Dm-verity in Android

Document Number:		**Document Version:**	
Owner:		**Date:**	
Document Type:			
NOTE:	ALL MATERIALS INCLUDED HEREIN ARE COPYRIGHTED AND CONFIDENTIAL UNLESS OTHERWISE INDICATED. The information is intended only for the person or entity to which it is addressed and may contain confidential and/or privileged material. Any review, retransmission, dissemination, or other use of or taking of any action in reliance upon this information by persons or entities other than the intended recipient is prohibited. This document is subject to change without notice. Please verify that your company has the most recent specification. Copyright © 2014 Spreadtrum Communications Inc.		

www.spreadtrum.com

版本历史

Revision	Date	Description
0.1	2015-02-20	Initial draft
0.2	2015-06-24	Update implementation
0.3	2015-11-11	Update some questions
0.4	2016-11-11	Update how to adapt
0.5	2017-03-30	Update android7.0
0.6	2017-04-01	Update common questions
0.7	2017-05-26	更新对性能的影响
1.0	2017-10-25	增加 key 配置说明及安全性建议

内容索引

1. 简介

该文档主要介绍了 Android verify boot 过程中对于 system 分区的校验原理以及实现方式。

1.1. 需求 & 目的

用于展讯平台中实现终端安全的能力及方案，包含以下内容：
1、DM-verity 介绍；
2、SPRD 平台，实现此项特性需要做的工作；

1.2. 定义

Name	Description
DM	Device mapper
OTA	Over-the-Air Technology

1.3. 相关内容

[1] https://source.android.com/security/verifiedboot/verified-boot
[2] https://github.com/nelenkov/cryptsetup

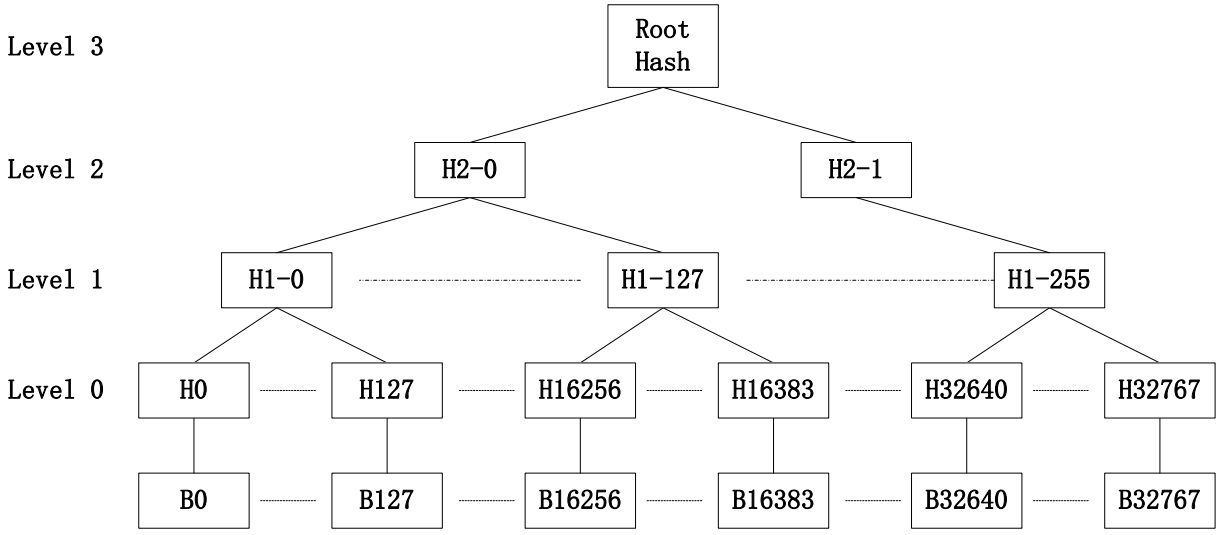
2. DM-verity 总览

Dm-verity 是一项 kernel 的功能（Android 从 5.1 版本开始支持），它可以提供透明的对块设备的完整性校验，这可以用于防止恶意程序对系统分区的篡改。

Dm-verity 虚拟了一个块设备，当对块数据进行读取时会首先进行哈希计算，并与预先计算好的哈希树进行校验，如果匹配则读取成功，否则会生成一个读取的 I/O 错误，以此达到对数据完整性校验的目的。

这棵预先计算好的哈希树包含了要校验的目标设备的所有块，对于每个块(一般是 4 k),有一个 SHA256 散列（32 字节），它存储在树的叶子节点。树的中间节点则是对这些叶节点的再次 SHA256 散列（还是以 4k 为一个单位），通过这样多次反复的哈希计算，直到得出唯一的一个哈希值为止，这个唯一的数值称为 Root Hash。那么基于散列的特性，当任意一个 block 有任意的变化，都会导致根哈希数值的变化。

以一个包含 32768 个块的设备为例，哈希树的结构图是这样的：



3. 前提

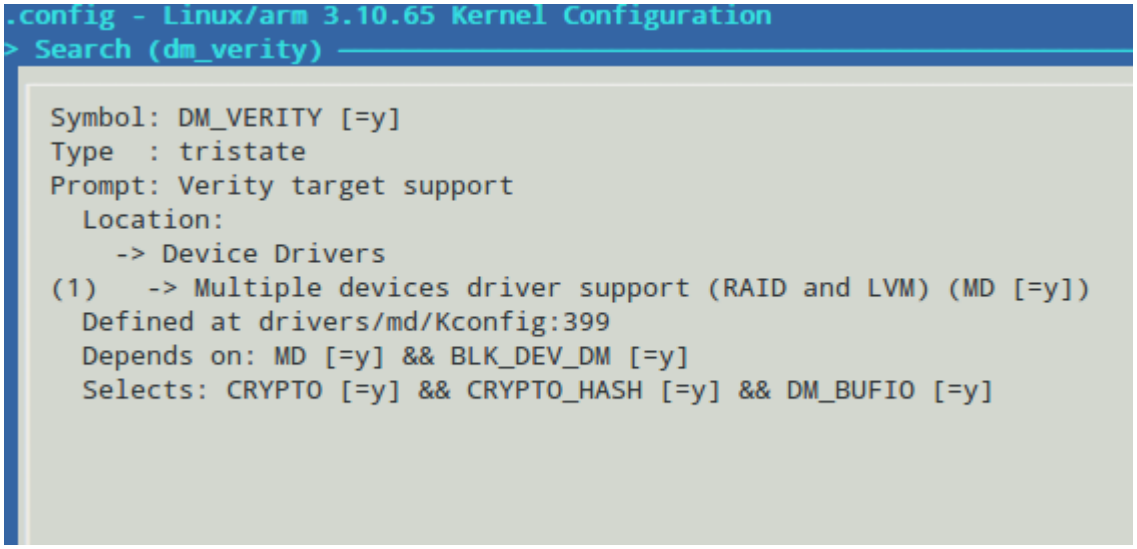
当使能 dm-vertiy 之后，需要 OTA 升级也切换到基于 Block 的 OTA 升级方式。也即是说 OTA 更新要在块设备层操作，并同时更新哈希树和 Verity metadata。

