HASPOC
Secure Boot
Functional Specification

Oniteo AB, Jan Stenbecks torg 17, 164 40 Kista, Sweden
Ph. +46 8 4441360, E-mail info@t2data.com www.t2data.com

19 September 2016

Revisions

Revision Overview

Revision	Log Message	Author	Date
PA1	Migrated from impl_spec document	Hans Thorsen	2016-07-04
PA2	Tools chapter updated	Hans Thorsen	2016-07-04

Table of Content

Table of Contents

Functional description 1
- Goals . . . 1
- Hardware initialization . . . 2
- Verified boot sequence . . . 2
- Power state Control Interface . . . 2
- Services . . . 2
- Assumptions . . . 2
- The SFRs tracing to the TSFIs . . . 3
  - TSFI_POWERON_COLD . . . 3
  - TSFI-Storage . . . 4

Bibliography 5

Tools 6
- Managing source . . . 7
- Entry point into high level . . . 7
- bios_high.h . . . 7

Chapter 1 .

Functional description

Secure Boot is divided into three functional blocks.

- Minimal hardware initialization
- Retrieve and verify object for later invocation
- Respond to requests for power management of processors

Goals

Elimination of complex functionality also eliminates subsequent and associated tasks in the project, such as documentation and testing.

Simplifications is another goal. Verification of signed images, could be provided by minimalistic cryptographic functions, without complex format such as X509.

Adaptation for formal verification. Together with the team responsible for modeling the actual hardware use in HASPOC, have been involved in several functional issues.

Support for industrial standards. The boot reference implementation from ARM (Arm Trusted Firmware version 1.1) has been used as a base for the project. This approach enables that Secure boot could integrate alternative payloads and service modules, according the the table below.

Portability. Generic functions, such as cryptographic functions should be portable, not only for smooth migration to new hardware, but also for enabling unit tests.

Functionality	Secure boot	ATF
Hardware initialization of a secure and constraint environment	X	I
Verified boot sequence by means of cryptographic checksums/signatures	X	
Power State Control Interface , for management of processor cores.		M
Start execution of Hypervisor		I
Software management of Secure Boot		O

I = Integrated , M = external module , O = optional function

Hardware initialization

This functionality starts when the system is powered on, or being reset. This function executes with full access to all hardware and must therefore be carefully designed, implemented and tested. Due to simplification the execution path is independent of previous power state, which assures that Secure boot transfer the hardware to next subsystem in a predefined and well-known state. This function could be triggered from the outside by altering the power to the TOE.

Verified boot sequence

When hardware initialization being successfully completed, the signed HASPOC image is retrieved from external storage followed by verification of each object found in the image. Prior to retrieval the trust anchor must be established, which conforms to the corresponding method in Arm Trusted Firmware. The hash value representing the actual public key is retrieved from special registers, which could be written only once.

Power state Control Interface

During hardware initialization all secondary cores are detected and blocked from further execution, only the primary core is allowed to execute. When Secure boot handover the hardware to next layer of software, only one core is active. Access to registers that control power state of core is by nature security critical and only accessible from Secure boot. The ARM architecture provides a mechanism based on secure messages allowing communication between different privilege / exception levels.

- modify guest access to a core,
- Memory Management Unit (MMU) support for a hypervisor mapping stage,
- peripherals with virtualization support, including virtual/shadow register section, virtual interrupt controller and S-MMU functionality,
- mechanism to monitor and control low-level communication: hypercalls, monitor calls, CPU event generation and interrupt signaling between cores.

Services

- SECURE POWER MANAGEMENT:Secure boot provides services for managing power states for cores, via a messaging service.

Assumptions

The vendor of the hardware provides a binary image (bl30.bin) required to configure the hardware. We assume that this image is correctly implemented.

The service runtime module originates from ARM reference implementation for boot. We assume that the power management implementation contained in this module is correctly implemented.

The runtime module also contain functionality required to change exception level and transition from secure to normal execution state. We assume that this function is correct.

The SFRs tracing to the TSFIs

The section gives thetrusted-board-boot.md mapping between SFRs and TSFIs.

SFR/TSFI	FPT_TST.1	FCS_CKM.1	FCS_CKM.4	FCS_COP.1
TSFI_POWERON_COLD				X
TSFI_STORAGE	X	X	X	X

TSFI_POWERON_COLD

This external interface relates to power-on/reset for the TOE. The functionality is represented by hardware initialization.

Purpose: Entrypoint of the boot sequence

Action: Configuration of hardware, upon success invocation of next stage.

