地 平 线
Horizon Robotics

X3J3 Secure Feature Introduction

V1.0

2021-02

Important Notice and Disclaimer

Information in this document is provided solely to enable system and software implementers to use Horizon products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

All statements, information and recommendations in this document are provided “AS IS”. Horizon makes no warranty, representation or guarantee of any kind, express or implied, regarding the merchantability, fitness or suitability of its products for any particular purpose, and non-infringement of any third party intellectual property rights, nor does Horizon assume any liability arising out of the application or use of any product, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

“Typical” parameters that may be provided in Horizon datasheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Buyers and others who are developing systems that incorporate Horizon products (collectively, “Users”) understand and agree that Users shall remain responsible for using independent analysis, evaluation and judgment in designing their applications and that Users have full and exclusive responsibility to assure the safety of Users’ applications and compliance of their applications (and of all Horizon products used in or for Users’ applications) with all applicable regulations, laws and other applicable requirements.

User agrees to fully indemnify Horizon and its representatives against any claims, damages, costs, losses and/or liabilities arising out of User’s unauthorized application of Horizon products and non-compliance with any terms of this notice.

Horizon Robotics, Inc.

https://www.horizon.ai

Revision History

Time	Version	Details
2020.08	V0.5	Document Created
2020.10	V0.6	Add AVB, FDE, Crypto Engine
2020.11	V0.7	Revised
2020.12	V0.8	add process of packing bpu and uboot
2021.1	v0.9	Refine the uboot signature part
2021.2	v1.0	delete the process of signing uboot and bpu, move this content to “X3J3 System Software Security Manual”

Contents

1 Introduction

1.1 Purpose

This document serves primarily to introduce the Secure Boot flow and Encryption functionality supported on X3J3 platform. If you want to know how to use the Secure Boot, such as how to sign uboot image, please refer to "*X3J3 System software Security Manual*"

1.2 Terminology

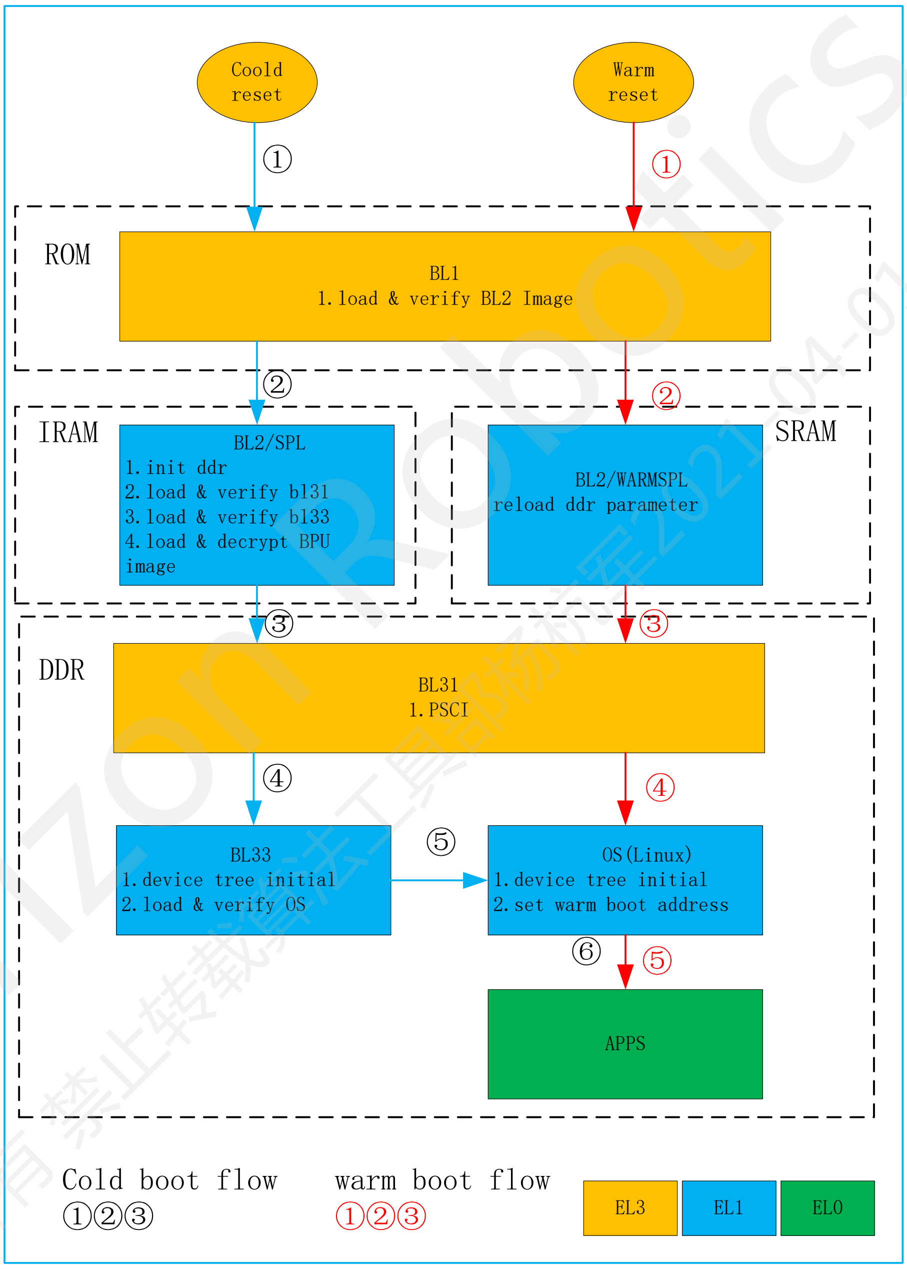
Abbr	Description
SoC	System on Chip
BL[x]	Boot Loader Stage [x]
SPL	Secondary Program Loader
BPU	Brain Process Unit
AES	Advanced Encryption Standard
IV	Initial Vector
SHA	Secure Hash Algorithm
AVB	Android Verified Boot
PSCI	Power State Coordination Interface
SMC	Secure Monitor Cal
CMA	Contiguous Memory Allocator
HBM	HoBot Model
PKA	Public Key Accelerator