4. Android 实现

通过以下步骤，可以使能 dm-verity 功能：

4.1. Kernel 中 配 置 dm-verity

```
Symbol: DM_VERITY
Prompt: Verity target support
Location:-> Device Drivers
```



Q: 哪里修改？
A: 在/kernel/arch/arm/configs/下面对应工程的配置。

4.2. 生成 hash tree for ext4 system 分区

Google 提供了一个 veritysetup 的工具，可用来生成哈希树。
命令参数及结果示例如下：

```
veritysetup format system.img syshash.txt
VERITY header information for syshash.txt
UUID:            aec254e2-525a-4abe-9fb3-ef357bf8106d
Hash type:       1
Data blocks:     117187
Data block size: 4096
Hash block size: 4096
Hash algorithm:  sha256
Salt:            b7a5041a13ceccedbe9bdba47b256e260b0a40d7e2b7726b7cff239c04dfe2fa
Root hash:       ca6164159179bdd647f913286d33cbc98bc6fb97fcf3c30a7f0a81a60bbdbe51
```

Q: veritysetup 在哪里?
A: https://github.com/nelenkov/cryptsetup 一个开源的网址。下载下来直接编译就好了。

4.3. 编译 dm-verity table for hash tree

dm-verity table 用于系统启动时构建 verity 的块设备，示例如下：

```
1❶ /dev/block/mmcblk0p21❷ /dev/block/mmcblk0p21❸ 4096❹ 4096❺
204800❻ 204809❼ sha256❽
1F951588516c7e3eec3ba10796aa17935c0c917475f8992353ef2ba5c3f47bcb❾
5f061f591b51bf541ab9d89652ec543ba253f2ed9c8521ac61f1208267c3bfb1❿
```

1、版本号
2、校验的目标设备
3、哈希树存储设备
4、目标块大小
5、哈希块大小
6、目标设备偏移
7、哈希树偏移
8、哈希算法名称
9、根哈希
10、salt 值

Q: 在哪里可以看到?
A: 查看编译输出：

```
build_verity_tree
        -A
        aee087a5be3b982978c923f566a94613496b417f2af592639bc80d141e34dfe7
        out/target/product/scx35_sp7731gea_hdr/obj/PACKAGING/systemimage_intermediates/system.img
        /tmp/tmpJWWIrJ_verity_images/verity.img
system/extras/verity/build_verity_metadata.py
        386252472
        /tmp/tmpJWWIrJ_verity_images/verity_metadata.img
        84f4d12089f9dfa3810d71e33a4d797bf9580137aee8107abcc0ca63b5838dc3
        aee087a5be3b982978c923f566a94613496b417f2af592639bc80d141e34dfe7
        /dev/block/platform/sdio_emmc/by-name/system
        out/host/linux-x86/bin/verity_signer build/target/product/security/verity.pk8
```

4.4. 签名 dm-verity table

对前面生成的 table 进行签名，这主要是用来保证根哈希的完整性。用于验证签名的公钥存放在 boot image 根目录下，名为 verity_key 的文件中。

4.5. 配置 verity flag

需要修改 fstab，增加 verity 标志，以标识此分区需要打开校验功能，示例如下

```
/dev/block/platform/msm_sdcc.1/by-name/system /system ext4 ro,barrier=1 wait,verify
```

4.6. 编译 table signature & dm-verity table 进 verity metadata

Verity metadata block 包含如下内容：

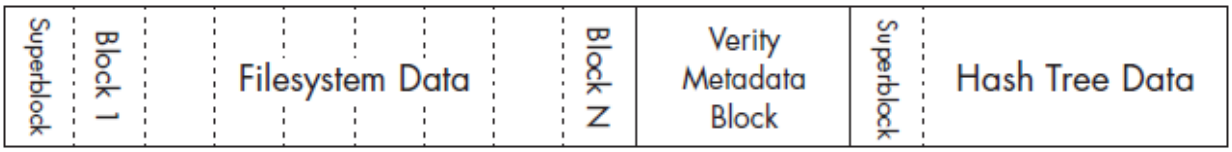
Field	Description	Size	Value
Magic number	Used by fs_mgr as a sanity check	4 bytes	0xb001b001
Version	Metadata block version	4 bytes	Currently 0
Signature	Mapping table signature (PKCS#1 v1.5)	256 bytes	
Mapping table length	Mapping table length in bytes	4 bytes	
Mapping table	dm-verity mapping table	variable	
Padding	Zero-byte padding to 32k byte length	variable	

Q: 哪里可以看到？

A: 编译后直接打包进 system 分区的，如果一定要看，可以通过二进制工具直接打开 system.img 来查看。

4.7. 打包 system image,verity metadata & hash tree

最终生成的 system image 应该是这样的：



Q: 如何检查功能？

A: 首先：编译完成后会有相应的提示。下面的就是 verity 功能的编译的相关 log。

```
build_verity_tree
        -A
        aee087a5be3b982978c923f566a94613496b417f2af592639bc80d141e34dfe7
        out/target/product/scx35_sp7731gea_hdr/obj/PACKAGING/systemimage_intermediates/system.img
        /tmp/tmpJWWIrJ_verity_images/verity.img
system/extras/verity/build_verity_metadata.py
        386252472
        /tmp/tmpJWWIrJ_verity_images/verity_metadata.img
        84f4d12089f9dfa3810d71e33a4d797bf9580137aee8107abcc0ca63b5838dc3
        aee087a5be3b982978c923f566a94613496b417f2af592639bc80d141e34dfe7
        /dev/block/platform/sdio_emmc/by-name/system
        out/host/linux-x86/bin/verity_signer build/target/product/security/verity.pk8
```

其次：可以通过 mount 命令，看 system 分区挂载在/dev/block/dm-0 或者/dev/block/dm-1 上（加密后的 data 分区也会挂载到这个虚拟设备上。）。
最后：此功能仅在 user 版本生效，userdebug 版本不生效。功能开启后，OTA 的升级方式需要改成以 Block 方式升级。

5. Android6.0 中 SPRD 适配的修改

在 Android5.1 及之后的版本中，Google 已将生成哈希树、生成 verity table，打包 metadata 等工作集成到它的编译系统中，我们只需要将相关的开关打开。
1、参考 4.1 内容，打开 kernel 的相关的配置。参考修改：

```
# CONFIG_DM_FLAKEY is not set
# CONFIG_DM_VERITY is not set

# CONFIG_TARGET_CORE is not set
```

```
1254 # CONFIG_DM_FLAKEY is not set
1255 CONFIG_DM_BUFIO=y
1256 CONFIG_DM_VERITY=y
1257 # CONFIG_TARGET_CORE is not set
```

2、修改 Board 相关的配置（这里以 7731g 为例）：

2.1、 /dev/sprd/scx35/common/device.mk，增加 dm-verity 的编译配置，同时用 TARGET_DM_VERITY 宏控制。参考修改：

device/sprd/scx35 / common/**device.mk**

Patch Set Base 1 2 (gitweb)

```
... skipped 238 common lines ... +104
239 else
240 ifeq ($(strip $(PRODUCT_RAM)),high)
241 $(call inherit-product, frameworks/native/build/phone-xhdpi-1024-dalvik-heap.mk)
242 PRODUCT_PROPERTY_OVERRIDES += \
243 »       ro.board_ram_size=high \
244 »       ro.config.low_ram=false \
245 »       ro.product.ram=high
246 endif
247 endif
248 # @}
249
```

