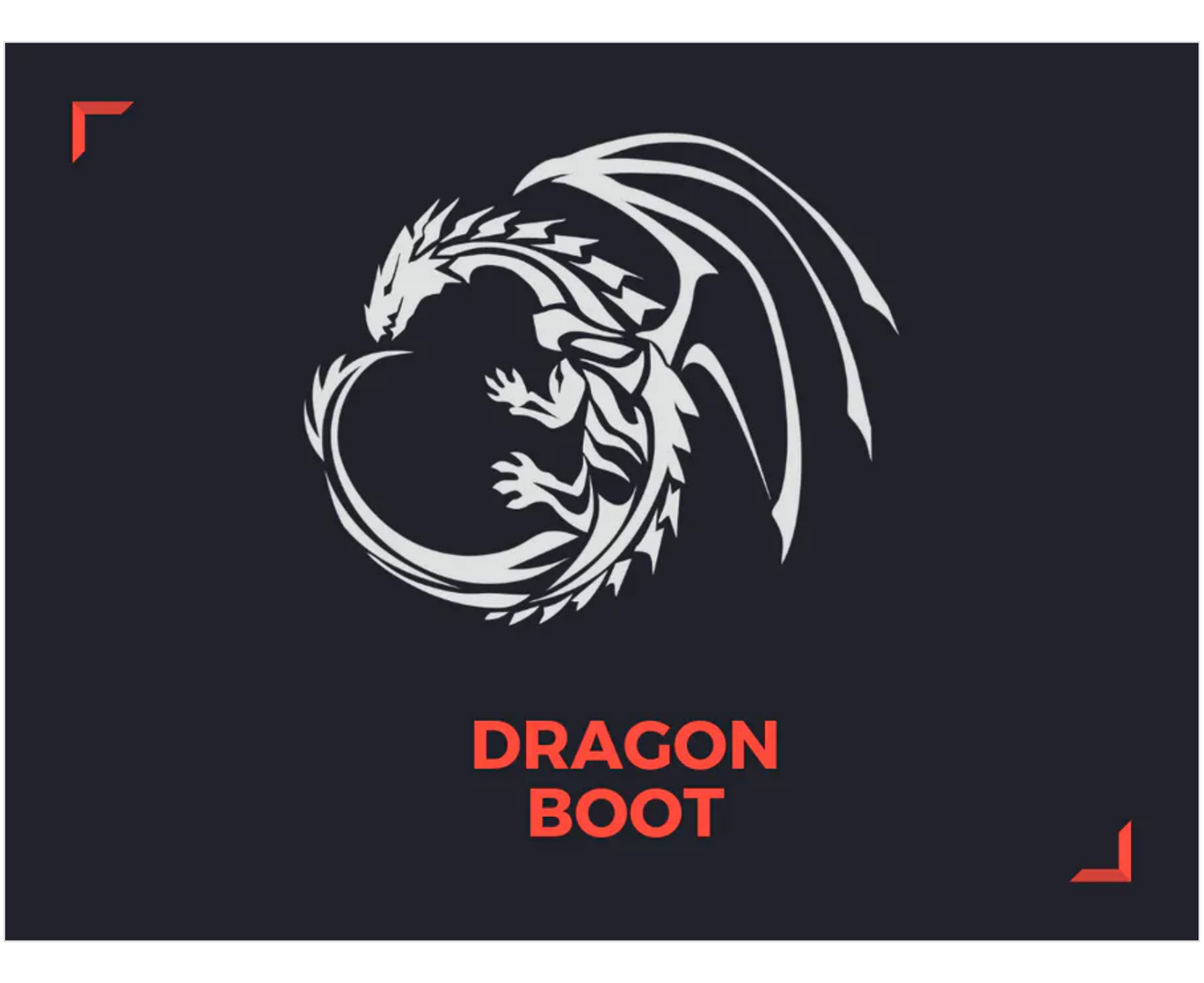
Embedded Systems Lab, Sirish Krishnan
Published July 5, 2019 © GPL3+

DragonBoot

A bootloader with an interactive console and secure boot feature for the Tiva TM4C123G series microcontroller.