1.3 Audience

This document aims at guiding developers and engineers who need to take advantage of the secure boot and encryption system. This document provides a general instruction and usage of the secure boot and encryption system. For detailed functionality designs, please refer to the corresponding design manual.

2 Secure Boot

2.1 Overall flow chart of system start-up

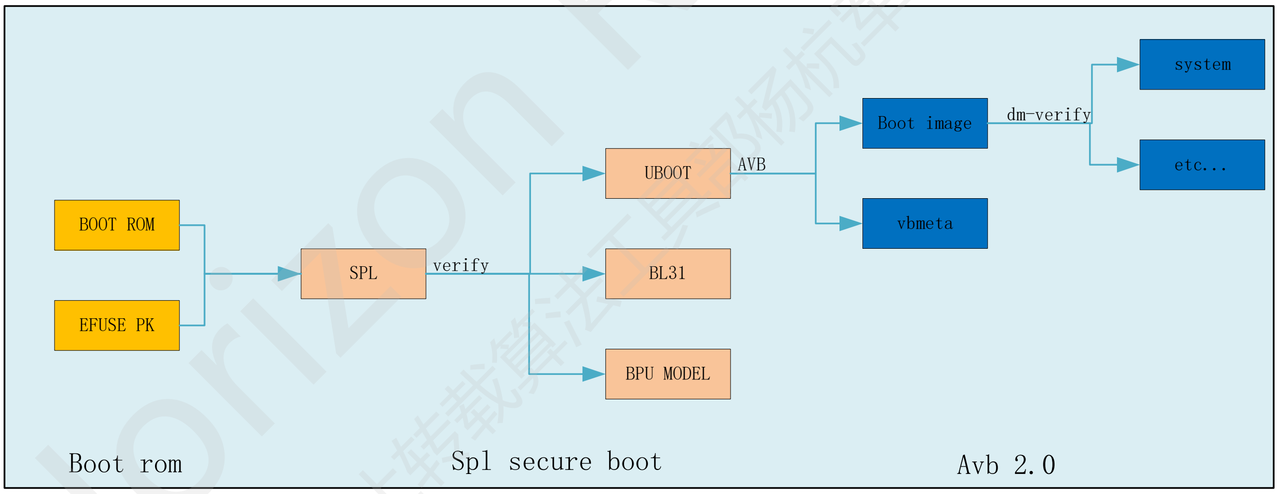


The system power-on process can be divided into two situations: one is cold boot, the other is warm boot. Warm boot specifically refers to the suspend/resume process; Other conditions, such as power-off and power-on start-up, reboot command start-up, etc. are all cold boot.

