Secure Xen on ARM
User's Guide

Revision History

Version	Date	Revised contents
1.0	2008-06-11	Initial revision

Document Information

- Pages: 19 pages

Contact Information & Copyright

Software Lab.
Samsung Electronics Co., Ltd.
San 14-1, Nongseo-Dong, Giheung-Gu, Yongin-Si, Gyeonggi-Do
Korea 449-712

Contact: Sang-bum Suh *sbuk.suh@samsung.com*, Minsung Jang *minsung.jang@samsung.com*

Table of Contents

Overview

Purpose

This document describes the procedure of setting up a development environment for Secure Xen on ARM solution, building the solution, deploying it to a real target, and booting it for creating a VM.

Terminology & Acronyms

Term	Description
VMM	Virtual Machine Monitor
VM	Virtual Machine
Dom0	The privileged domain constructed by Xen on ARM at initial start-up time.
Dom1	An unprivileged domain constructed by dom0.
HID	Human Interface Device

References

1. ARM Ltd., ARM926EJ-S Technical Reference Manual, r0p4/r0p5
2. Freescale Semiconduct, “i.MX21 Application Processor Reference Manual,” Rev. 2, 2005
3. Xen Interface Manual
   http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/interface/interface.html
4. “Secure Architecture and Implemation of Xen on ARM for Mobile Devices”, Sang-bum Suh, presented at Xen Summit Spring 2007, IBM TJ Watson, Arpil 2007.
   http://www.xen.org/xensummit_4/Secure_Xen_ARM_xen-summit-04_07_Suh.pdf
5. “Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-Based Secure Mobile Phones”, Joo-Young Hwang, Sang-bum Suh, Sung Kwan Heo, Chan-Ju Park, Jae-Min Ryu, Seong-Yeol Park, Chul-Ryun Kim, In Proceedings of the 5th Annual IEEE Consumer Communications & Networking Conference, USA, January 2008.
6. “A Multi-Layer Mandatory Access Control Mechanism for Mobile Devices Based on Virtualization,” Sung-Min Lee, Sang-bum Suh, Bokdeuk Jeong, Sangdok Mo, In Proceedings of the 5th Annual IEEE Consumer Communications & Networking Conference, USA, January 2008.

1. Introduction

1.1. Overview

Xen is an open source VMM originated as a research project at the University of Cambridge. Its first version, 1.0, came out in 2003 and now the version has reached to 3.2. The virtualization technique adopted by Xen is para-virtualization which requires operating system modification.

Secure Xen on ARM is an ARM port of the x86 version of Xen-3.0.2-2 plus security features in Xen. It allows the simultaneous execution of multiple operating systems and their legacy application software stacks on a single ARM core-based system-on-chip. Each guest OS instance runs in their own OS partition called “domain” and the OS partitions are securely isolated from each other.

The original Xen solution comes with many user-land utilities. We have ported most core component such as xend, xenstore, xm and xenconsole.

Notes: The current version of Secure Xen on ARM only supports “static partitioning” of system memory, which means that the number of guest domains and the amount of memory allocated to the guest domains is fixed at compile time. You have to configure system memory partitioning properly before building Secure Xen on ARM.
The shadow page table and the writable page table are not included in this release but we are working on them so hopefully they will be enabled in our next release.

1.2. Hardware Support

The current release of the Secure Xen on ARM supports and tested on the following platform.

- MX21ADS evaluation board shipped from Freescale Semiconductor (DBMX9328).

2. Building Environment

We tested Secure Xen on ARM are successfully built under the system environment:

- Compiler: GCC-3.4 or higher
- OS: Linux Fedora Core 6

3. Installation Procedure

3.1. Host PC Environment Setup

3.1.1. Cross Toolchain

The gcc 3.4.4 and glibc 2.3.5 is used for cross-compilation. You can download it at the links.
http://www.ertos.nicta.com.au/downloads/tools/arm-linux-3.4.4.tar.gz
http://www.ertos.nicta.com.au/downloads/tools/arm-linux-3.4.4.tar.bz2

3.1.2. Tftp Server

You can download binary files from host PC to target by using tftp. If tftp server is not configured in the host PC, install and setup a tftp server first. Installing tftp server and enabling tftp service can be different depending on your host pc environment.