Intermediate · Full instructions provided · 2 hours · 777



Things used in this project

Hardware components

Component	Quantity
Texas Instruments EK-TM4C123GXL TM4C Tiva LaunchPad	× 1
USB-A to Micro-USB Cable	× 1

Software apps and online services

Texas Instruments TivaWare C Series v2.1.4
Includes royalty-free libraries (Peripheral, USB, Graphics, Sensor) and kit-/peripheral-specific code examples for all TM4C devices

LM4Tools
Tools to enable multi-platform development on the TI Stellaris Launchpad boards

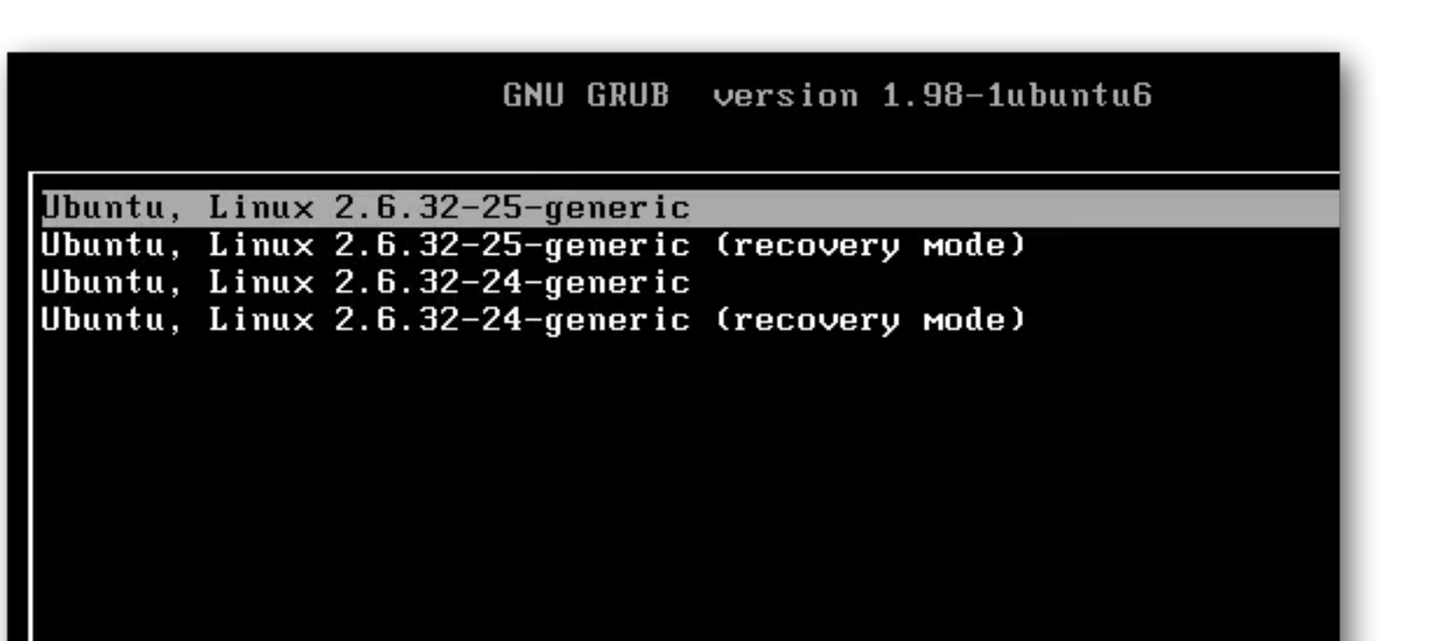
Story

Video walkthrough for DragonBoot

What's a bootloader?

A bootloader, also called a boot manager, is a small program that places the operating system (OS) of a computer into memory. Once the OS has been placed into memory, the processor runs the OS.

It's usually the first piece of code that runs on an embedded system to initialize the CPU and its peripherals before the OS takes over.



The GRUB bootloader menu

What's DragonBoot ?

DragonBoot is a bootloader that allows the user to flash multiple applications (to different memory partitions) and boot any one of them through a command-line interface (very much like the GRUB bootloader).

Flash and bootup of an application is done in a secure manner that involves the processes of digital signature validation & asymmetric key cryptography.

Motivation

Software that comes shipped along with an embedded system can be :

• Replaced / "upgraded" by a clever hacker to run his / her piece of software.