Patch Set 1 2 (gitweb)

```
... skipped 238 common lines ... +104
239 else
240 ifeq ($(strip $(PRODUCT_RAM)),high)
241 $(call inherit-product, frameworks/native/build/phone-xhdpi-1024-dalvik-heap.mk)
242 PRODUCT_PROPERTY_OVERRIDES += \
243 »       ro.board_ram_size=high \
244 »       ro.config.low_ram=false \
245 »       ro.product.ram=high
246 endif
247 endif
248 # @}
249 ifeq (true,$(strip $(TARGET_DM_VERITY)))
250 $(call inherit-product, build/target/product/verity.mk)
251 PRODUCT_SYSTEM_VERITY_PARTITION := /dev/block/platform/sdio_emmc/by-name/system
252 endif
```

2.2、 fstab 中对 system 分区增加校验标志，参考修改：

device/sprd / scx35l/emmc/**fstab_dt.sc8830**

Patch Set Base 1 2 3 4 5 6 7 (gitweb)

```
1 id fstab file.
2 ilesystem that contains the filesystem checker binary (typically /system) cannot
3 fy MF_CHECK, and must come before any filesystems that do specify MF_CHECK
4                                   <mnt_point> <type> <mnt_flags and options>          <fs_mgr_fla
5
6 ock/platform/sdio_emmc/by-name/system   /system      ext4 ro,barrier=1                  wait
7 ock/platform/sdio_emmc/by-name/userdata /data        ext4 noatime,nosuid,nodev,nomblk_io_submit,nosauto_da_all
8 ock/platform/sdio_emmc/by-name/cache»     /cache       ext4 noatime,nosuid,nodev,nomblk_io_submit,nosauto_da_
9 ock/platform/sdio_emmc/by-name/prodnv   /productinfo ext4 noatime,nosuid,nodev,nomblk_io_submit,nosauto_da_all
10
11 s/sdio_sd/mmc_host/mmc1      auto         vfat defaults voldmanaged=sdcard0:auto:external,noemulatedsd
12 s/f5116000.usb/usb1/1-1/1-1:1.0   auto         vfat defaults voldmanaged=usbdisk:auto
```

File 3 of 4

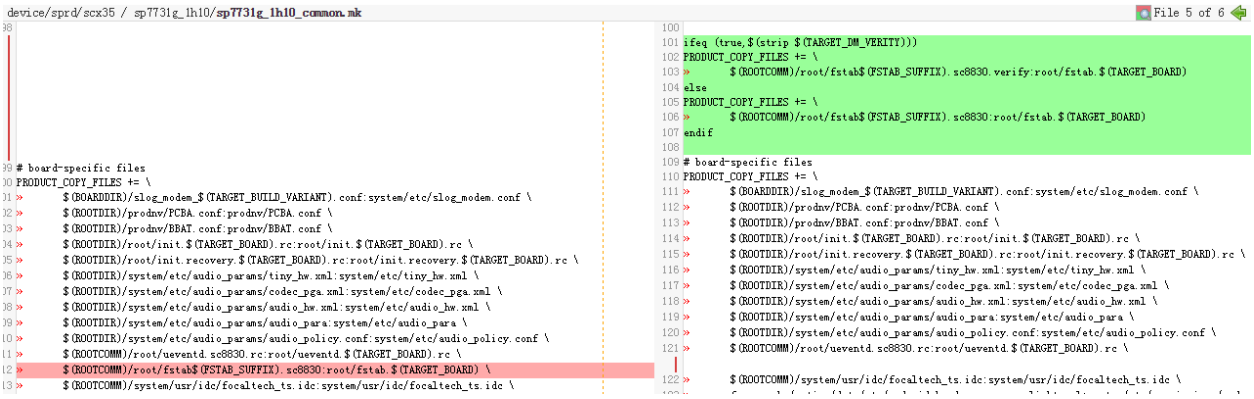
Patch Set 1 2 3 4 5 6 7 (gitweb)

```
1 id fstab file.
2 ilesystem that contains the filesystem checker binary (typically /system) cannot
3 fy MF_CHECK, and must come before any filesystems that do specify MF_CHECK
4                                   <mnt_point> <type> <mnt_flags and options>          <fs_mgr_fl
5
6 ock/platform/sdio_emmc/by-name/system   /system      ext4 ro,barrier=1                  wait,verify
7 ock/platform/sdio_emmc/by-name/userdata /data        ext4 noatime,nosuid,nodev,nomblk_io_submit,nosauto_da_al
8 ock/platform/sdio_emmc/by-name/cache»     /cache       ext4 noatime,nosuid,nodev,nomblk_io_submit,nosauto_da
9 ock/platform/sdio_emmc/by-name/prodnv   /productinfo ext4 noatime,nosuid,nodev,nomblk_io_submit,nosauto_da_al
10
11 s/sdio_sd/mmc_host/mmc1      auto         vfat defaults voldmanaged=sdcard0:auto:external,noemulatedsd
12 s/f5116000.usb/usb1/1-1/1-1:1.0   auto         vfat defaults voldmanaged=usbdisk:auto
```

但是这里我们为了兼容性，不做这么直接的暴力修改。换一种修改方式：

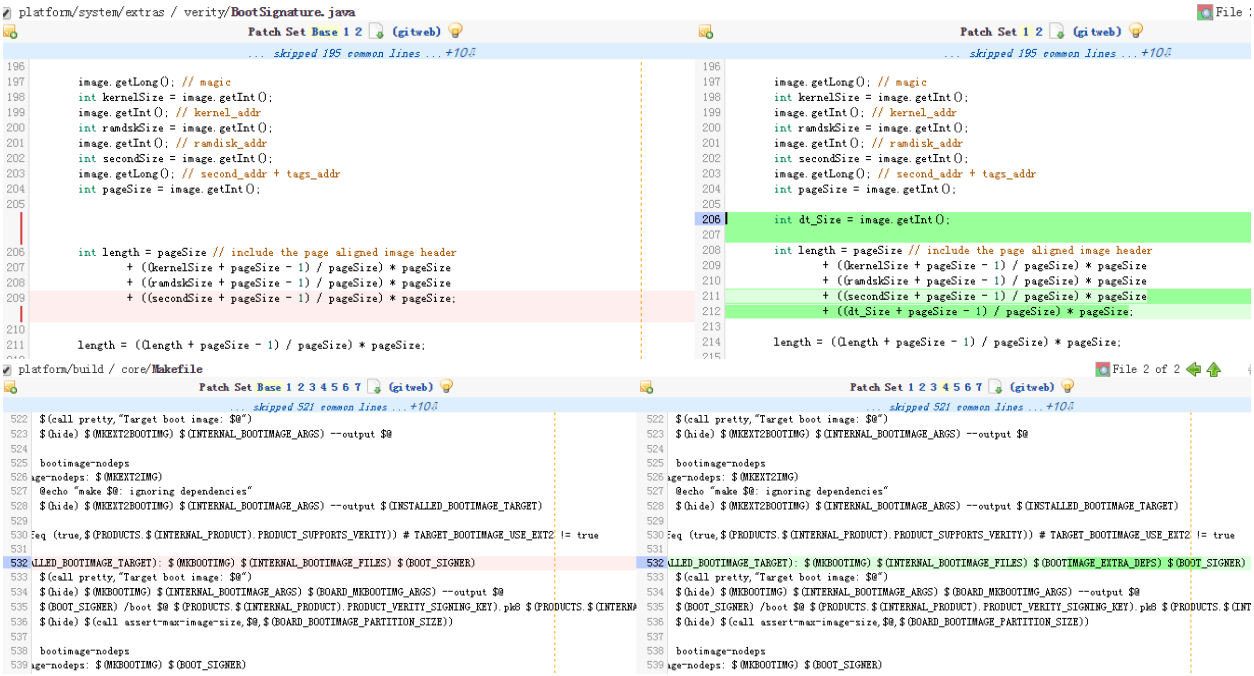
首先、增加一个 fstab 文件：**fstab.sc8830.verify**，这个文件其实就是 **fstab.sc8830** 拷贝过来，并参考上图所展示的差异进行修改。

