Safeguard IP and device authenticity

IN THIS DOCUMENT

xCORE devices contain on-chip one-time programmable (OTP) memory that can be blown during or after device manufacture testing. You can program the xCORE AES Module into the OTP of a device, allowing programs to be stored encrypted on flash memory. This helps provide:

- **Secrecy**

  Encrypted programs are hard to reverse engineer.

- **Program Authenticity**

  The AES loader will not load programs that have been tampered with or other third-party programs.

- **Device Authenticity**

  Programs encrypted with your secret keys cannot be cloned using xCORE devices provided by third parties.

Once the AES Module is programmed, the OTP security bits are blown, transforming each tile into a “secure island” in which all computation, memory access, I/O and communication are under exclusive control of the code running on the tile. When set, these bits:

- force boot from OTP to prevent bypassing,
- disable JTAG access to the tile to prevent the keys being read, and
- stop further writes to OTP to prevent updates.

The AES module provides a strong level of protection from casual hackers. It is important to realize, however, that there is no such thing as unbreakable security and there is nothing you can do to completely prevent a determined and resourceful attacker from extracting your keys.

Publication Date: 2013/11/11 Document Number: X6225B

XMOS © 2013, All Rights Reserved

1 The xCORE AES module

The xCORE AES Module authenticates and decrypts programs from SPI flash devices. When programmed into a device, it enables the following secure boot procedure, as illustrated in Figure 1.

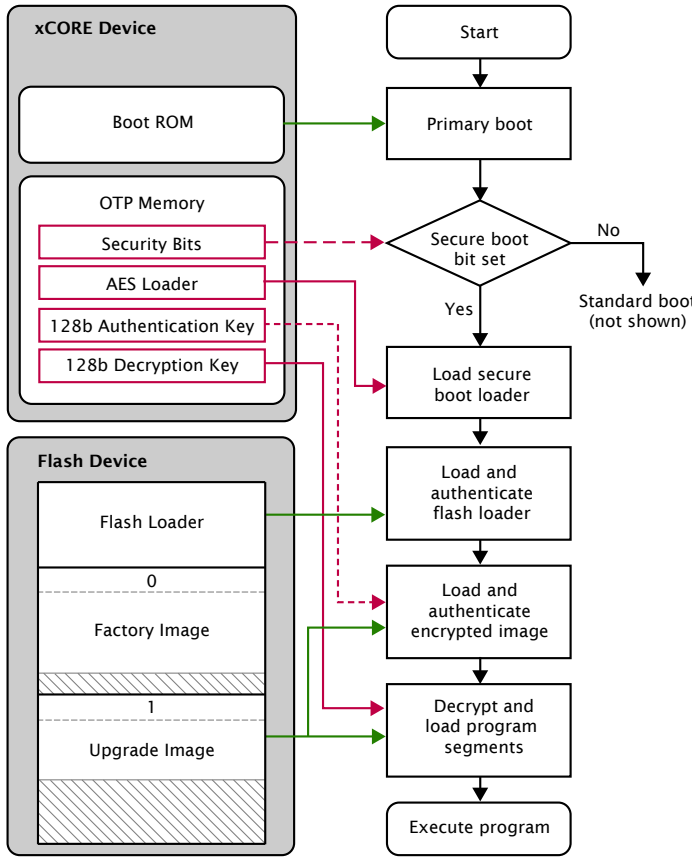

Figure 1: Secure boot procedure used with the AES Module

1. The device loads the primary bootloader from its ROM, which detects that the secure boot bit is set in the OTP and then loads and executes the AES Module from OTP.
2. The AES Module loads the flash loader into RAM over SPI.
3. The AES Module authenticates the flash loader using the CMAC-AES-128 algorithm and the 128-bit authentication key. If authentication fails, boot is halted.
4. The AES Module places the authentication key and decryption key in registers and jumps to the flash loader.

The flash loader performs the following operations:

1. Selects the image with the highest number that validates against its CRC.
2. Authenticates the selected image header using its CMAC tag and authentication key. If the authentication fails, boot is halted.
3. Authenticates, decrypts and loads the table of program/data segments into memory. If any images fail authentication, the boot halts.
4. Starts executing the program.

For multi-node systems, the AES Module is written to the OTP of one tile, and a secure boot-from-xCONNECT Link protocol is programmed into all other tiles.