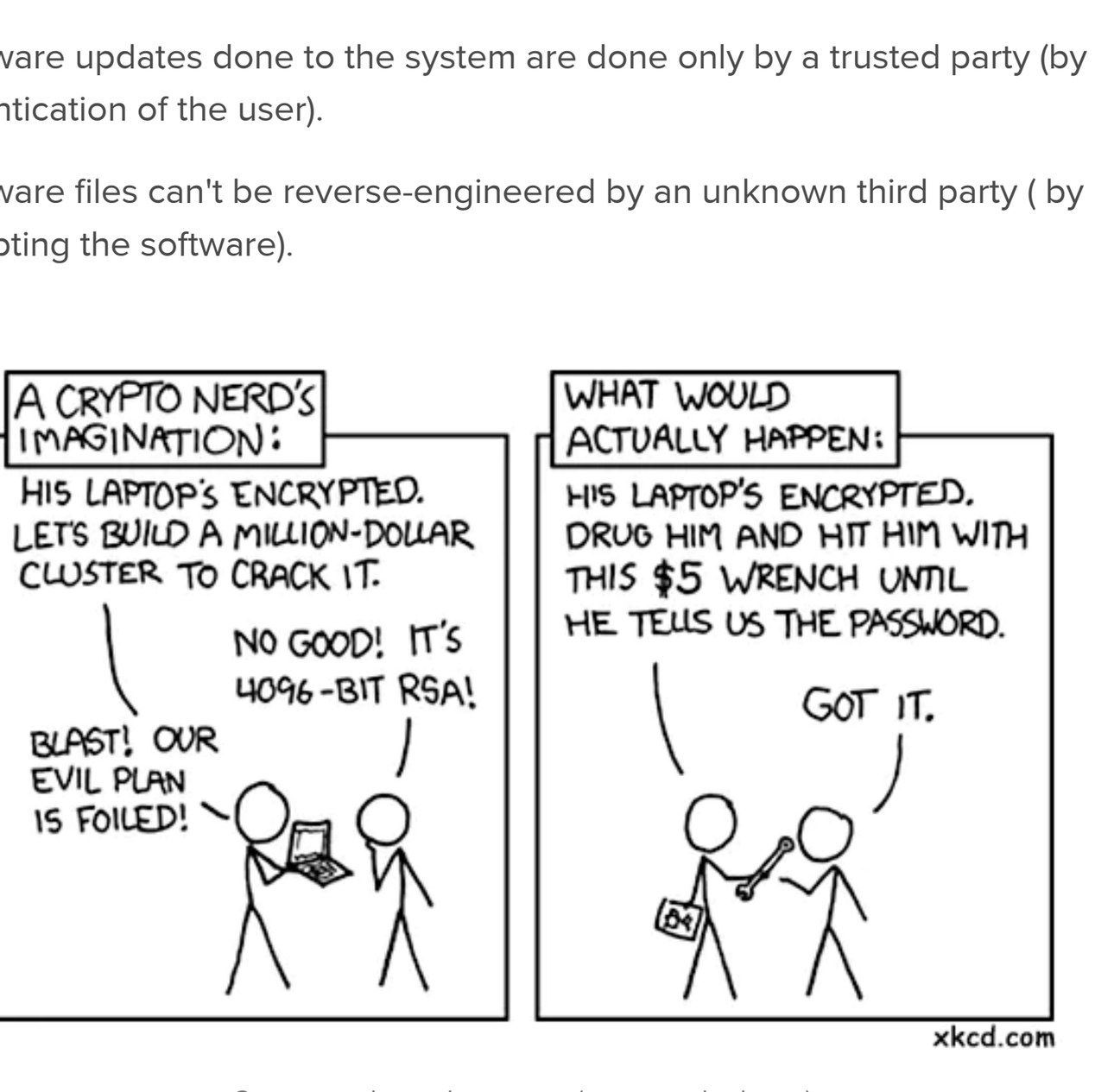
• Reverse-engineered if the file falls into an unknown third party's hands.

This could spoil the manufacturer's business or even pose a serious safety hazard (if somebody succeeds in downloading a malicious piece of code onto the embedded system).

Therefore, there's a need to ensure that :

• Software updates done to the system are done only by a trusted party (by authentication of the user).

• Software files can't be reverse-engineered by an unknown third party (by encrypting the software).



Cryptography to the rescue (source : xkcd.com)

How it works

DragonBoot is an interactive bootloader that allows software updates to happen only through a 2-step process of :

• Decrypting the software binary & calculating an SHA256 hash

• Verifying the authenticity of the user by performing verification of the digitally signed embedded SHA256 hash that is included with the binary & comparing the same with the computed value in the previous step

Authentication ensures that we're able to verify the identity of the sending party before attempting to upgrade the system. So only the intended party (i.e. the manufacturer) can upgrade the system. This is accomplished by implementing the idea of a digital signature.

Supplying the software binary in encrypted form ensures that nobody else can make sense out of it / reverse-engineer the same if it falls into their hands. Only the bootloader has the key to decrypt the encrypted software binary.

Features

• Low memory footprint (roughly 65Kbytes after stripping headers) for the rich feature set, making this bootloader an attractive option for memory constrained microcontrollers.

• Interactive shell (thanks to the Microrl library by Helius) with command auto-complete feature (via TAB key).

• File transfer via XMODEM protocol (thanks to Georges Menie). Minicom has native support for file transfer via XMODEM and can be used off-the-shelf.

• Wide range of cryptographic functions (RSA, AES, Elliptic Curve etc.) supported thanks to the microcontroller-friendly BearSSL library developed by Thomas Pornin.

• Software ring-buffer for asynchronous storage of high-speed UART data thanks to Anders Kalør.