其次、在对应的 Board 中通过 TARGET_DM_VERITY 宏来控制决定拷贝的 fstab 文件，参考修改：

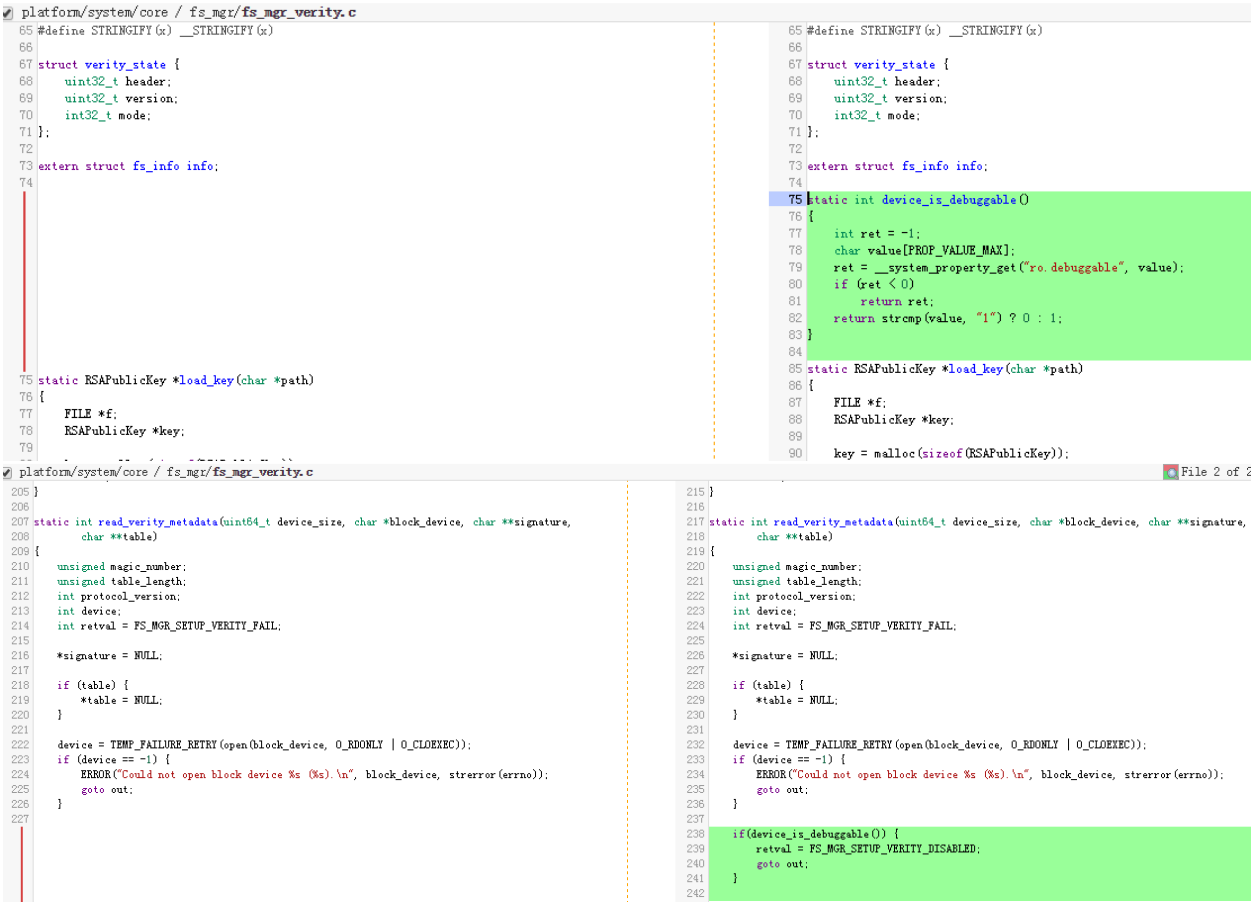


最后、在需要打开此功能的 Board 将 TARGET_DM_VERITY 置为 true。参考修改：

3、与原生 kernel 相比，sprd 的 kernel 中增加了 device tree 功能，因此需要修改对应的签名程序以及编译脚本，增加 dt 的相关信息，参考修改：



4、由于开启此功能会导致 system 分区不能被修改，在 userdebug 版本将不方便直接替换 APK 或者 SO 文件进行问题的 debug。因此此方案在 userdebug 版本不生效，仅在 user 版本生效。参考修改：



platform/system/core / fs_mgr/fs_mgr_verity.c

```
#define STRINGIFY(x) __STRINGIFY(x)

struct verity_state {
    uint32_t header;
    uint32_t version;
    int32_t mode;
};

extern struct fs_info info;

static RSAPublicKey *load_key(char *path)
{
    FILE *f;
    RSAPublicKey *key;
```

```
#define STRINGIFY(x) __STRINGIFY(x)

struct verity_state {
    uint32_t header;
    uint32_t version;
    int32_t mode;
};

extern struct fs_info info;

static int device_is_debuggable()
{
    int ret = -1;
    char value[PROP_VALUE_MAX];
    ret = __system_property_get("ro.debuggable", value);
    if (ret < 0)
        return ret;
    return strcmp(value, "1") ? 0 : 1;
}

static RSAPublicKey *load_key(char *path)
{
    FILE *f;
    RSAPublicKey *key;

    key = malloc(sizeof(RSAPublicKey));
```

platform/system/core / fs_mgr/fs_mgr_verity.c

```
}

static int read_verity_metadata(uint64_t device_size, char *block_device, char **signature,
        char **table)
{
    unsigned magic_number;
    unsigned table_length;
    int protocol_version;
    int device;
    int retval = FS_MGR_SETUP_VERITY_FAIL;

    *signature = NULL;

    if (table) {
        *table = NULL;
    }

    device = TEMP_FAILURE_RETRY(open(block_device, O_RDONLY | O_CLOEXEC));
    if (device == -1) {
        ERROR("Could not open block device %s (%s).\n", block_device, strerror(errno));
        goto out;
    }
```

```
}

static int read_verity_metadata(uint64_t device_size, char *block_device, char **signature,
        char **table)
{
    unsigned magic_number;
    unsigned table_length;
    int protocol_version;
    int device;
    int retval = FS_MGR_SETUP_VERITY_FAIL;

    *signature = NULL;

    if (table) {
        *table = NULL;
    }

    device = TEMP_FAILURE_RETRY(open(block_device, O_RDONLY | O_CLOEXEC));
    if (device == -1) {
        ERROR("Could not open block device %s (%s).\n", block_device, strerror(errno));
        goto out;
    }

    if(device_is_debuggable()) {
        retval = FS_MGR_SETUP_VERITY_DISABLED;
        goto out;
    }
```

