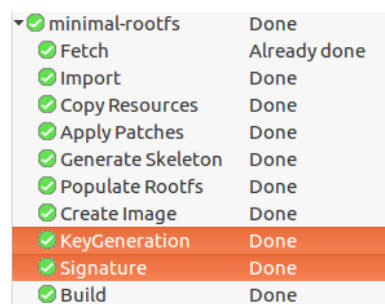


Secured embedded Linux using Ac6 tools

Introduction

With the increasing deployment of embedded systems in critical applications, such technologies face many security challenges. Securing an image was never an easy task in the image generation procedure, because it needs specific knowledge about both the hardware part of the embedded chip and the software used to secure and sign the image. In order to respond to the various kinds of attacks on electronics and computer systems, AC6 has focused on security with its latest product with the aim of developing a mechanism to create secure embedded Linux distributions.

Simplifying the generation of a secured embedded Linux is the main goal of System Workbench for Linux (SW4Linux). In addition to creating, configuring and building an embedded Linux image using a graphical user interface, now it is also possible to sign the deployable image and generate the needed signing keys.



minimal-rootfs	Done
Fetch	Already done
Import	Done
Copy Resources	Done
Apply Patches	Done
Generate Skeleton	Done
Populate Rootfs	Done
Create Image	Done
KeyGeneration	Done
Signature	Done
Build	Done

Figure 1-SW4Linux: Signing image build steps added to the rootfs

Secure Boot

Secure Boot is a process in which the images and boot code of a Linux operating system are verified and authenticated by the hardware before being used in the boot process. To do this, the hardware must be configured beforehand to enable image authentication. It ensures that only crypto-authenticated software (bootloader, kernel, etc.) will be able to run on a given target.

At power-on, the ROM (Read Only Memory) code is the first code executed by the target, which performs the initialization (like programming of clocks, batteries, interrupt configuration...). The ROM code is also responsible for loading the SPL (Secondary Program Loader) which in turn initializes the hardware components and prepares the memory space for loading the SBL (Secondary Bootloader) into the RAM (Random Access Memory). And the SBL loads the kernel and the Root Files System.

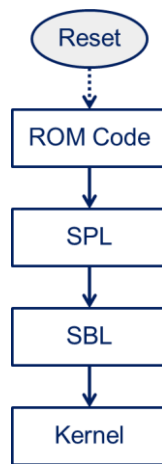


Figure 2-Normal boot sequence

For a secure boot, each element is signed by a key, which is a parameter used as input in a cryptographic operation and based on an encryption algorithm. The signature is verified at each step of the startup sequence. This creates a chain of trust from power-on to the actual operating system startup, thus ensuring the integrity of the system.

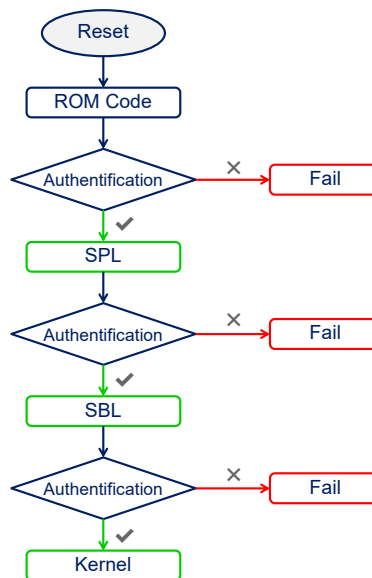


Figure 3-Secure boot sequence

The secure boot sequence is described as follows: the ROM code loads the Bootloader from memory. It verifies that the hash of the public key contained in the image matches the one stored on the trusted root (which is a physical element where the authentication information is stored). Then, it decrypts the signature of the image using the previously verified public key in order to extract the "hash". It compares the calculated hash with the extracted hash, if the two values are identical that means that the image is trusted and can be executed. Otherwise the startup process is aborted and the image will not boot.

To setup such a mechanism, several preparatory steps are necessary. The first step is key generation. There are several tools available, usually manufacturers provide their own key generation and signature tools. A private and public key pair is required for the signature. The private key is used for signing images and the public key is used for signature verification. The private key is one of the essential links in the Secure Boot process

There are two methods for signing: image signature and configuration signature. The configuration signature has the advantage of signing all the "hashes" contained in the configuration. This avoids certain types of attacks when signing only the images and not the configuration.