3.1.3. Minicom

You can access to the target by using a terminal program. '**Minicom**' is one of the popular terminal programs running in Linux PC. Here we'll explain about serial port set-up in Minicom. You can also use other terminal programs such as '**HyperTerminal**' in Windows PC.

1. Execute a minicom on setting mode.
   # minicom -s

2. Select '**Serial port setup**' menu and set up the parameters as in Figure 3-1.

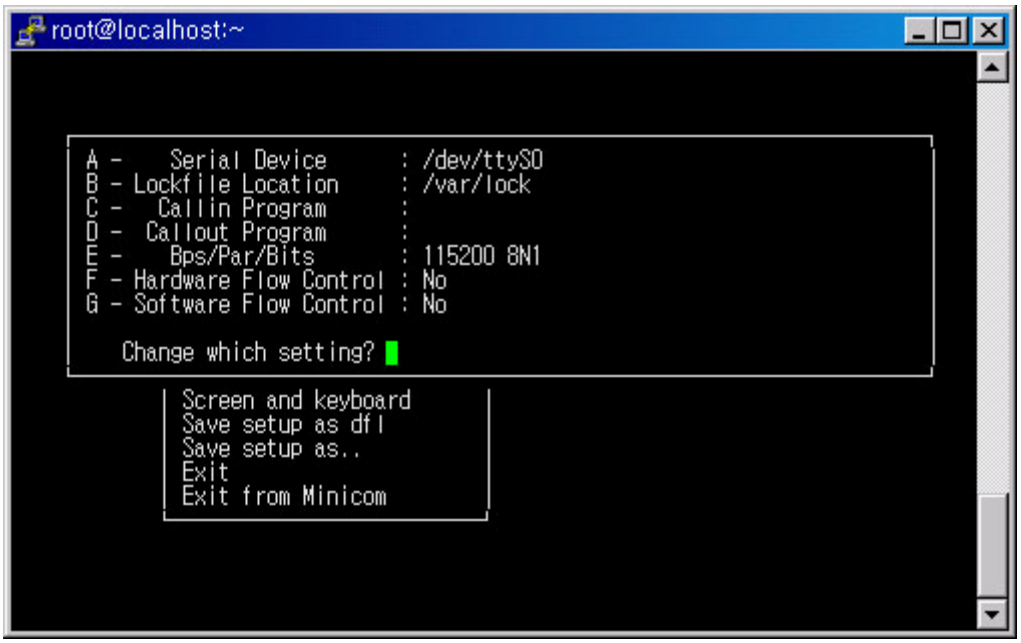

Figure 3-1 Minicom Configuration

3.2. Domains Configuration

3.2.1. Configuring Memory Partition

On the current release of Secure Xen on ARM, the number of domains and the amount of memory allocated to the guest domain is fixed. You have to configure system memory partition properly before building Secure Xen on ARM.

1. Go to the Secure Xen on ARM source directory. $(XEN_ROOT) is root directory of the Secure Xen on ARM.

   # cd $(XEN_ROOT)

2. The all works are done with menuconfig script. We have assigned the default values as illustrated in the Figure 3-2.

   # make menuconfig

```
General Setup  --->
System Type  --->
Customize Memory Map  --->
        (0xFF000000) Hypervisor virtual address
                Domain Memory Size  --->
                        (0x02000000) domain0 memory size (including xen memory size :
2MB)
                        (0x01000000) domain1 memory size (NEW)
                        (0x01000000) domain2 memory size (NEW)
                        (0x00000000) domain3 memory size (NEW)
                Image Max Size  --->
                        (0x00400000) domain0 image max size (NEW)
                        (0x00400000) domain1 image max size (NEW)
                        (0x00400000) domain2 image max size (NEW)
                        (0x00400000) domain3 image max size (NEW)
                Ram Disk Size  --->
                        (0x00400000) domain0 ramdisk size (NEW)
                        (0x00400000) domain1 ramdisk size (NEW)
                        (0x00400000) domain2 ramdisk size (NEW)
                        (0x00400000) domain3 ramdisk size (NEW)
Security Support  --->
Debugging and profiling Support  --->
```

Figure 3-2 Memory Partitioning

The meaning of each menu entry displayed in the Figure 3-2 is as followings.