完成上述配置后，需要做一次完整编译，并重新烧录 boot.img、 system.img、userdata.img，即可保证 dm-verity 功能在 user 版本生效。

6. Android7.0 中 SPRD 适配的修改

Android7.0 中针对 DM-VERITY 增加了一项新的子项功能：DM_VERITY_FEC，即 Verity forward error correction。由于此功能的引入，导致 4.7 中所述的 system.img 格式发生变化，从而校验方式也发生变化。这就要求 system.img 的大小和 xml 中定义 system 分区的大小需要保持一致，否则就会由于找不到校验数据，导致 system 分区无法挂载，从而出现无法开机的情况。

为了满足上述新功能需求，平台做了两笔功能性修改：

1、增加 System 分区自适应功能。通过 SYSTEM_IMAGE_SIZIE_ADAPT 宏来控制。当 SYSTEM_IMAGE_SIZIE_ADAPT 为 false 的时候，功能关闭，否则功能开启。

2、针对 System 分区特殊大小（比如 1600M），进行分区大小再调整的修改。具体方案为：当 System 分区自适应功能打开时，在编译脚本中自动进行分区调整；

当 system 分区自适应功能关闭，在编译阶段报错，并主动提示用户需要配置 BOARD_SYSTEMIMAGE_PARTITION_SIZE 的大小。

注意：特殊大小包含两个含义：

一是当 system 分区为 1600M 的时候，无法合适完整的分配 system 文件系统（以 A 表示）和校验数据的大小（以 B 表示），因为 B 的大小是跟随 A 的大小而变化的。此问题的点是：当 A 为 1651507200，B 计算出的大小为 26210304，总大小为 1677717504（刚好比 1600M 少一个 block，也就是 4096）；当 A 为 1651511296，B 计算出的大小为 26214400。总大小为 1677725696（刚好比 1600M 多一个 block，也就是 4096）其中 B 包含三部分：B1（GetVerityTreeSize）、B2（GetVerityMetadataSize）、B3（GetVerityFECSize）。

二是文件系统的大小有要求规则：假设传进来的镜像大小是 X，X 与 128M 求余，余数为 Y, 如果 0<Y<4M，则文件系统会把 Y 丢掉， 如果大于等于 4M，则文件系统不会丢 block。

平台默认开启 System 分区大小自适应功能。如需关闭该功能，则对 system 分区的大小进行调整时，需要同时修改 xml 中的 system 分区大小以及 BOARD_SYSTEMIMAGE_PARTITION_SIZE 的大小一致。同时，所修改的大小值需要为 50M 的倍数，并且排除 1600M 这个特殊值。

修改内容包含以下部分（这里以 9832 为例）：

1、Kernel 中将 dm 相关驱动配置打开。

kernel/common / arch/arm/configs/sp9832a_2h11_volte_defconfig

Patch Set Base 1 2 3 4 (gitweb)

```
1281 # CONFIG_BLK_DEV_MD is not set
1282 # CONFIG_BCACHE is not set
1283 CONFIG_BLK_DEV_DM_BUILTIN=y
1284 CONFIG_BLK_DEV_DM=y
1285 # CONFIG_DM_DEBUG is not set
1286 CONFIG_DM_CRYPT=y
1287 # CONFIG_DM_SNAPSHOT is not set
1288 # CONFIG_DM_THIN_PROVISIONING is not set
1289 # CONFIG_DM_CACHE is not set
1290 # CONFIG_DM_MIRROR is not set
1291 # CONFIG_DM_RAID is not set
1292 # CONFIG_DM_ZERO is not set
1293 # CONFIG_DM_MULTIPATH is not set
1294 # CONFIG_DM_DELAY is not set
1295 # CONFIG_DM_UEVENT is not set
1296 # CONFIG_DM_FLAKEY is not set
1297 # CONFIG_DM_VERITY is not set
1298 # CONFIG_TARGET_CORE is not set
1299 CONFIG_NETDEVICES=y
1300 CONFIG_NET_CORE=y
1301 # CONFIG_BONDING is not set
```

Patch Set 1 2 3 4 (gitweb)

```
1281 # CONFIG_BLK_DEV_MD is not set
1282 # CONFIG_BCACHE is not set
1283 CONFIG_BLK_DEV_DM_BUILTIN=y
1284 CONFIG_BLK_DEV_DM=y
1285 # CONFIG_DM_DEBUG is not set
1286 CONFIG_DM_BUFIO=y
1287 CONFIG_DM_CRYPT=y
1288 # CONFIG_DM_SNAPSHOT is not set
1289 # CONFIG_DM_THIN_PROVISIONING is not set
1290 # CONFIG_DM_CACHE is not set
1291 # CONFIG_DM_MIRROR is not set
1292 # CONFIG_DM_RAID is not set
1293 # CONFIG_DM_ZERO is not set
1294 # CONFIG_DM_MULTIPATH is not set
1295 # CONFIG_DM_DELAY is not set
1296 # CONFIG_DM_UEVENT is not set
1297 # CONFIG_DM_FLAKEY is not set
1298 CONFIG_DM_VERITY=y
1299 CONFIG_DM_VERITY_FEC=y
1300 # CONFIG_TARGET_CORE is not set
1301 CONFIG_NETDEVICES=y
1302 CONFIG_NET_CORE=y
1303 # CONFIG_BONDING is not set
```

2、修改 Board 相关的配置（这里以 7731g 为例）：

2.1、 /dev/sprd/scx35l/common/device.mk，增加 dm-verity 的编译配置，同时用 TARGET_DM_VERITY 宏控制。参考修改：

device/sprd/scx35l / common/device.mk

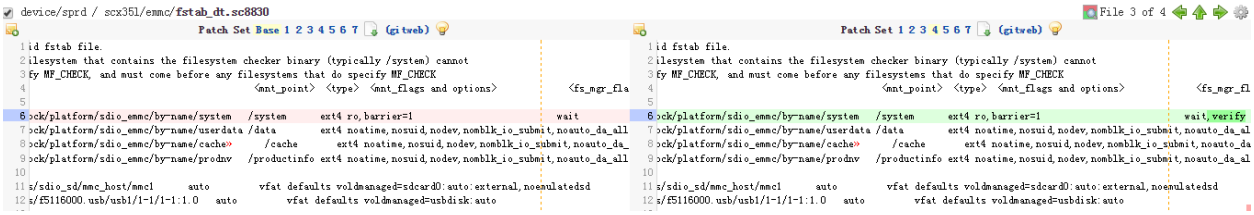
Patch Set Base 1 2 3 4 5 6 7 8 (gitweb)

```
... skipped 301 common lines ... +103
302 »       ro.product.ram=low
303 else
304 ifeq ($(strip $(PRODUCT_RAM)),high)
305 $(call inherit-product, frameworks/native/build/phone-xhdpi-1024-dalvik-heap.mk)
306 PRODUCT_PROPERTY_OVERRIDES += \
307 »       ro.board_ram_size=high \
308 »       ro.config.low_ram=false \
309 »       ro.product.ram=high
310 endif
311 endif
312 # @}
313
314 # add widevine build
315 PRODUCT_PACKAGES += \
316 »       libwvdrmengine
```