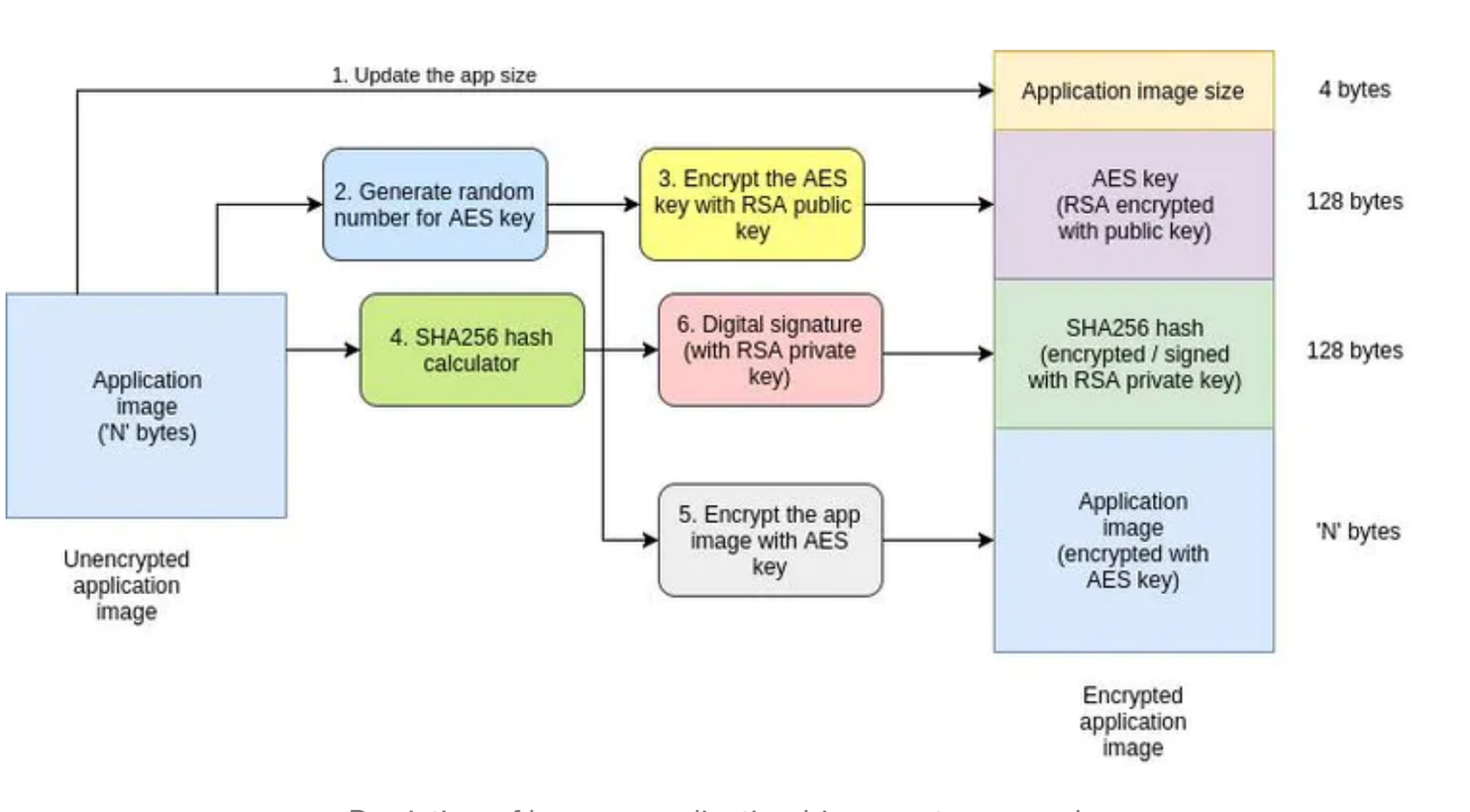
But...Wouldn't the interactive shell break the system's security ?!

Yeah, duh ! Adding a user-friendly console would definitely make the bootloader insecure. However, the aim of the project is simply to demonstrate some interesting features in a nice, conducive environment !

The reader is encouraged to hack the code / customize it to suit his / her needs. That's one of the advantages of DragonBoot : the code has been modularized and designed in a way to facilitate easy customization as per one's needs. Indeed, one could even port it to other platforms (there's even a platform porting guide in the included documentation).

Design & Implementation

Secure upgrade process



Depiction of how an application binary gets secured

The first step in the secure upgrade process, as depicted in the above diagram is to generate an encrypted version of the application that would be used for software upgrade. The below steps need to be followed for the same :

1. Take the input application binary (stripped off all headers & debug info).

2. Create a new binary, update its header with the actual size of the application.

3. Generate a random number for use as a 128-bit AES key,

4. Encrypt the AES key using an RSA public key and store it in the final application's header.

5. Calculate SHA256 hash of the application binary (used for integrity check later by the DragonBoot bootloader after receiving the same).