Parameters: None

Errors:Error indication

TSFI-Storage

_main

This function retrieves each object
in the HSBF image and then boot
Upon sucess it never returns

int _main(boot_start_sector, initial_read)

Name	Direction	Type	Purpose
boot_start_sector	in	int	first sector of HSBF image
initial_read	in	int	limited read in pass 1

Return Codes

- BIOS_HIGH_FAILED_RETRIEVE_ROOT_PK_HASH
- BIOS_HIGH_FAILED_INSTALL_ROOT_PK_HASH
- BIOS_HIGH_MALLOC_SRAM_FAILED
- BIOS_HIGH_FAILED_VERIFY_ROOTPK
- BIOS_HIGH_ROOTPK_NOT_FOUND
- BIOS_HIGH_FAILED_VERIFY_DRAM
- BIOS_HIGH_DRAM_NOT_FOUND
- BIOS_HIGH_MALLOC_DRAM_FAILED
- BIOS_HIGH_FAILED_VERIFY_ATF_BOOT
- BIOS_HIGH_FAILED_VERIFY_RAMDISK
- BIOS_HIGH_FAILED_VERIFY_ATF_PAYLOAD
- BIOS_HIGH_FAILED_VERIFY_ATF_RUNTIME
- BIOS_HIGH_FAILED_UNKNOWN_OBJECT
- BIOS_HIGH_UNKNOWN_SIZE_HSBF
- BIOS_HIGH_FAILED_NO_BOOT_OBJECT
- BIOS_HIGH_FAILED_ATF_BOOT
- BIOS_HIGH_BAD_STORAGE_READ
- BIOS_HIGH_SDRAM_SETUP_ERROR
- BIOS_HIGH_FAILED_INIT_STORAGE

Chapter 2 .

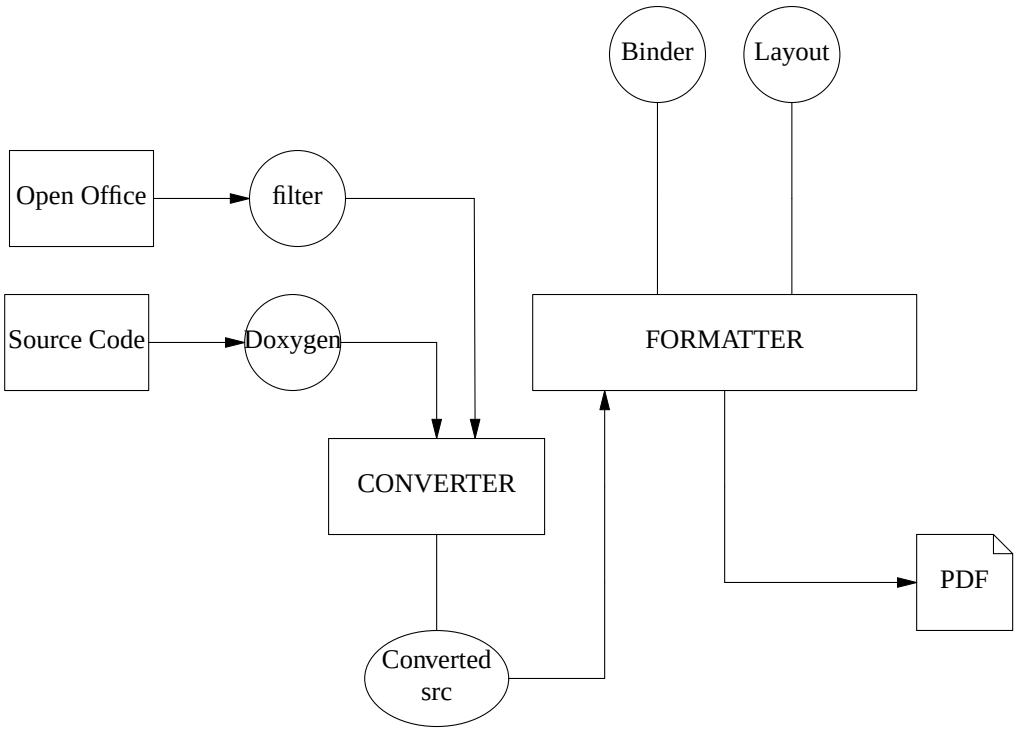
Bibliography

1: ARM Ltd., ARM Architecture Reference Manual - ARMv8, for ARMv8-A architecture profile, 2013
2: ARM Ltd., ARM v8-A Foundation Platform Version 9.1, 2014
3: 96 boards, HiKey ARMv8 community development board, 2015
4: ARM Ltd., SMC Calling Convention - System Software on ARM Platforms, 2013
5: ARM Ltd., Power State Coordination Interface (PSCI), 2015
7: Arm Trusted Firmware (docs/trusted-board-boot.md)

Chapter 3 .

Tools

Extensive testing is often required for high assurance systems, combined with verification of the implementation on code-level. To assure that the functional specification described in a document actually represents the implementation, header files describing interfaces are connected to document via conversion and transform of commented source code. In the Appendix of this document the header files are verbatim included.



Managing source

The picture above illustrates how multiple sources being included into the same document. Each source is first converted into metadata, and then formatted into the final document. The process is batch oriented and included into the software build process.