2.2 Process explanation

1. Jump to BL1 execution when Power On Reset, running at EL3
2. BL1 will jump to BL2 to execute, and the running state will switch to EL1. Depending on the power-on method, BL2 may be SPL or WARM SPL
3. BL2/SPL initializes DDR, loads the images of BL31, BL33 and BPU encryption model, and performs verification. WARM SPL only reload ddr parameters from sram to set ddr parameters. Then SPL jumps to BL31 and BL31 runs at EL3 state.
4. BL31 jumps to BL33 to continue with code boot mode, BL33 runs in EL1 state. BL31 jumps to OS with warm boot mode.
5. BL33 initializes the device tree, loads and verifies the OS, then starts the OS, and the OS runs in EL1 state
6. After the OS is started, it starts to run the user service process, and the user service runs in the EL0 state

In the overall start-up process, the secure boot of the system is checked down step by step in a hierarchical manner. The overall process is shown in the figure below



The verification process of secure boot is shown in the figure above. Bootrom verifies the BL2(SPL or WARM SPL) image. WARM SPL will jump to BL31 without verification; SPL will load and verify the uboot, bl31, bpu model, then uboot will verify the OS Kernel image (boot image, including ramdisk, dtb, etc.). Verifying the corresponding system partition through dm-verify is supported by OS.

In the above start-up verification process, verifying BL2 in bootrom and verifying BL31 in SPL are mandatory. Others such as verifying uboot and bpu model in SPL, verifying kernel in uboot, etc. are optional; customers can use eFUESE to configure it. For details, refer to the document "*X3J3 System software Security Manual*".

3 AVB

3.1 Overview

Android Verified Boot (AVB) is a secure boot flow implemented by android. X3J3 adopts the AVB2.0 boot flow for assuring end user the integrity of X3J3 Linux Kernel and root filesystem. For detailed description of AVB2.0, please refer to the AVB official site.

3.2 AVB Flow on X3J3

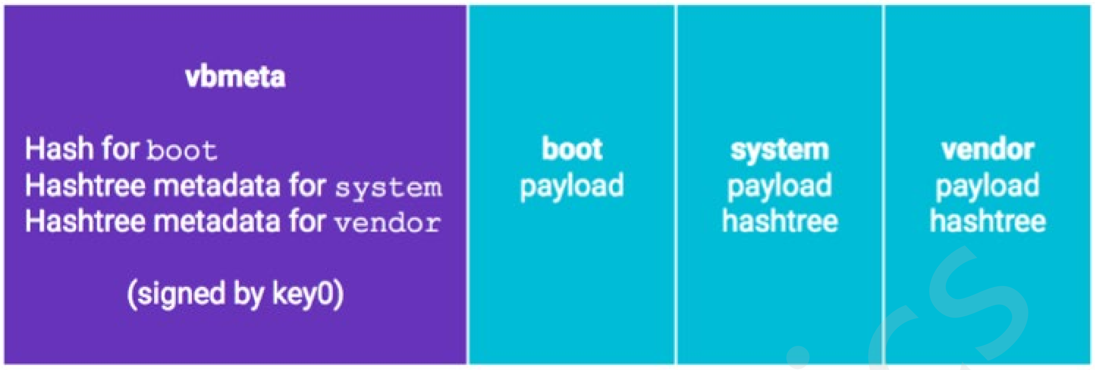
On X3J3 system, AVB is divided into two major phases. First phase is Kernel verification. Kernel verification is handled by UBoot, who will verify the Kernel being loaded has not been modified. The second phase is root filesystem verification, which will be handled collectively by UBoot and Kernel. For root filesystem verification, UBoot is responsible for providing verification information to Kernel through command line arguments (bootargs). The Kernel will then verify the root filesystem by dm-verity.

3.2.1 DM-Verity

DM-Verity is a block-oriented verification method based on the Device Mapper framework implemented in Linux Kernel. Device Mapper is a mapping mechanism between logical and physical devices. Under device mapper, user can easily customize the storage resources according to the different design specifications. DM-Verity utilizes a hash tree to describe the whole root filesystem image. DM-Verity also allows verification on the fly, which verifies each data block when loaded, instead of verifying the whole image at once providing much more performance than otherwise. For more detailed description of device mapper and DM-Verity, please refer to the Kernel Documentation and AVB official site.