Patch Set 1 2 3 4 5 6 7 8 (gitweb)

```
... skipped 301 common lines ... +103
302 »       ro.product.ram=low
303 else
304 ifeq ($(strip $(PRODUCT_RAM)),high)
305 $(call inherit-product, frameworks/native/build/phone-xhdpi-1024-dalvik-heap.mk)
306 PRODUCT_PROPERTY_OVERRIDES += \
307 »       ro.board_ram_size=high \
308 »       ro.config.low_ram=false \
309 »       ro.product.ram=high
310 endif
311 endif
312 ifeq (true,$(strip $(TARGET_DM_VERITY)))
313 $(call inherit-product, build/target/product/verity.mk)
314 PRODUCT_SYSTEM_VERITY_PARTITION := /dev/block/platform/sdio_emmc/by-name/system
315 endif
316 # @}
317
318 # add widevine build
319 PRODUCT_PACKAGES += \
320 »       libwvdrmengine
```

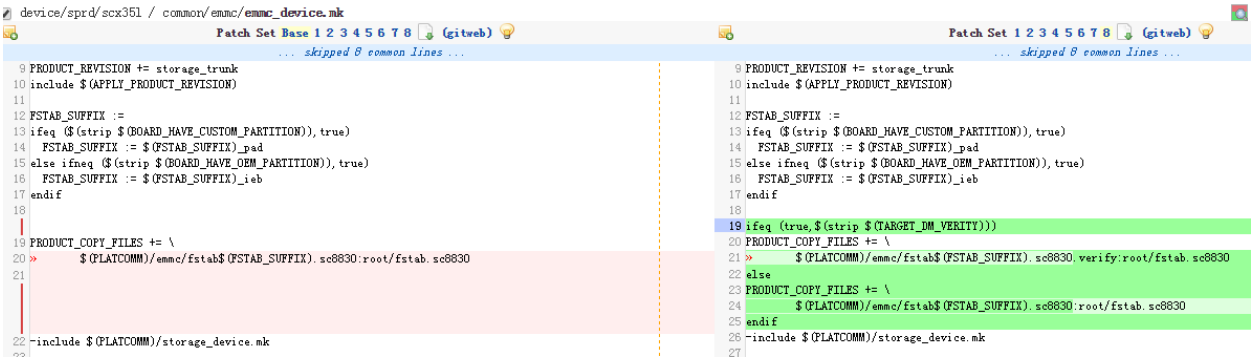
2.2、 fstab 中对 system 分区增加校验标志，参考修改：



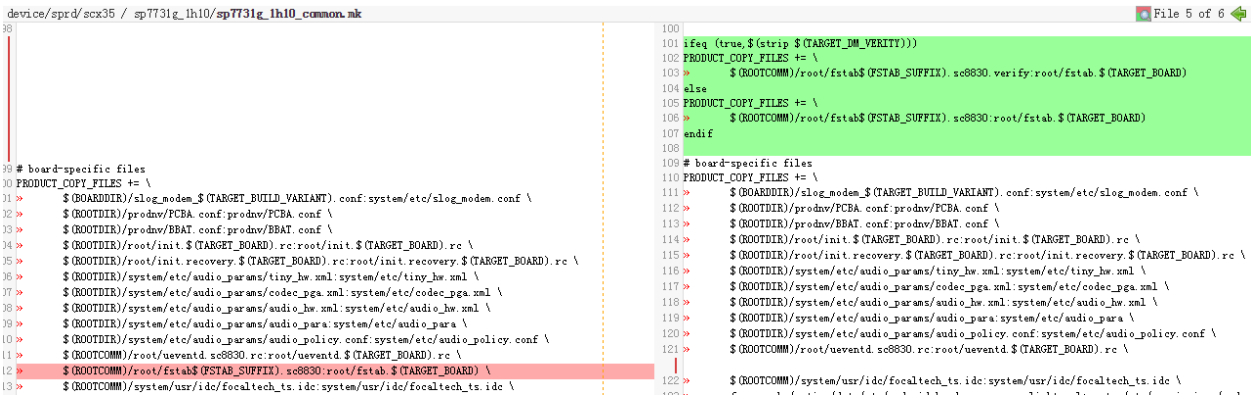
但是这里我们为了兼容性，不做这么直接的暴力修改。换一种修改方式：

首先、增加一个 fstab 文件：**fstab.sc8830.verify**，这个文件其实就是 **fstab.sc8830** 拷贝过来，并参考上图所展示的差异进行修改。

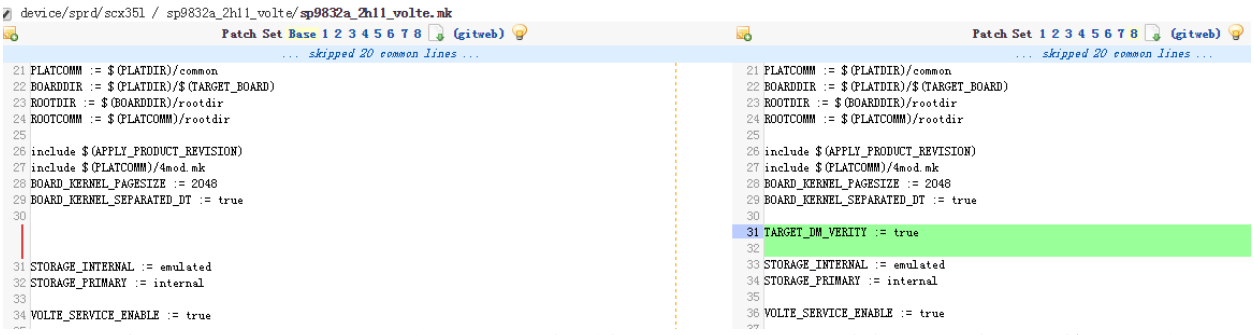
其次、在对应的 Board 中通过 TARGET_DM_VERITY 宏来控制决定拷贝的 fstab 文件，参考修改：



或者



最后、在需要打开此功能的 Board 将 TARGET_DM_VERITY 置为 true。参考修改：



3、与原生 kernel 相比，sprd 的 kernel 中增加了 device tree 功能，因此需要修改对应的签名程序以及编译脚本，增加 dt 的相关信息，参考修改：

platform/system/extras / verity/**BootSignature.java**

Patch Set Base 1 2 (gitweb)

```
... skipped 195 common lines ... +10
196
197         image.getLong(); // magic
198         int kernelSize = image.getInt();
199         image.getInt(); // kernel_addr
200         int ramdskSize = image.getInt();
201         image.getInt(); // ramdisk_addr
202         int secondSize = image.getInt();
203         image.getLong(); // second_addr + tags_addr
204         int pageSize = image.getInt();
205
206         int length = pageSize // include the page aligned image header
207                 + ((kernelSize + pageSize - 1) / pageSize) * pageSize
208                 + ((ramdskSize + pageSize - 1) / pageSize) * pageSize
209                 + ((secondSize + pageSize - 1) / pageSize) * pageSize;
210
211         length = (length + pageSize - 1) / pageSize) * pageSize;
```

