removed BCM4390x chip revision B0 specific details<br>Updated HWCrypto max size for 2nd stage bootloader 16K<br>Changed Secure flash key directory<br>Updated template |

# Worldwide Sales and Design Support

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturers' representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

### Products

| | |
|---|---|
| Arm® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Internet of Things | cypress.com/iot |
| Memory | cypress.com/memory |
| Microcontrollers | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| Power Management ICs | cypress.com/pmic |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless Connectivity | cypress.com/wireless |

### PSoC®Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

### Cypress Developer Community

Community | Projects | Video | Blogs | Training | Components

### Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709