- The Domain Memory Size means the amount of memory allocated to the guest domain.
- The Image Max Size means the maximum size of guest kernel image to be loaded. The guest kernel image should be less than the value.

The Ram Disk Size is mainly used for Ram disk. This value may be neglected if you do not use the ramdisk.

3.3. Compilation

3.3.1. How to compile Secure Xen on ARM

1. Go to the Secure Xen on ARM source directory. $(XEN_ROOT) is root directory of the Secure Xen on ARM.

 # cd $(XEN_ROOT)

2. Lets get on with the task of configuring the Secure Xen on ARM. When you do the make menuconfig of your Secure Xen on ARM configuration, don't forget to check the system type that is suitable for your target board.

 # make menuconfig

```
General Setup  --->
    [ ] optimize for size
    [ ] Use AEM EABI to compile the Secure Xen on ARM

System Type  --->
    Select target platform (Freescale i.MX21ADS board)  --->
            ( ) Samsung Fiore board
            (X) Freescale i.MX21ADS board
            ( ) Texas Instrument OMAP5912 OSK board

Customize Memory Map  --->
    (0xFF000000) Hypervisor virtual address

Security Support  --->
    [ ] Security support

Debugging and profiling Support  --->
    [ ] Debugging and Profiling Support
```

Figure 3-3 Secure Xen on ARM Configuration

3. Finally compile the Secure Xen on ARM by executing '**make**' command.

 # make

4. Then **xen-bin** file is created in $(XEN_ROOT)/xen, and copy to root directory of tftp server.

 # cp $(XEN_ROOT)/xen/xen-bin $(TFTP_ROOT)

3.3.2. How to compile mini-os

1. Go to the mini-os source directory. The mini-os source directory is $(XEN_ROOT)/extras/mini-os-arm.

 # cd $(XEN_ROOT)/extras/mini-os-arm

2. Finally compile the mini-os by executing '**make**' command.

 # make

3. Then **mini-os.elf** file is created in $(XEN_ROOT)/tools/minios, and copy to root directory of tftp server.

 # cp $(XEN_ROOT)/extras/mini-os-arm/mini-os.elf $(TFTP_ROOT)

3.4. Running Secure Xen on ARM and two mini-os

We assume that u-boot is installed in the i.mx21 target board.

1. Turn on the target board, then boot the target and enter interactive command line mode of u-boot.

2. Download Xen and mini-os image.
u-boot> tftp 0xc0008000 xen-bin
u-boot> tftp 0xc1c00000 mini-os.elf
u-boot> tftp 0xc2c00000 mini-os.elf