Entry point into high level

bios_high.h

Header file

```
/*! \file bios_high.h
    \brief High level boot logic

 @\n

    This module retrieves the HSBF image from storage

    and process each object.


    @\n

    The object is first verified



    @\n

    Upon failure the module returns a error code to C runtime.


*/

/*
  Copyright (C) 2016 Oniteo
 */

#ifndef __ONITEO_BIOS_HIGH_H__
#define __ONITEO_BIOS_HIGH_H__ "$HeadURL: https://cm-ext.dev.oniteo.com/svn/nanodev/pack-
ages/bios_high/branches/haspoc/hikey/bios_high/bios_high.h $ $Revision: 26907 $"


/*! \brief  Unable to allocate heap in SRAM  */
#define BIOS_HIGH_MALLOC_SRAM_FAILED 1

/*! \brief  Unable to verify ROOTPK against hash  */
#define BIOS_HIGH_FAILED_VERIFY_ROOTPK 2

/*! \brief  ROOTPK object not found */
#define BIOS_HIGH_ROOTPK_NOT_FOUND  3
```

```
/*! \brief  Unable to verify  DRAM object  */
#define BIOS_HIGH_FAILED_VERIFY_DRAM  4

/*! \brief  DRAM object  not found */
#define BIOS_HIGH_DRAM_NOT_FOUND  5

/*! \brief  Unable to allocate heap in DRAM  */
#define BIOS_HIGH_MALLOC_DRAM_FAILED 6

/*! \brief  Unable to verify ATF Boot  */
#define BIOS_HIGH_FAILED_VERIFY_ATF_BOOT  7

/*! \brief  Unable to verify  ramdisk  */
#define BIOS_HIGH_FAILED_VERIFY_RAMDISK 8

/*! \brief  unknown object in HSBF image   */
#define BIOS_HIGH_FAILED_UNKNOWN_OBJECT  9

/*! \brief   Unable to verify  ATF Payload bl33.bin   */
#define BIOS_HIGH_FAILED_VERIFY_ATF_PAYLOAD  10

/*! \brief   Unable to verify  ATF Runtime bl31.bin   */
#define BIOS_HIGH_FAILED_VERIFY_ATF_RUNTIME  11

/*! \brief   Size of HSBF image could not be obtained   */
#define BIOS_HIGH_UNKNOWN_SIZE_HSBF 12

/*! \brief   No object in HSBF that boot  */
#define  BIOS_HIGH_FAILED_NO_BOOT_OBJECT 13

/*! \brief   Failed to install hash value of public root key   */
#define BIOS_HIGH_FAILED_INSTALL_ROOT_PK_HASH 14

/*! \brief   Failed to retrieve hash value of public root key   */
#define BIOS_HIGH_FAILED_RETRIEVE_ROOT_PK_HASH 15

/*! \brief   Arm Trusted Firmware final boot step failed */
#define BIOS_HIGH_FAILED_ATF_BOOT 16

/*! \brief   Failed to retrieve image from external storage */
#define BIOS_HIGH_BAD_STORAGE_READ 17

/*! \brief   Failed to configure MCU/SDRAM */
#define BIOS_HIGH_SDRAM_SETUP_ERROR 18

/*! \brief   Failed to configure storage */
#define BIOS_HIGH_FAILED_INIT_STORAGE 19

/*! \brief   Number of sectors to read in pass 1 */
#define INITIAL_READ 300

/*! \brief   First sector to read  */
#define BOOT_START_SECTOR 8192
```

```
/*! \name Boot
@{
*/

/*! \brief retrieve HSBF and boot

    This function retrieves each object

    in the HSBF image and then boot

    Upon sucess it never returns

    \retval BIOS_HIGH_FAILED_RETRIEVE_ROOT_PK_HASH
    \retval BIOS_HIGH_FAILED_INSTALL_ROOT_PK_HASH
    \retval BIOS_HIGH_MALLOC_SRAM_FAILED
    \retval BIOS_HIGH_FAILED_VERIFY_ROOTPK
    \retval BIOS_HIGH_ROOTPK_NOT_FOUND
    \retval BIOS_HIGH_FAILED_VERIFY_DRAM
    \retval BIOS_HIGH_DRAM_NOT_FOUND
    \retval BIOS_HIGH_MALLOC_DRAM_FAILED
    \retval BIOS_HIGH_FAILED_VERIFY_ATF_BOOT
    \retval BIOS_HIGH_FAILED_VERIFY_RAMDISK
    \retval BIOS_HIGH_FAILED_VERIFY_ATF_PAYLOAD
    \retval BIOS_HIGH_FAILED_VERIFY_ATF_RUNTIME
    \retval BIOS_HIGH_FAILED_UNKNOWN_OBJECT
    \retval BIOS_HIGH_UNKNOWN_SIZE_HSBF
    \retval BIOS_HIGH_FAILED_NO_BOOT_OBJECT
    \retval BIOS_HIGH_FAILED_ATF_BOOT
    \retval BIOS_HIGH_BAD_STORAGE_READ
    \retval BIOS_HIGH_SDRAM_SETUP_ERROR
    \retval BIOS_HIGH_FAILED_INIT_STORAGE

    \param[in] boot_start_sector first sector of HSBF image
    \param[in] initial_read limited read in pass 1

*/
int _main(int boot_start_sector, int initial_read);




/*!
@}
*/


/*! \name C-Runtime
@{
*/

/*! \brief invoke boot logic
```

```
    This function assigns a small heap

    residing in static ram and then invoke main

    @\n

*/


void bios_high_runtime(void);

/*!
@}
*/



#endif
```