6. Encrypt the application using the AES key generated earlier.

7. Digitally sign the SHA256 hash value (using a different RSA key - private version) computed earlier and store it in the header. This is used for authentication later by DragonBoot.

Once the encrypted binary has been transferred to DragonBoot via UART, the following steps are followed by the bootloader to decrypt & authenticate the image before storing it in flash :

1. Decrypt the AES key by using the RSA private key counterpart of the public key used for encryption at the source earlier.

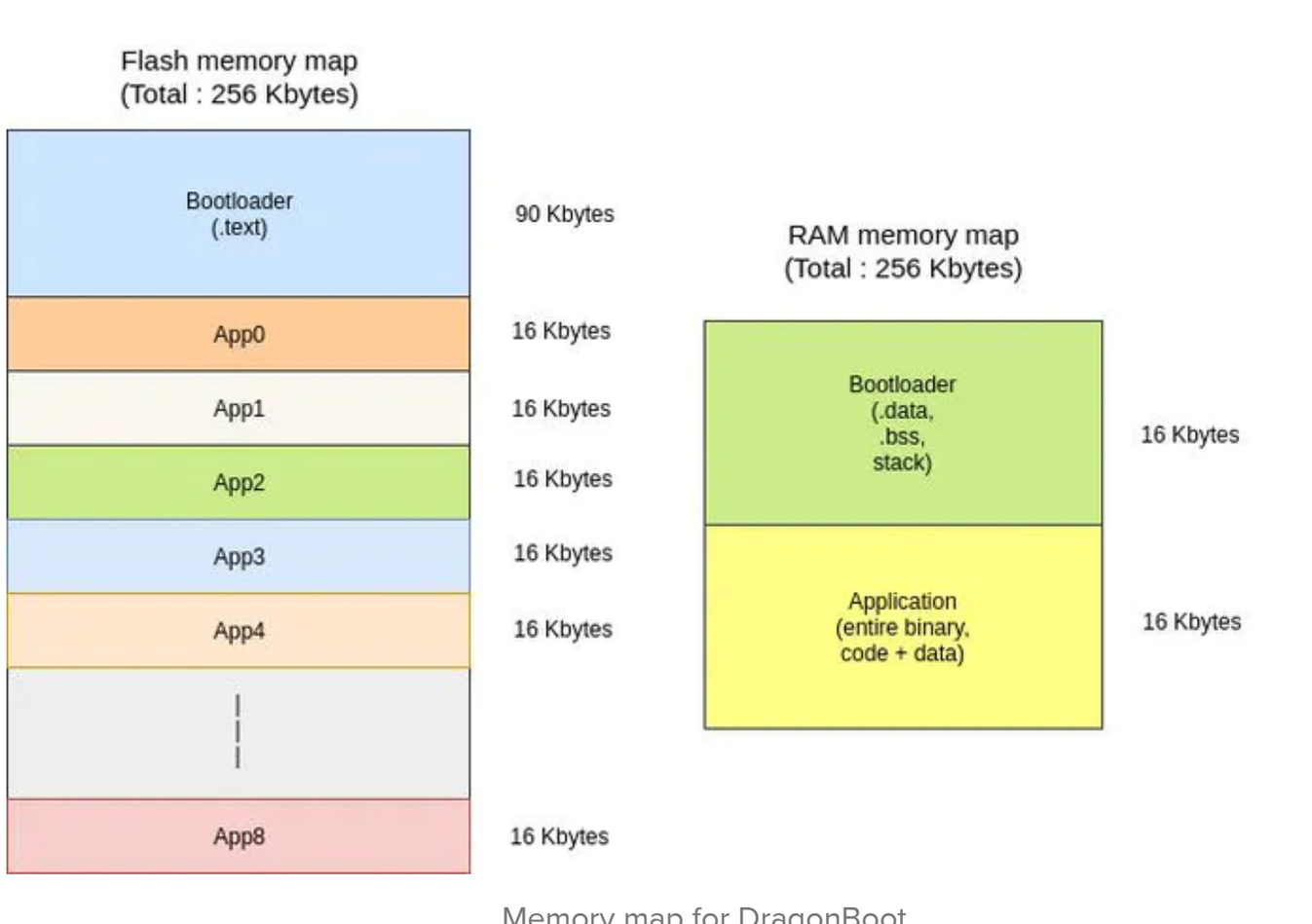
2. Use this AES key to decrypt the application binary.