3. Execute Secure Xen on ARM and mini-os.
u-boot> go 0xc0008000

4. Then you can see running the two mini-os on Secure Xen on ARM as in Figure 3-4.

```
Samsung:mx21ads> tftp 0xc0008000 xen-bin
TFTP from server 192.168.161.1; our IP address is 192.168.161.2
Filename 'xen-bin'.
Load address: 0xc0008000
Loading: ##########################################
done
Bytes transferred = 206388 (32634 hex)
Samsung:mx21ads> tftp 0xc1c00000 mini-os.elf
TFTP from server 192.168.161.1; our IP address is 192.168.161.2
Filename 'mini-os.elf'.
Load address: 0xc1c00000
Loading: #################################
done
Bytes transferred = 161538 (27702 hex)
Samsung:mx21ads> tftp 0xc2c00000 mini-os.elf
TFTP from server 192.168.161.1; our IP address is 192.168.161.2
Filename 'mini-os.elf'.
Load address: 0xc3c00000
Loading: #################################
done
Bytes transferred = 161538 (27702 hex)
Samsung:mx21ads> go 0xc0008000
## Starting
 [XEN]
Starting Xen/ARM
Copyright (C) 2007 Samsung Electronics Co, Ltd. All Rights Reserved.
[XEN] Using scheduler: Borrowed Virtual Time (bvt)
[XEN] *** LOADING DOMAIN : 0 ***
[XEN] Physical Memory Arrangement: c0200000->c2000000
[XEN] VIRTUAL MEMORY ARRANGEMENT:
[XEN]  Loaded kernel: c0008000->c00249f4
[XEN]  Init. ramdisk: c0025000->c0025000
[XEN]  Phys-Mach map: c0025000->c002c800
[XEN]  Start info:    c002d000->c002e000
```

```
[XEN]  Page tables:   c0030000->c0046000
[XEN]  Boot stack:    c0046000->c0047000
[XEN]  TOTAL:         c0000000->c1e00000
[XEN]  ENTRY ADDRESS: c0008000
[XEN]  [TODO] dma channel access permission, in construct_dom0()
[XEN] [dom0] Bootstrapping .....
[dom0] Xen Minimal OS!
[dom0] Magic : xen-3.0-arm_32
[dom0] Total Pages allocated to this domain : 7680
[dom0] MACHINE address of shared info struct : 0xc00f1000
[dom0] VIRTUAL address of page directory : 0xc0030000
[dom0] Number of bootstrap p.t. frames : 22
[dom0] VIRTUAL address of page-frame list : 0xc0025000
[dom0] VIRTUAL address of pre-loaded module : 0x0
[dom0] Size (bytes) of pre-loaded modules : 0
[dom0] min mfn(min_page in xen) : 786432
[dom0] Command-Linux Address : 0xc002d054
[dom0] Command-Line String :
[dom0]   flags       : 0x3
[dom0] sizeo fo pte_t = 4
[dom0] Shared info Offset = 0x0
[dom0] Shared Info : 0xc0009000
[dom0] MM: Init
[dom0] MM: Initializing page allocator for memory c0049000(49000)-c1e00000(1e00)
[dom0] MM: initializing page allocator
[dom0] MM: initializing grant table
[dom0] MM: done
[dom0] Initialising timer interface
[dom0] Initialising scheduler
[dom0] Thread "Idle": pointer: 0xc004a00c, stack: 0xc004c000
[dom0] Thread "3": pointer: 0xc004a034, stack: 0xc004e000
[dom0] Thread "4": pointer: 0xc004a05c, stack: 0xc0050000
[dom0] Thread "5": pointer: 0xc004a084, stack: 0xc0052000
[dom0] Thread "6": pointer: 0xc004a0ac, stack: 0xc0054000
*** LOADING DOMAIN : 1 ***
[XEN] Physical Memory Arrangement: c2000000->c4000000
[XEN] VIRTUAL MEMORY ARRANGEMENT:
[XEN]  Loaded kernel: c0008000->c00249f4
[XEN]  Init. ramdisk: c0025000->c0025000
[XEN]  Phys-Mach map: c0025000->c002d000
[XEN]  Store mfn:     c002d000->c002e000
[XEN]  Console mfn:   c002e000->c002f000
[XEN]  Start info:    c002f000->c0030000
[XEN]  Page tables:   c0030000->c0047000
[XEN]  Boot stack:    c0047000->c0048000
[XEN]  TOTAL:         c0000000->c2000000
[XEN]  ENTRY ADDRESS: c0008000
[XEN] store_mfn physical addres c202d000
[XEN] console_mfn physical addres c202e000
[XEN]  [TODO] dma channel access permission, in construct_guest_dom()
[XEN] [dom1] Bootstrapping .....
[dom1] Xen Minimal OS!
[dom1] Magic : xen-3.0-arm_32
[dom1] Total Pages allocated to this domain : 8192
[dom1] MACHINE address of shared info struct : 0xc00ed000
[dom1] VIRTUAL address of page directory : 0xc0030000
[dom1] Number of bootstrap p.t. frames : 23
[dom1] VIRTUAL address of page-frame list : 0xc0025000
```

```
[dom1] VIRTUAL address of pre-loaded module : 0x0
[dom1] Size (bytes) of pre-loaded modules : 0
[dom1] min mfn(min_page in xen) : 786432
[dom1] Command-Linux Address : 0xc002f054
[dom1] Command-Line String :
[dom1]   flags       : 0x0
[dom1] sizeo fo pte_t = 4
[dom1] Shared info Offset = 0x1
[dom1] Shared Info : 0xc000a000
[dom1] MM: Init
[dom1] MM: Initializing page allocator for memory c004a000(4a000)-c2000000(2000)
[dom1] MM: initializing page allocator
[dom1] MM: initializing grant table
[dom1] MM: done
[dom1] Initialising timer interface
[dom1] Initialising scheduler
[dom1] Thread "Idle": pointer: 0xc004b00c, stack: 0xc004c000
[dom1] Thread "3": pointer: 0xc004b034, stack: 0xc004e000
[dom1] Thread "4": pointer: 0xc004b05c, stack: 0xc0050000
[dom1] Thread "5": pointer: 0xc004b084, stack: 0xc0052000
[dom1] Thread "6": pointer: 0xc004b0ac, stack: 0xc0054000
[dom0] 3 Thread
[dom1] 3 Thread
[dom0] 4 Thread
[dom1] 4 Thread
[dom0] 5 Thread
[dom1] 5 Thread
[dom0] 6 Thread
[dom1] 6 Thread
[dom0] After 3 Thread
[dom1] After 3 Thread
[dom0] 3 Thread
[dom1] 3 Thread
[dom0] After 4 Thread
[dom1] After 4 Thread
[dom0] 4 Thread
[dom1] 4 Thread
[dom0] After 5 Thread
[dom1] After 5 Thread
[dom0] 5 Thread
[dom1] 5 Thread
[dom0] After 6 Thread
[dom1] After 6 Thread
[dom0] 6 Thread
[dom1] 6 Thread
[dom0] After 3 Thread
[dom1] After 3 Thread
[dom0] 3 Thread
[dom1] 3 Thread
[dom0] After 4 Thread
[dom1] After 4 Thread

…
```