3.3 Partition Design

For X3J3 platform, the AVB related partition consists of 3 major part: vbmeta, boot and system. Partition vbmeta stores the meta data describing boot and system. Partition boot stores Kernel image and dts. Partition System stores root filesystem.



As shown in the image above, currently X3J3 uses boot and system by default, user can add other payloads as desired.

3.4 Build Support

For the convenience of users, X3J3 platform embedded the AVB Image creation into the build process. For more information, please refer to the build scripts listed under build/tools/mkbootimg and build/tools/avbtools.

4 Encrypt Filesystem

4.1 Overview

X3J3 has full disk encryption (FDE) support implemented for filesystem encryption. X3J3 primarily use this functionality to protect partition userdata. The lowest version compatible with FDE is 2020.02.22.

Usage

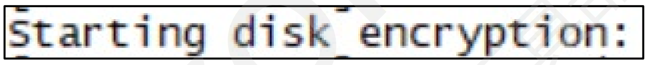
4.2 Image Generation

To use FDE, add "-p yes" in the build command to enable FDE support in the generated images.

4.3 Function Verification

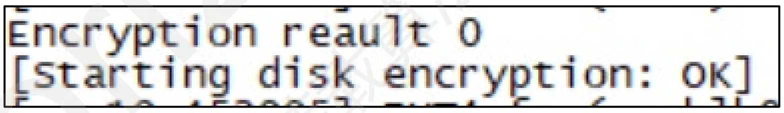
If the image is correctly built and burnt to the storage, when booting from that image, the following print should appear on screen:

The following print indicates the encryption has started:



```
Starting disk encryption:
```

The following print indicates that the encryption is successful:



```
Encryption reault 0
[Starting disk encryption: OK]
```

If [Starting disk encryption: FAIL] is shown instead of "OK", that means FDE function is not properly initialized. When the userdata partition is mounted under FDE, the sysfs node mounted will be shown as "/dev/mapper/userdata".

4.3.1 Mount FDE partition

To mount FDE partitions, a correct password must be provided. There are two types of passwords: files and user inputs.

4.3.1.1 Using File as Password

This is also the default option for FDE passwords. The default path of the password file on board is located under "/etc/init.d/fde_default.bin". If a new password file is desired, copy the new password file to "prebuilts/root/etc/init.d/" and rename it to "/fde_default.bin" or modify the password file entry in file "prebuilts/root/etc/fde_crypttab" and copy the desired new password file to "prebuilts/root/etc/init.d" and rebuild the image.

4.3.1.2 Using User Input as Password

To use user input as password, execute "user_cryptsetup" on board. For detailed usage, please refer to help manual of user_cryptsetup.

5 Crypto Engine

The crypto engine on X3J3 SOC support hardware acceleration of commonly used Cipher, Hash and PKA algoritms. There are three parts in the engine, a Cipher and Hash accelerator name SPAcc, a Public Key Accelerator(PKA) and a True Random Number Generator(TRNG). Each part has independent software stack.

In this chapter mainly introduce the supported features, software framework and performance issue of each part.

5.1 Supported format

The supported cipher and hash algorithms and key length are listed in below table.

Algorithm	key length	iv length
sha512	64	
sha256	32	
sha1	20	
hmac(sha512)	64	
hmac(sha256)	32	
hmac(sha1)	20	
ecb(des)	8	8
ecb(des3_ede)	24	8
ecb(aes)	16, 24, 32	16
ctr(aes)	16, 24, 32	16
cbc(des)	8	8
cbc(des3_ede)	24	8
cbc(aes)	16, 24, 32	16

Cipher and Hash supported algorithms

Kernel is also configured to support software implements of these algorithms. When crypto driver modules are not install, software algorithms with use used by default. Once driver modules installed, the algorithms will run on crypto engine.

RSA support 1024/2048/3072/4096 bits signature verfication.

TRNG can generate 16 Bytes or 32 Bytes length random number.