2 Develop with the AES module enabled

You can activate the AES Module at any time during development or device manufacture. In a development environment, you can activate the module but leave the security bits unset, enabling:

- XFLASH to use the device to load programs onto flash memory,
- XGDB to debug programs running on the device, and
- XBURN to later write additional OTP bits to protect the device.

In a production environment, you must protect the device to prevent the keys from being read out of OTP by the end user.

To program the AES Module into the xCORE device on your development board, start the command-line tools (see XM-000950-PC) and enter the following commands:

1. `xburn --genkey` *keyfile*

   XBURN writes two random 128-bit keys to *keyfile*. The first line is the authentication key, the second line the decryption key.

   The keys are generated using the open-source library crypto++. If you prefer, you can create this file and provide your own keys.

2. `xburn -l`

   XBURN prints an enumerated list of all JTAG adapters connected to your PC and the devices on each JTAG chain, in the form:

   ```
   ID - NAME (ADAPTER-SERIAL-NUMBER)
   ```

3. `xburn --id` *ID* `--lock` *keyfile* `--target-file` *target*`.xn --enable-jtag --disable-master-lock`

   XBURN writes the AES Module and security keys to the OTP memory of the target device and sets its secure boot bit. The SPI ports used for booting are taken from the XN file (see XM-000929-PC).

To encrypt your program and write it to flash memory, enter the command:

- `xflash --id` *ID* *bin*`.xe --key` *keyfile*

To protect the xCORE device, preventing any further development, enter the command:

- `xburn --id` *ID* `--target-file` *target*`.xn --disable-jtag --lock` *keyfile*

3 Production flash programming flow

In production manufacturing environments, the same program is typically programmed into multiple SPI devices.

To generate an encrypted image in the xCORE flash format, start the command-line tools (see XM-000950-PC) and enter the following command:

- `xflash` *prog*`.xe -key` *keyfile* `-o` *image-file*

This image can be programmed directly into flash memory using a third-party flash programmer, or it can be programmed using XFLASH (via an xCORE device). To program using XFLASH, enter the following commands:

1. `xflash -l`

   XFLASH prints an enumerated list of all JTAG adapters connected to your PC and the devices on each JTAG chain, in the form:

   `ID - NAME (ADAPTER-SERIAL-NUMBER)`

2. `xflash --id` *ID* `--target-file` *platform*`.xn --write-all` *image-file*

   XFLASH generates an image in the xCORE flash format that contains a first stage loader and factory image comprising the binary and data segments from your compiled program. It then writes this image to flash memory using the xCORE device.

The XN file must define an SPI flash device and specify the four ports of the xCORE device to which it is connected (see XM-000929-PC).

4 Production OTP programming flow

In production manufacturing environments, the same keys are typically programmed into multiple xCORE devices.

To generate an image that contains the AES Module and security keys to be written to the OTP, start the command-line tools (see XM-000950-PC) and enter the following commands:

1. `xburn --genkey` *keyfile*

   XBURN writes two random 128-bit keys to keyfile. The first line is the authentication key, the second line the decryption key.

   The keys are generated using the open-source library crypto++. If you prefer, you can create this file and provide your own keys.

2. `xburn --target-file` *target*`.xn --lock` *keyfile* `-o` *aes-image*`.otp`

   XBURN generates an image that contains the AES Module, security keys and the values for the security bits.

The image contains the keys and must be kept secret.

To write the AES Module and security bits to a device in a production environment, enter the following commands:

1. `xburn -l`

   XBURN prints an enumerated list of all JTAG adapters connected to the host and the devices on each JTAG chain, in the form:

   ```
   ID - NAME (ADAPTER-SERIAL-NUMBER)
   ```

2. `xburn --id` *ID* `--target-file` *target*`.xn` *aes-image*`.otp`

   XBURN loads a program onto the device that writes the AES Module and security keys to the OTP, and sets its secure boot bits. XBURN returns 0 for success or non-zero for failure.

XMOS®

Copyright © 2013, All Rights Reserved.

Xmos Ltd. is the owner or licensee of this design, code, or Information (collectively, the “Information”) and is providing it to you “AS IS” with no warranty of any kind, express or implied and shall have no liability in relation to its use. Xmos Ltd. makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.