Figure 3-4 Console after two mini-os booting

4. How to Enable Security Features

Secure Xen on ARM provides some kinds of security features. This chapter explains how users can use them. Two jobs are required to enable security features. One is about boot loader, and the other is about cryptographic library. Without this job, Secure Xen on ARM may not operate properly.

4.1. Why modify boot loader for Secure Xen on ARM?

In order to use security features, Secure Xen on ARM should be based on security-chain (i.e., security-enabled boot loader → Secure Xen on ARM → security-enabled OS). In this release of Secure Xen on ARM, however, you cannot use full security features we made because this release does not include security-enabled boot loader and security-enabled OS. The core of security-chain is composed of secure-boot and security-information-transfer from boot loader to Secure Xen on ARM.

4.1.1. Secure boot

Security-enabled boot loader makes use of data located on the secure ROM (actually it depends on a target platform feature) and other persistent memory like NOR or NAND flash memory. The data needed for security-enabled boot loader include master key, certificate, access control policies, binary images, their signatures and so on. To make security-enabled boot loader run correctly, these data should be stored on a secure ROM and a persistent memory.

The process of secure boot is as follows:
First of all, boot loader initializes security information for secure boot and checks integrity of Secure Xen on ARM image. If the Secure Xen on ARM image is not changed, boot loader starts it.
In addition, the Secure Xen on ARM receives the security information from boot loader and checks integrity of OS images. If the OS images are not changed, Secure Xen on ARM runs them.

4.1.2. Security information transfer

The security information described above is shared with a boot loader and Secure Xen on ARM. So the security information must be transferred from the boot loader to Secure Xen on ARM. To share the security information, the security-enabled boot loader calls the Secure Xen on ARM image just as a function with a parameter and transfers the security information through the parameter.

4.2. Detailed description about boot loader operation

4.2.1. Initialization

As we described above, the security information, which consist of master key, certificate, access control policies, and binary images (VMM image, kernel images) & their signatures, etc., are used for secure boot and other security mechanisms by boot loader and Secure Xen on ARM. So the boot loader reads these information right after it runs. Generally, the master key is loaded from a secure ROM, and others are loaded from a persistent memory, but it depends on a target board.

4.2.2. To load Secure Xen on ARM and dom0 binary images to predefined memory locations

Based on the security information, the boot loader knows the memory location to be loaded Secure Xen on ARM and OS image. The boot loader loads Secure Xen on ARM and OS binary image as well as other security information (e.g., access control policy, cryptographic keys, etc) to predefined memory location.