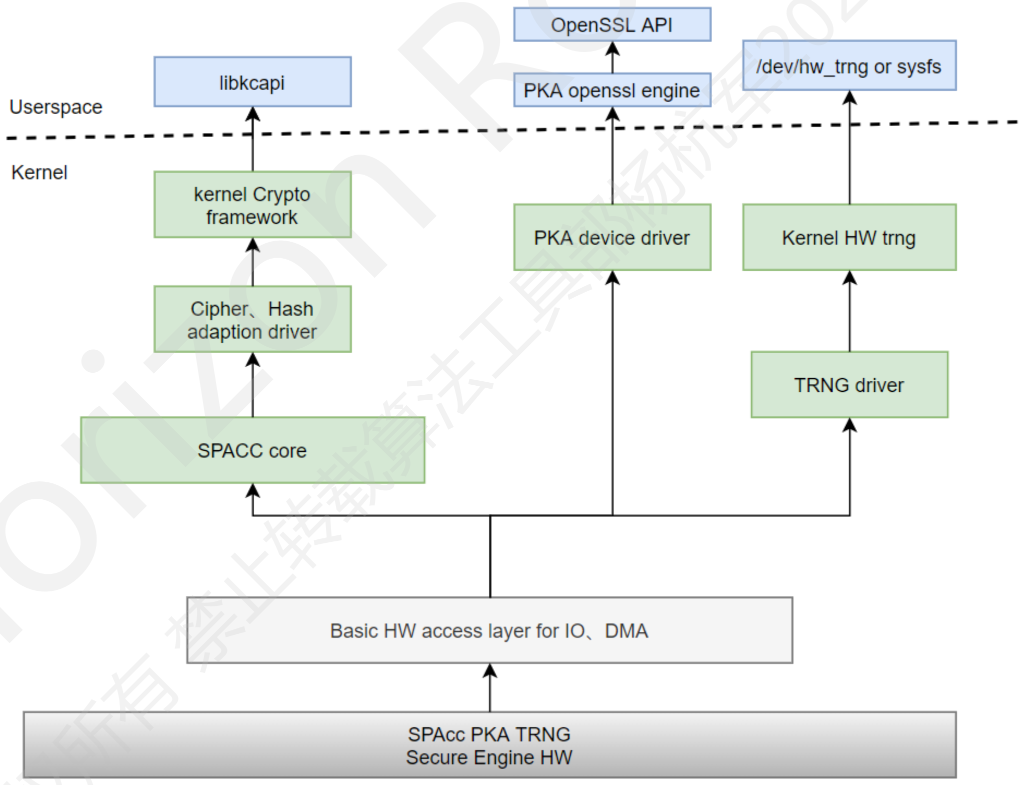
There is a limitation of SHA Hash algorithms. The hash result is not standard if the data length is no less than 64K bytes. So the digest message can't be compared across other platform or with softare generated digest.

5.2 Software Framework

Cipher and Hash drivers are implemented to adapt Linux kernel crypto framework, so that usespace program can use the HW accelerator via kernel socket interface AF_ALG. We recommend libkcapi in userspace programing. libkcapi is a full featured crypto library for using kernel supported crypto API.

The PKA module supports RSA public-key algorithm for asymmetric cipher and signature operations. This function should be used via OpenSSL API, and an openssl engine lib is implemented to access the HW driver.

In addition, there is an HW random number generator (TRNG) that can provide true random ramdom number. It can generate 16 Bytes or 32 Bytes ramdump numbers. User can get the random number via /dev/hw_trng or sysfs attributes.



Crypto Engine Software Architecture

For libkcapi usage, please refer to its offical website: https://www.chronox.de/libkcapi.html.

For openssl usage, please refer to openssl offical website: https://www.openssl.org/.

5.3 Performance issue

In our performance test, the HW accelerator drivers do not out perform the software algorithms implemention in Linux kernel significantly. Cipher is about 10% faster than software algorithms, Hash is about 90% faster. RSA is even 40% slower.

One reason is that ARM Cortex-A8 Linux kernel already has NEON optimizations on the AES and SHA algorithms, the performance increase significantly. Another reason is data tranfer from usespace to kernel space software stack limit the speed of data feeding to the HW accelerators..

Although the HW driver is not much faster than software algorithms, it can offload CPU usage. For Cipher and Hash, CPU usage is 5% on HW algorithms, and 20% on SW algorithm.