3. Compute the SHA256 hash of the decrypted binary.

4. Perform signature validation of the signed SHA256 hash sent by the source (by using the RSA public key counterpart).

5. Next, compare the transmitted hash value against the computed value to verify that the image has not been tampered and to confirm that the authorized entity has sent the application image. So the hash & digital signature serves a dual purpose of integrity check as well as authentication.

Memory map



Memory map for DragonBoot

• The memory map is depicted on the figure above.

• The flash memory has been divided into 9 equal sized partitions, 1 for storing each application. 90 KBytes has been reserved for the bootloader code area; that is a super-conservative estimate.

• RAM, 32 KBytes total has been divided equally between the bootloader & application.

• The memory layout as depicted above is a super-conservative layout, only used for demonstration purposes. The application to be booted up is stored initially in one of the partitions designed for it in the flash memory.

Application boot sequence

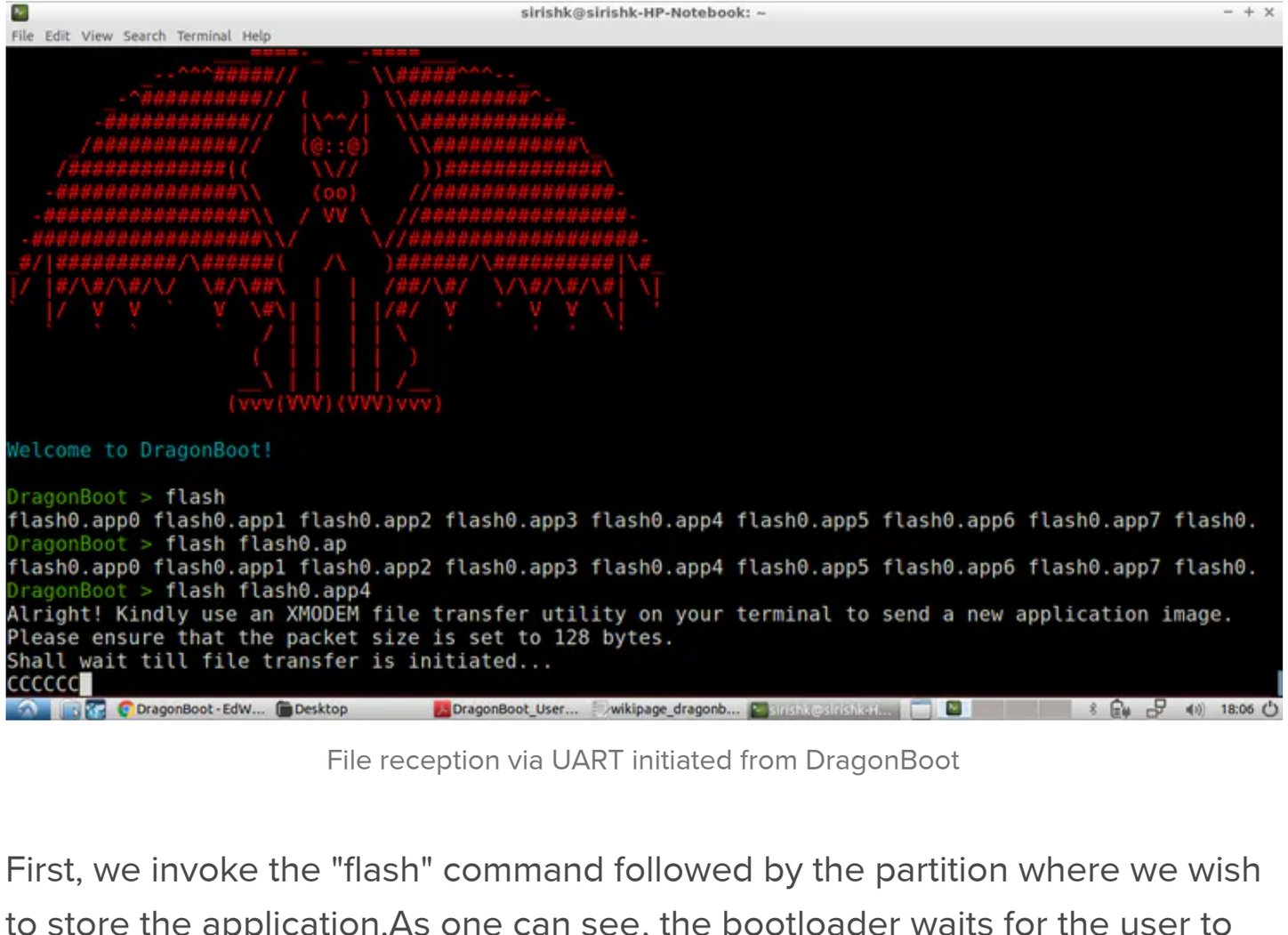
• Copy the decrypted application from flash to RAM location 0x20004000 (16K from start of RAM address 0x20000000).