4.2.3. To verify Secure Xen on ARM binary image

It is important that the boot loader checks integrity of the Secure Xen on ARM image with its signature, because the Secure Xen on ARM modified by a malicious entity can be used to attack the OSes on top of it. If the Secure Xen on ARM is altered, the boot loader stops its execution and warns about this modification.

4.2.4. To call the Secure Xen on ARM binary image with a parameter

If the verification is successfully passed, the boot loader calls the Secure Xen on ARM binary image as a function with a parameter, which is a pointer type of 'ssb_transfer_container_t' structure defined in 'secure_storage_struct.h'. Followings are sample codes (see Sample code 4-1) for a security-enabled boot loader to start Secure Xen on ARM.

```
/* XEN_ARM_POS       predefined memory location for Secure Xen on ARM */
/* SEC_INFO_POS      memory location for security information structure */

void (*start_xen_arm)(ssb_image_container_t *trans);
ssb_transfer_container_t *ptc=NULL;

start_xen_arm = (void(*)(ssb_image_container_t *)) XEN_ARM_POS;
ptc = (ssb_transfer_container_t *) SEC_INFO_POS;
/* Secure Xen on ARM start */
(*start_xen_arm)(ptc);
```

Sample code 4-1

4.3. Data structure to be transferred from boot loader to Secure Xen on ARM

In the 'secure_storage_struct.h' header file, you can see the detailed data structure to be transferred from a boot loader to Secure Xen on ARM. As you can see from the sample codes below, transferred structure is the 'ssb_transfer_container_t'. This structure contains 'transfer_struct_t' which has a binary image for each secure partition (see Sample code 4-2).

```
typedef enum {
  PART_MBB, PART_SP1=1, PART_SP2=2, PART_SP3=3, PART_OS_IMAGE=4,
  PART_DRV_RFS, PART_DOM0_RFS, PART_DOM1_RFS, PART_DOM2_RFS,
  TRANSFER_MASTER_KEY, PART_SUB_VMM_IMAGE,
  PART_END
} partition_type_t;

typedef struct {
  partition_type_t type;
  u_int32_t size;
  char *ptr;
} transfer_struct_t;
```

```
typedef struct {
  u_int32_t images_size;
  transfer_struct_t image[MAX_IMAGE_STRUCT_SIZE];
} ssb_transfer_container_t;
```

Sample code 4-2

If you would like to know more about transferred data structure and binary format, see the following functions in the 'sra_func.c' and 'crypto.c' (see Sample code 4-3).

```
int sra_init(void);
int crypto_init(void);
int _load_part(ssb_transfer_container_t *tc, partition_type_t ptype);
int init_master_key(void);
```

Sample code 4-3

4.4. Description about cryptographic library

4.4.1. Overview

The 'crypto.c' file in the 'crypto' directory contains APIs for cryptographic functions and these APIs are used for cryptographic operations such as key/algorithms initialization, hashing, encryption, decryption, and digital signature (They are called by Secure Xen on ARM). If there is cryptographic hardware support, you can connect these APIs to hardware cryptographic engine. Otherwise, you can use open source cryptographic library such as OpenSSL. In this release of Secure Xen on ARM, only cryptographic APIs are provided. So if you want to use cryptographic APIs, you must get cryptographic library first and connect those APIs to cryptographic functions.

4.4.2. APIs for cryptographic library

4.4.2.1. int init_master_key(void);

Description

This function loads the master key to be used by cryptographic library.
This function must be called once before other cryptographic library is called.

Parameters

None

Return Value

0 if succeeds, otherwise error value

4.4.2.2. int crypto_init(void);

Description

This function selects cryptographic algorithm and gets the RSA public key or public key certificate
This function must be called once before other cryptographic library is called.

Parameters

None

Return Value

0 if succeeds, otherwise error value

4.4.2.3. int crypto_sign_data(unsigned char* src, int src_len, unsigned char** sig, int* sig_len);

Description

This function generates a digital signature using RSA algorithm.