Patch Set 1 2 (gitweb)

```
... skipped 195 common lines ... +10
196
197         image.getLong(); // magic
198         int kernelSize = image.getInt();
199         image.getInt(); // kernel_addr
200         int ramdskSize = image.getInt();
201         image.getInt(); // ramdisk_addr
202         int secondSize = image.getInt();
203         image.getLong(); // second_addr + tags_addr
204         int pageSize = image.getInt();
205
206         int dt_Size = image.getInt();
207
208         int length = pageSize // include the page aligned image header
209                 + ((kernelSize + pageSize - 1) / pageSize) * pageSize
210                 + ((ramdskSize + pageSize - 1) / pageSize) * pageSize
211                 + ((secondSize + pageSize - 1) / pageSize) * pageSize
212                 + ((dt_Size + pageSize - 1) / pageSize) * pageSize;
213
214         length = (length + pageSize - 1) / pageSize) * pageSize;
```

platform/build / core/**Makefile**

File 2 of 2

Patch Set Base 1 2 3 4 5 6 7 (gitweb)

```
... skipped 521 common lines ... +10
522 $(call pretty,"Target boot image: $@")
523 $(hide) $(MKEXT2BOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) --output $@
524
525 .bootimage-nodeps
526 age-nodeps: $(MKEXT2IMG)
527     @echo "make $@: ignoring dependencies"
528     $(hide) $(MKEXT2BOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) --output $(INSTALLED_BOOTIMAGE_TARGET)
529
530 eq (true,$(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_SUPPORTS_VERITY)) # TARGET_BOOTIMAGE_USE_EXT2 != true
531
532 LLED_BOOTIMAGE_TARGET): $(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_FILES) $(BOOT_SIGNER)
533     $(call pretty,"Target boot image: $@")
534     $(hide) $(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) $(BOARD_MKBOOTIMG_ARGS) --output $@
535     $(BOOT_SIGNER) /boot $@ $(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_VERITY_SIGNING_KEY).pk8 $(PRODUCTS.$(INTERNA
536     $(hide) $(call assert-max-image-size,$@,$(BOARD_BOOTIMAGE_PARTITION_SIZE))
537
538 .bootimage-nodeps
539 age-nodeps: $(MKBOOTIMG) $(BOOT_SIGNER)
```

Patch Set 1 2 3 4 5 6 7 (gitweb)

```
... skipped 521 common lines ... +10
522 $(call pretty,"Target boot image: $@")
523 $(hide) $(MKEXT2BOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) --output $@
524
525 .bootimage-nodeps
526 age-nodeps: $(MKEXT2IMG)
527     @echo "make $@: ignoring dependencies"
528     $(hide) $(MKEXT2BOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) --output $(INSTALLED_BOOTIMAGE_TARGET)
529
530 eq (true,$(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_SUPPORTS_VERITY)) # TARGET_BOOTIMAGE_USE_EXT2 != true
531
532 LLED_BOOTIMAGE_TARGET): $(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_FILES) $(BOOTIMAGE_EXTRA_DEPS) $(BOOT_SIGNER)
533     $(call pretty,"Target boot image: $@")
534     $(hide) $(MKBOOTIMG) $(INTERNAL_BOOTIMAGE_ARGS) $(BOARD_MKBOOTIMG_ARGS) --output $@
535     $(BOOT_SIGNER) /boot $@ $(PRODUCTS.$(INTERNAL_PRODUCT).PRODUCT_VERITY_SIGNING_KEY).pk8 $(PRODUCTS.$(INT
536     $(hide) $(call assert-max-image-size,$@,$(BOARD_BOOTIMAGE_PARTITION_SIZE))
537
538 .bootimage-nodeps
539 age-nodeps: $(MKBOOTIMG) $(BOOT_SIGNER)
```

4、 由于开启此功能会导致 system 分区不能被修改，因此在 userdebug 版本将不方便直接替换 APK 或者 SO 文件进行问题的 debug。因此此方案在 userdebug 版本不生效，仅在 user 版本生效。参考修改：

platform/system/core / fs_mgr/**fs_mgr_verity.cpp**

Patch Set Base 1 2 (gitweb)

```
... skipped 108 common lines ... +10
109         ERROR("Invalid key length %d\n", key->len);
110         fclose(f);
111         free(key);
112         return NULL;
113     }
114
115     fclose(f);
116     return key;
117 }
118
119 static int verify_table(const uint8_t *signature, const char *table,
120         uint32_t table_length)
121 {
122     RSAPublicKey *key;
123     uint8_t hash_buf[SHA256_DIGEST_SIZE];
124     int retval = -1;
```

Patch Set 1 2 (gitweb)

```
... skipped 108 common lines ... +10
109         ERROR("Invalid key length %d\n", key->len);
110         fclose(f);
111         free(key);
112         return NULL;
113     }
114
115     fclose(f);
116     return key;
117 }
118
119 static int device_is_debuggable()
120 {
121     int ret = -1;
122     char value[PROP_VALUE_MAX];
123     ret = __system_property_get("ro.debuggable", value);
124     if (ret < 0)
125         return ret;
126     return strcmp(value, "1") ? 0 : 1;
127 }
128
129 static int verify_table(const uint8_t *signature, const char *table,
130         uint32_t table_length)
131 {
132     RSAPublicKey *key;
133     uint8_t hash_buf[SHA256_DIGEST_SIZE];
134     int retval = -1;
```

platform/system/core / fs_mgr/**fs_mgr_verity.cpp**

```
125
126     // Hash the table
127     SHA256_hash((uint8_t*)table, table_length, hash_buf);
128
+10... skipped 739 common lines ... +10
868     char *invalid_table = NULL;
869     char *verity_blk_name = NULL;
870     struct fec_handle *f = NULL;
871     struct fec_verity_metadata verity;
872     struct verity_table_params params;
873
874     alignas(dm_ioctl) char buffer[DM_BUF_SIZE];
875     struct dm_ioctl *io = (struct dm_ioctl *) buffer;
876     char *mount_point = basename(fstab->mount_point);
877
878     if (fec_open(&f, fstab->blk_device, O_RDONLY, FEC_VERITY_DISABLE,
879             FEC_DEFAULT_ROOTS) < 0) {
880         ERROR("Failed to open '%s' (%s)\n", fstab->blk_device,
881             strerror(errno));
882         return retval;
883     }
```

```
135
136     // Hash the table
137     SHA256_hash((uint8_t*)table, table_length, hash_buf);
138
+10... skipped 739 common lines ... +10
878     char *invalid_table = NULL;
879     char *verity_blk_name = NULL;
880     struct fec_handle *f = NULL;
881     struct fec_verity_metadata verity;
882     struct verity_table_params params;
883
884     alignas(dm_ioctl) char buffer[DM_BUF_SIZE];
885     struct dm_ioctl *io = (struct dm_ioctl *) buffer;
886     char *mount_point = basename(fstab->mount_point);
887
888     if(device_is_debuggable()) {
889         retval = FS_MGR_SETUP_VERITY_DISABLED;
890         goto out;
891     }
892
893     if (fec_open(&f, fstab->blk_device, O_RDONLY, FEC_VERITY_DISABLE,
894             FEC_DEFAULT_ROOTS) < 0) {
895         ERROR("Failed to open '%s' (%s)\n", fstab->blk_device,
896             strerror(errno));
897         return retval;
898     }
```

完成上述配置后，需要做一次完整编译，并重新烧录 boot.img、 system.img、userdata.img，dm-verity，即可保证 dm-verity 功能在 user 版本生效。