• Change the NVIC vector table base address to 0x20004000 (that is the starting address of the application where the ARM Vector table is stored).

• Load the Stack pointer with 4 bytes from memory address [0x20004000].

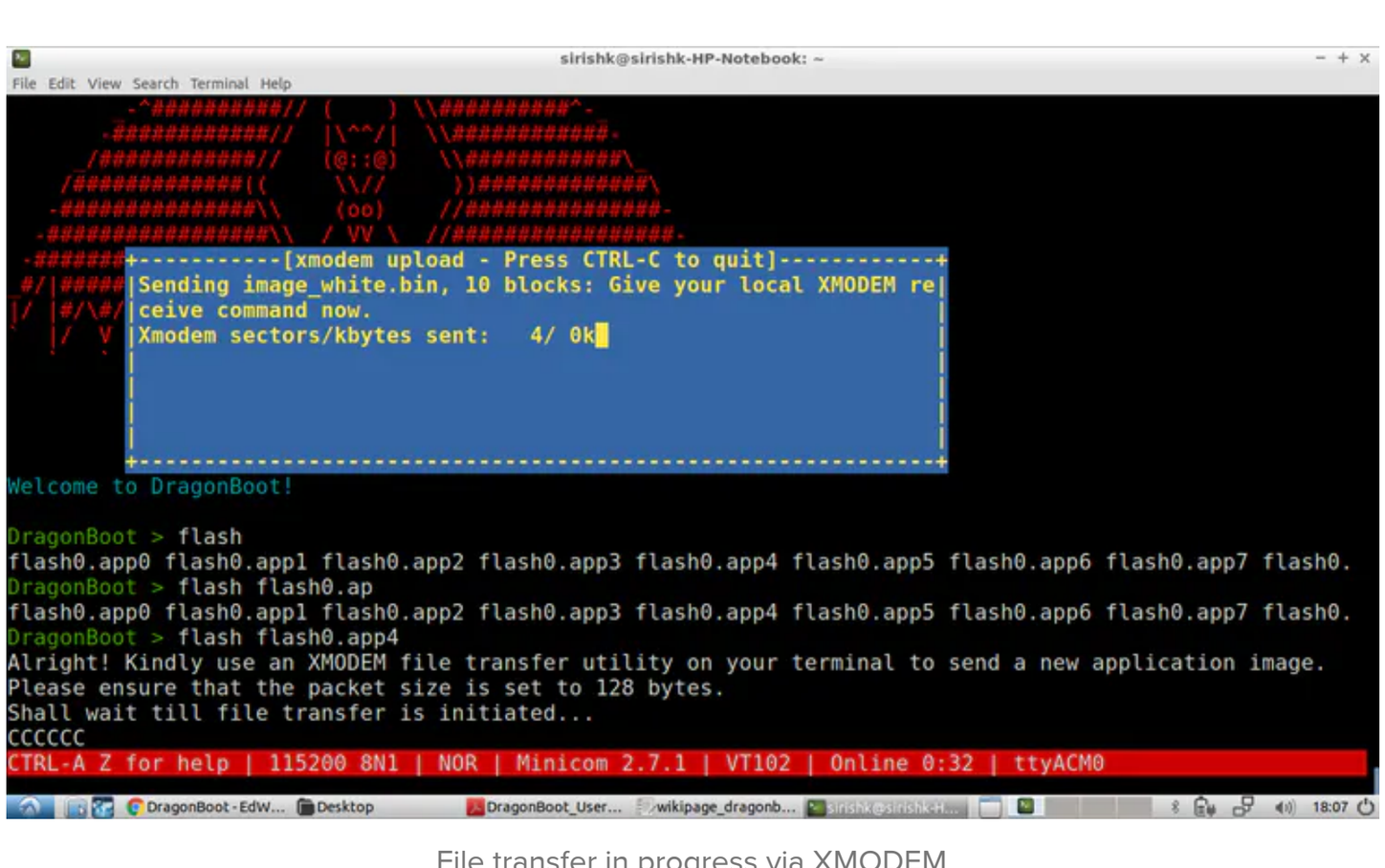
• Load the Program Counter with 4 bytes from memory address [0x20004004] to boot up the application.

DragonBoot in action!