The last step in securing the system is to store the public key in a secured and protected memory area. The main solution to ensure the integrity of a system from the boot phase is the use of a physical component: an OTP (One-Time Programmable) memory. The latter allows a safe storage of the public key that represents the root of trust.

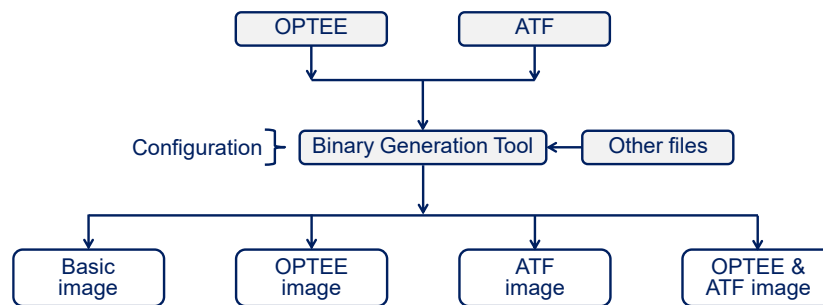


Figure 4-Generating the secured image

To boot a secure embedded device, several firmware and software are required. To store all images in a single binary file, the FIT (Flattened Image Tree) image structure is used. The FIT image structure is like a kind of container that can hold several image types. The structure of the FIT file is described as follows:

The bootloader like for example U-Boot, which is open source and both a first stage and second-stage bootloader. It runs a command-line interface on a serial port, and can read and write flash memory, download files from the serial port or network in order to load and boot the kernel.

Arm Trusted Firmware (ATF) is a trusted software reference implementation for the ARMv7-A and ARMv8-A processors. It includes a secure monitor running at Exception Level 3 (EL3). The ARMv8-A architecture implements multiple levels of privilege to access system and processor resources. Each exception level is numbered (from 0 to 3) and higher privilege levels have higher values. Exception level 3 represents the highest level of privilege reserved for low-level software and secure code.

OP-TEE is a secure runtime environment designed as a companion to an unsecured Linux kernel running on ARM using TrustZone technology as a hardware isolation mechanism. OP-TEE allows the coexistence of systems offering different levels of security on the same architecture. OP-TEE distinguishes between two worlds, a secure world in which services that use sensitive data run, and an unsecured world used for common applications. One of the key elements of a TEE is the Secure Monitor, a secure program for switching between the two worlds.

There are also some other files specific to each board and configuration, for example the binaries needed to configure and initialize the external DDR RAM (DDR firmware), HDMI or Display port firmware.

At startup, after loading the FIT image, the public key is read, and the FIT image is extracted into the memory and the hash of the configuration recovered from the FIT image is calculated. The signature is decrypted using the public key to extract the hash. If the two hashes are identical, then U-Boot will be able to load the images and thus guarantee their authentication.

Key generation

The first step for signing Linux images is to generate the keys, each vendor has its own tool for key generation each with different options. These tools need to generate the private and public keys in the supported formats in addition to calculating their hashes.

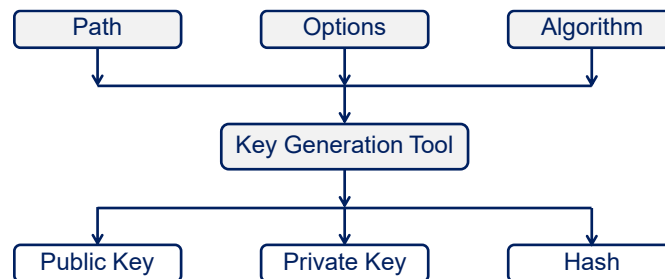


Figure 5-Key generation

Another tool will use these keys to encrypt the image hash using the private key and then to create a header that contains both the signature and the public key. The keys (in a specific format) should be flashed into the one-time programmable memory.

Flashing the keys require in some cases a special tool or it can be done using the bootloader.

Signing embedded Linux with SW4Linux

As you can see, securing a Linux system consists on many sophisticated phases, where each has its own complexity. Therefore, SW4Linux focused on simplifying each phase by adding the required features that let securing embedded Linux distribution easier than ever, in which it is possible to select these settings using graphical user interface, where underlying each option resides the complex operations in a transparent way to the user. This optional feature can be found in the root file system project's properties menu.

Key generation and image signing are additional build steps that are now available besides the classical steps. These options are configured by default to support the different architectures currently available in the market.