7. 验证 DM-VERITY 功能

验证功能是否成功开启的办法：查看 system 分区的挂载方式，如下图所示，当挂载方式为 dm-x 时，表明功能开启成功，反之则是未开启。

```
SPREADTRUM\zhongjie.liu@sh01428pcu:~$ adb shell mount
rootfs / rootfs ro,seclabel,size=369544k,nr_inodes=92386 0 0
tmpfs /dev tmpfs rw,seclabel,nosuid,relatime,mode=755 0 0
devpts /dev/pts devpts rw,seclabel,relatime,mode=600 0 0
proc /proc proc rw,relatime 0 0
sysfs /sys sysfs rw,seclabel,relatime 0 0
selinuxfs /sys/fs/selinux selinuxfs rw,relatime 0 0
none /acct cgroup rw,relatime,cpuacct 0 0
none /sys/fs/cgroup tmpfs rw,seclabel,relatime,mode=750,gid=1000 0 0
tmpfs /mnt tmpfs rw,seclabel,relatime,mode=755,gid=1000 0 0
none /dev/cpuctl cgroup rw,relatime,cpu 0 0
/dev/block/dm-0 /system ext4 ro,seclabel,relatime,data=ordered 0 0
/dev/block/platform/soc/soc:ap-ahb/50430000.sdio/by-name/userdata /data ext4 rw,seclabel,nosuid,nodev,noatime,noauto_da_alloc,data=ordered
/dev/block/platform/soc/soc:ap-ahb/50430000.sdio/by-name/cache /cache ext4 rw,seclabel,nosuid,nodev,noatime,noauto_da_alloc,data=ordered 0
/dev/block/platform/soc/soc:ap-ahb/50430000.sdio/by-name/prodnv /productinfo ext4 rw,seclabel,nosuid,nodev,noatime,noauto_da_alloc,data=or
adb /dev/usb-ffs/adb functionfs rw,relatime 0 0
tmpfs /storage tmpfs rw,seclabel,relatime,mode=755,gid=1000 0 0
/dev/fuse /mnt/runtime/default/emulated fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,group_id=1023,default_permissions,allow_other 0 0
/dev/fuse /storage/emulated fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,group_id=1023,default_permissions,allow_other 0 0
/dev/fuse /mnt/runtime/read/emulated fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,group_id=1023,default_permissions,allow_other 0 0
/dev/fuse /mnt/runtime/write/emulated fuse rw,nosuid,nodev,noexec,noatime,user_id=1023,group_id=1023,default_permissions,allow_other 0 0
```

8. 对性能影响及关闭方法

DM-VERITY 功能开启后对性能有一定的影响。根据它的工作原理，对性能的影响主要体现在两个方面：

1、开机时候在挂载 system 分区前需要对它的 verity metadata 数据进行校验，这部分耗时预计在 2s 左右。

2、任何时候从 system 分区读取的数据都需要和 hash tree 进行匹配，这就会导致读取数据的耗时会增加。

DM-VERITY 功能是属于安全的范畴，功能的开启必然会对性能有一定的影响。如在某项目上此功能非必须，可通过将 TARGET_DM_VERITY 置为 false 来关闭。验证方法见“验证 DM-VERITY 功能”。

9. 常见问题

9.1. Key 配置说明

9.1.1. Key 文件作用说明

Dm-Verity 包含三个 key 文件，路径位于：build/target/product/security/，具体作用为：

verity.pk8 – private key used to sign boot.img and system.img

verity.x509.pem – certificate include public key

verity_key – public key used in dm verity for system.img

9.1.2. 替换原生 key 说明

1 替换步骤：

第一步：生成自定义的 verity 相关的三个 key 文件：verity.pk8、verity.x509.pem、verity_key。

1、在 Linux 系统中，确保所安装的 openssl 版本足够新。（ubuntu 终端输入“openssl version”查看版本号，比如：OpenSSL 1.0.1f 6 Jan 2014 是没有问题的）。

2、终端切换到 idh 代码的根目录下，输入如下命令（直接回车，不用输入密码），就会在当前根目录生成 verity.pk8 和 verity.x509.pem：

development/tools/make_key verity '/C=US/ST=California/L=Mountain View/O=Android/OU=Android/CN=Android/emailAddress=android@android.com'

3、在终端中执行 source build/envsetup.sh、lunch 选择对应的工程、kheader 后，再输入 make generate_verity_key 或者 mmm system/extras/verity/，就会生成 generate_verity_key。

4、在终端继续输入如下命令，将会在 idh.code 根目录生成 verity_key.pub：

out/host/linux-x86/bin/generate_verity_key -convert verity.x509.pem verity_key

5、将 verity_key.pub 重命名为 verity_key。

第二步：将上述生成的三个 key 替换 build/target/product/security/目录下同名的三个 key。

第三步：重新进行完整版本编译。

2 验证 key 是否替换成功的方法：

2.1 Verity_key 对比验证

verity_key 最后会被打包进 boot.img 中，编译时生成的目录位于（以 9832e 为例）：out/target/product/sp9832e_1h10/root。对比上述目录下的 verity_key 和前述用命令生成的 verity_key，如果两者一致则说明替换成功。

2.2 交叉验证

基于同一软件版本，分别使用 A 和 B 两组不同的 key 编译得到 A 和 B 两个完整版本（确保仅 verity 的 key 不同）。验证步骤如下：

1、下载版本 A 至手机中，开机并按照“验证 DM-VERITY 功能”说明，确认 DM-VERITY 功能开启正常。预期结果：手机开机正常；system 分区按照 dm-0 的方式挂载。

2、下载替换版本 B 的 system.img，开机验证。预期结果：手机无法开机。

3、继续下载替换版本 B 的 boot.img，开机验证。预期结果：手机开机正常；system 分区按照 dm-0 的方式挂载。

9.2. 无法开机

DM-VERITY 功能在平台版本上适配 OK，一般不会出现问题，一旦出现问题，现象就是无法开机。可以按照如下几个方面进行分析：

1、按照平台版本，检查本文档中“Android6.0 中 SPRD 适配的修改”或“Android7.0 中 SPRD 适配的修改”的各项是否正确，尤其要注意 Board 的适配

2、检查编译 log，确认是否编译出 build_verity_tree 和 build_verity_metadata 信息

3、串口抓取开机 kernel log，检查 system 分区是否挂载正常，关键字：fs_mgr

4、针对 android7.0，需要确认 pac 包中 system.img 的大小和 xml 分区表上 system 分区的大小是否一致。

10. 安全性建议

Android 原生设计中，针对开启 dm-verity 功能的版本，root 之后，可以通过 adb 命令开关 dm-verity 功能，具体命令如下：

$ adb disable-verity //关闭 dm-verity

$ adb enable-verity //开启 dm-verity

出于安全性考虑，建议版本中移除对上述两条命令的支持，防止破解者通过 adb 关闭 dm-verity 功能。

另外，根据“5. Android6.0 中 SPRD 适配的修改”和“6. Android7.0 中 SPRD 适配的修改”两章中第 4 部分适配修改的说明，debug 版本默认关闭了 dm-verity 功能。

为确保整体的安全性，建议在整个研发生命周期内将 debug 版本的 dm-verity 功能开启，并关闭 adb disable verify 功能的操作。

Debug 版本开启 dm-verity 功能的方法为：根据 android 版本的不同，参考对应的章节，将第 4 部分的修改移除即可。

注意：

Debug 版本打开 dm-verity 后，如果单独替换 so 或者 apk 文件等导致 system 分区改变，会出现无法正常开机的情况，因此在 debug 过程中需要修改 system 分区时，必须要重新编译 system 分区进行验证。