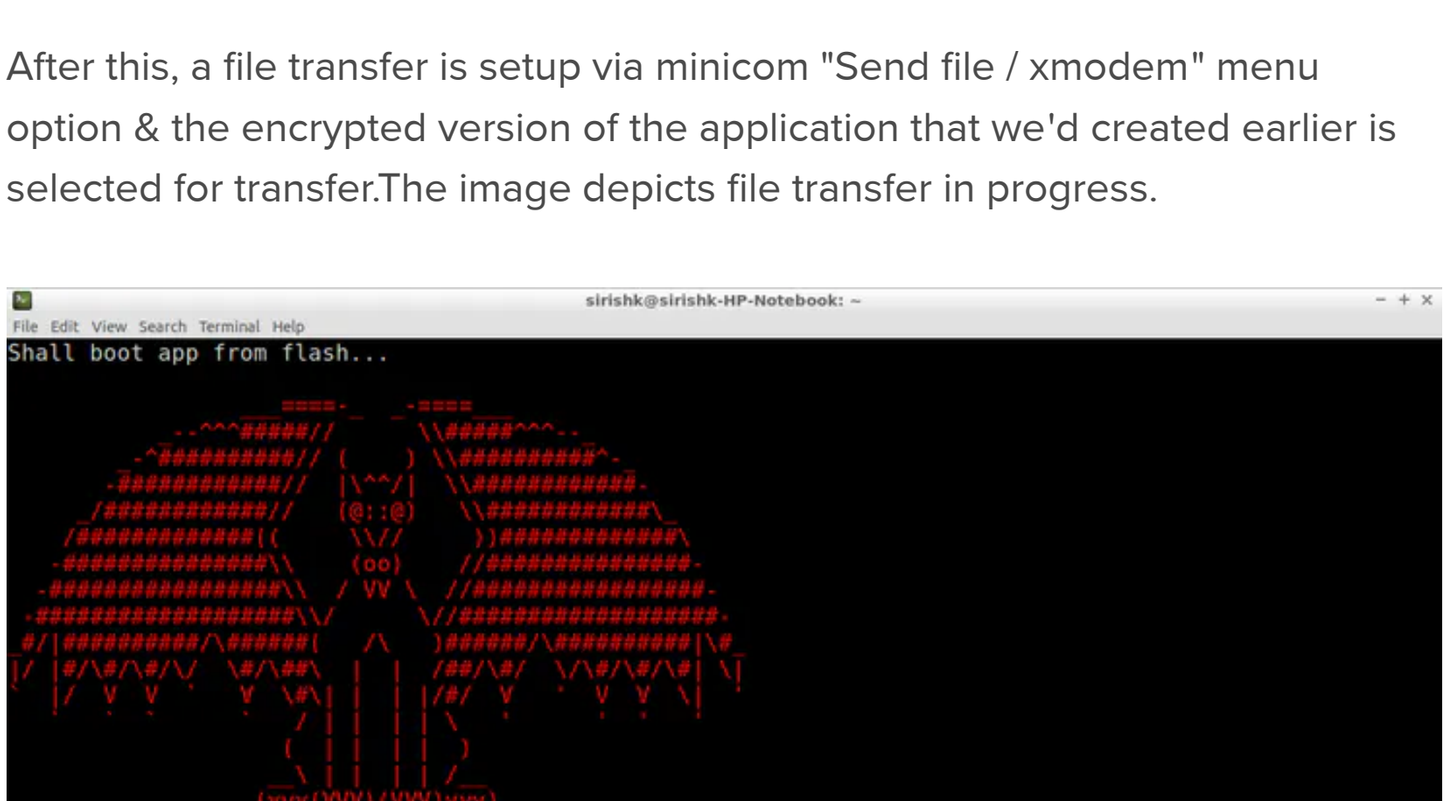
File reception via UART initiated from DragonBoot

First, we invoke the "flash" command followed by the partition where we wish to store the application.As one can see, the bootloader waits for the user to start file transfer via XMODEM protocol.



File transfer in progress via XMODEM

After this, a file transfer is setup via minicom "Send file / xmodem" menu option & the encrypted version of the application that we'd created earlier is selected for transfer.The image depicts file transfer in progress.



Booting an app from flash partition

Finally, we boot the flashed application binary by using the "boot" command followed by the name of the partition from where we wish to run an application.

Further details

For further details, you may access the below document:

DragonBoot User & Developer's Guide

For source code, you may visit the following page:

DragonBoot Source Code

For instructions on how to build and test the software:

Build & Test Instructions

Cheers, have fun!

Code

DragonBoot source code
Contains source code for building DragonBoot along with Cryptograf (encryption tool used to encrypt applications before they can be flashed onto the board) as well as useful documentation.

Credits

Embedded Systems Lab
8 projects • 5 followers
Follow Contact

Sirish Krishnan
3 projects • 1 follower
Follow Contact

Thanks to **J Shankarappa**, **Eugene Samoylov aka Helius**, **Georges Menie**, **Thomas Pornin**, and **Anders Kaloer**.

Comments

Please log in or sign up to comment.