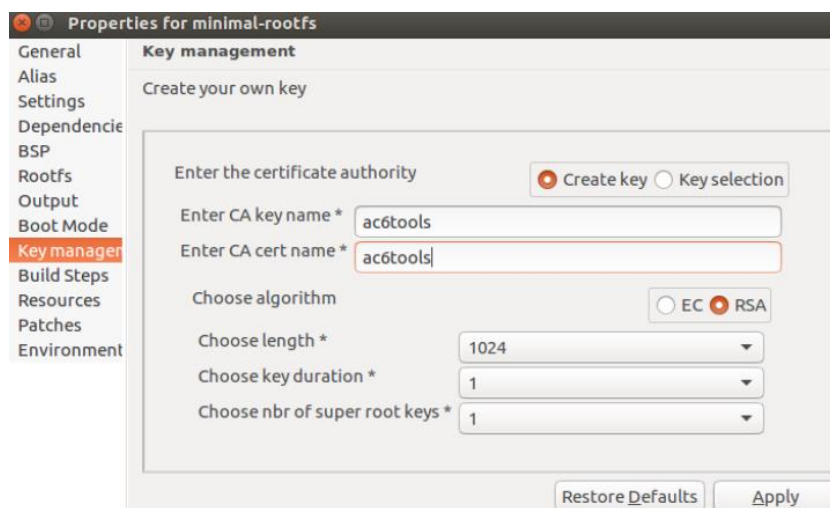


Figure 6-SW4Linux : Key management options

The user can use the preconfigured option, where the recommended options are set, and then he can simply select other options if he wants.

Key management feature is uniform across all platforms. The user has a wide choice of key generation. The key generation internal script is a set of Openssl commands. The “Key management” figure shows the configuration menu of the interface. It can be based on an existing key or new key generation, the user must select the key name, its algorithm and other needed parameters. Then a key structure, containing a set of certificates, private keys and public keys, is generated and used automatically.

To choose the startup mode for a target, a graphical interface is available that lets the user chooses the configuration, either Arm Trusted Firmware (ATF), OP-TEE or both, In addition to the startup signature (signed or unsigned) as shown in the “Boot mode” figure.

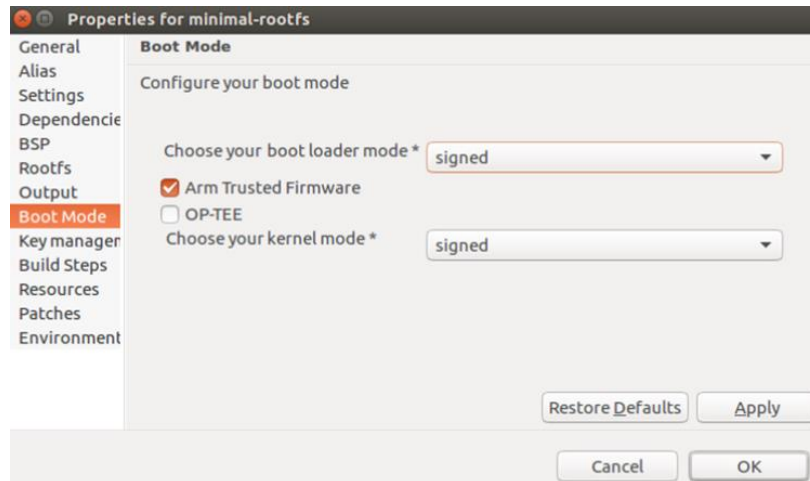


Figure 7-SW4Linux:Boot mode options

Arm Trusted Firmware (ATF) is a trusted software implementation for ARMV7-A and ARMV8-A processors. It includes a secure monitor running at exception level 3. The ATF package can be created with two different options. The first option is to use the built-in secure monitor from Arm Trusted Firmware. The secure monitor is the secure software that allows switching between the "Secure" and "Non-Secure" world. The second is to use the OP-TEE secure monitor to manage the switch between the two worlds. The difference between these two options lies in the configurations and build options.

Conclusion

In this white paper, we were able to explore the different phases of a secured boot, it is essential to ensure the security of the system, but secure boot has always been a sophisticated step in the image creation; it relies a lot on both the underlying hardware and software. It has always needed an embedded security specialist in order to create and configure the system. System Workbench for Linux is a build system tool that not only provides customizable embedded Linux distribution build but also hardening, securing and signing the deployable image. The key feature is simplicity where everything is based on an eclipse project with a graphical user interface that only shows needed options and then performs all the complex work internally.