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
sig	unsigned char**	Pointer to the pointer to the digital signature. If *sig == NULL, newly allocated memory is returned. In the case, memory must be freed by caller
sig_len	int*	Byte length of digital signature

Return Value

0 if succeeds, otherwise error value

4.4.2.4. int crypto_verify_data(unsigned char* src, int src_len, unsigned char* sig, int sig_len);

Description

This function verifies image with digital signature using RSA algorithm.

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
sig	unsigned char*	Pointer to the digital signature.
sig_len	int	Byte length of digital signature

Return Value

0 if succeeds, otherwise error value

4.4.2.5. int crypto_verify_data_with_certm(unsigned char* src, int src_len, unsigned char* certm, int certm_len);

Description

This function verifies image with manufacture’s certificate using RSA algorithm.

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
certm	unsigned char*	Pointer to manufacture’s certificate
certm_len	int	byte length of manufacture’s certificate

Return Value

0 if succeeds, otherwise error value

4.4.2.6. int crypto_hash_data(unsigned char* src, int src_len, unsigned char** hash, int* hash_len);

Description

This function hashes image using SHA-1 algorithm

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
hash	unsigned char**	Pointer to the pointer to the hashed data. If *hash == NULL, newly allocated memory is returned. In the case, memory must be freed by caller
hash_len	Int *	byte length of hashed data

Return Value

0 if succeeds, otherwise error value

4.4.2.7. int crypto_encrypt_data(unsigned char *src, int src_len, unsigned char **enc, int *enc_len);

Description

This function encrypts data using AES algorithm

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
enc	unsigned char**	Pointer to the pointer to the encrypted data. If *enc == NULL, newly allocated memory is returned. In the case, memory must be freed by caller
enc_len	int*	Byte length of encrypted data

Return Value

0 if succeeds, otherwise error value

4.4.2.8. int crypto_decrypt_data(unsigned char *src, int src_len, unsigned char **dec, int *dec_len);

Description

This function decrypts data using AES algorithm

Parameters

Name	Type	Description
src	unsigned char *	Pointer to the source data
src_len	int	Byte length of source data
dec	unsigned char**	Pointer to the pointer to the decrypted data. If *dec == NULL, newly allocated memory is returned. In the case, memory must be freed by caller
dec_len	int*	Byte length of decrypted data

Return Value

0 if succeeds, otherwise error value

Appendix

A.1. New hypercalls for Secure Xen on ARM

Table 1 describes newly added hypercalls to Secure Xen on ARM.

Table 1New hypercalls for Secure Xen on ARM

Hypercall Name	Description
__HYPERVISOR_restore_guest_context	Restore CPU context stored in guest kernel stack.
__HYPERVISOR_do_print_profile	Dispatch profiling data
__HYPERVISOR_do_set_foreground_domain	Change foreground domain.
__HYPERVISOR_do_set_HID_irq	Register HID irq. The HID irq is only delivered to foreground domain select by **__HYPERVISOR_do_set_foreground_domain** hypercall.
__HYPERVISOR_dma_op	Request DMA operations
__HYPERVISOR_set_pirq_type	Change IRQ type and attributes
__ HYPERVIOSR_do_acm_op	Override native Xen hypercall. User can choose native or Secure Xen hypercall via menuconfig
__HYPERVISOR_sra_op	Manage secure storage data

A.2. Credit

Sang-bum Suh (sbuk.suh@samsung.com)
Jooyoung Hwang (jooyoung.hwang@samsung.com)
Sungmin Lee (sung.min.lee@samsung.com)
Chanju Park (bestworld@samsung.com)
Sungkwan Heo (sk.heo@samsung.com)
Sangdok Mo (sd.mo@samsung.com)
Jaemin Ryu (jy0922.shim@samsung.com)
Bokdeuk Jung (bd.jeong@samsung.com)
Junghyun Yoo (yjhyun.yoo@samsung.com)
Minsung Jang (minsung.jang@samsung.com)
Joonyoung Shim (jy0922.shim@samsung.com)
Donghyuk Lee (dh5050.lee@samsung.com)
Inki Dae (inki.dae@samsung.com)
Yongho Hwang (yongh.hwang@samsung.com)