PRPL-FLR001

prpl Standard Flash Layout

REVISION HISTORY

Version	Date	Authors	Description
0.0	May 2022	David Cluytens, Vincent Harlé	Initial Confluence version
0.1	July 2023	Onur Zengin, Matthias Hoffman	Generic architecture version, import to MS word
0.2	August 2023	Brendan Black	update template / formatting, minor edits
0.3	October 2023	Vincent Harlé	Added §8 Appendixes with additional explanations Updated diagrams
1.0	February 2024		Title changed, Initial Public Release

1 Contents

2 Important notices, IPR statement, disclaimer and Copyright

This chapter contains important information about PRPL and this document (hereinafter 'This PRPL Document').

2.1 About PRPL

The prpl Foundation (PRPL) is a not-for-profit organization which publishes documents including, but not limited to, Requirements, Specifications, Recommendations, API application programming interfaces, and Test Plans.

2.2 This may not be the latest version of This PRPL Document

This PRPL Document is the output of the Working Groups of the PRPL and its members as of the date of publication. Readers of This PRPL Document should be aware that it can be revised, edited or have its status changed according to the PRPL working procedures.

2.3 There is no warranty provided with This PRPL Document

The services, the content and the information in This PRPL Document are provided on an "as is" basis. PRPL, to the fullest extent permitted by law, disclaims all warranties, whether express, implied, statutory or otherwise, including but not limited to the implied warranties of merchantability, non-infringement of third-parties rights and fitness for a particular purpose. PRPL, its affiliates and licensors make no representations or warranties about the accuracy, completeness, security or timeliness of the content or information provided in the PRPL Document. No information obtained via the PRPL Document shall create any warranty not expressly stated by PRPL in these terms and conditions.

2.4 Exclusion of Liability

Any person holding a copyright in This PRPL Document, or any portion thereof, disclaims to the fullest extent permitted by law (a) any liability (including direct, indirect, special, or consequential damages under any legal theory) arising from or related to the use of or reliance upon This PRPL Document; and (b) any obligation to update or correct this technical report.

2.5 This PRPL Document is not binding on PRPL nor its member companies

This PRPL Document, though formally approved by the PRPL member companies, is not binding in any way upon the PRPL members.

2.6 Intellectual Property Rights

Patents essential or potentially essential to the implementation of features described in This PRPL Document may have been declared in conformance to the PRPL IP Policy (available at the PRPL website: www.prplFoundation.org).

2.7 Copyright Provisions

This PRPL Document is copyrighted by PRPL, and all rights are reserved. The contents of This PRPL Document are protected by the copyrights of PRPL or the copyrights of third-parties that are used by agreement. Trademarks and copyrights mentioned in This PRPL Document are the property of their respective owners. The content of This PRPL Document may only be reproduced, distributed, modified, framed, cached, adapted or linked to, or made available in any form by any means, or incorporated into or used in any information storage and retrieval system, **with the prior written permission of PRPL or the applicable third-party copyright owner**. Such written permission is **not** however required under the conditions specified in Section 2.7.1 and Section 2.7.2:

2.7.1 Incorporating PRPL Documents in Whole or Part Within Documents Related to Commercial Tenders

Any or all section(s) of PRPL Documents may be incorporated into Commercial Tenders (RFP, RFT, RFQ, ITT, etc.) by PRPL and non-PRPL members under the following conditions:

(a) The PRPL Requirements numbers, where applicable, must not be changed from those within the PRPL Documents;
(b) A prominent acknowledgement of the PRPL must be provided within the Commercial document identifying any and all PRPL Documents referenced and giving the web address of the PRPL;
(c) The Commercial Tender must identify which of its section(s) include material taken from PRPL Documents and must identify each PRPL Document used, and the relevant PRPL Section Numbers; and,
(d) The Commercial Tender must refer to the copyright provisions of PRPL Documents and must state that the sections taken from PRPL Documents are subject to copyright by PRPL and/or applicable third parties.

2.7.2 Copying This PRPL Document in its Entirety

This PRPL Document may be electronically copied, reproduced, distributed, linked to, or made available by other means, or incorporated into or used in any information storage and retrieval system, but **only in**

its original, unaltered PDF format, and with its original PRPL title and file name unaltered. It may not be modified without the advanced written permission of the PRPL.

3 Abbreviations and Acronyms

Name	Description
U-Boot	A.K.A Prpl Compliant Boot Loader. **U-Boot** versions that are later than v2022.07. Starting with this version, U-Boot integrates requirements for secure boot, firmware update and refurbishment functionality into a single bootloader.
UBI	Unsorted Block Images. A Linux kernel flash handling layer that manages wear leveling and bad blocks
BBT	Bad Block Table, is a table maintained by the flash memory controller that keeps track of any bad or defective blocks on the flash drive.
DTB	Device Tree Blob, a binary file that describes the hardware configuration of a system in a standardized way. It is used to provide a standardized and flexible way for hardware vendors to describe the hardware of their devices to the operating system.
eMMC	Embedded MultiMediaCard, is a type of flash storage that has a controller built in and manages wear leveling by itself. It provides a cost-effective and compact storage solution.
MTD	Memory Technology Device. An abstraction layer in Linux for managing and accessing memory devices.
GPT	GUID Partitioning Table. A standard for the partition tables of a storage device.
RoT	Root of Trust. An immutable asset with which a bootloader can ensure at least the first flash binary of Secure Boot is unmodified and authentic.
CoT	Chain-of-Trust. A chain of images where verification and authentication for the next stage is performed by the executable running at the time.
FIT	Flattened Image Table. An image table format that provides flexible image handling.
TEE	Trusted Execution Environment
SVN	Secure Version Number. A version number to identify security version of the firmware
prpl / PRPL	(The) prpl Foundation

4 Introduction

Prpl platforms utilize flash memory for non-volatile mutable data storage. There are two popular types of flash that are typically used by platform owners (operators / OEMs / ODMs). Depending on cost and other constraints and considerations, either unmanaged NAND flash, or eMMC – NAND flash plus controller, are chosen. Unmanaged flash systems such as NAND flash require dedicated management for wear-leveling and bad-block handling. Unsorted Block Image (UBI) is a type of Memory Technology Device (MTD) specialized for UBI File Systems (UBIFS) which provide features such as flash-stored indexing and write-caching. eMMC has a flash controller built in which performs such operations, thereby offloading the host from that responsibility.

This document aims to provide a common understanding of flash memory layout design and optimization strategies for prpl's use cases. To achieve this, we list the structure and contents of UBI volumes in prpl-compliant systems, and compare it to the layout of eMMC flash memory. We will also list the role of components in the boot process to show how the layout can be optimized for better performance.
The focus in this document is on the flash layout for U-Boot, and any components which are booted after U-Boot, to avoid having to cover variations caused by different silicon designs. Silicon-specific information will be provided by the silicon suppliers.

Please note that in this document we refer to image verification and authentication interchangeably. Secure boot stages must perform these operations together; integrity of an image implicitly requires its authenticity at the same process.

4.1 SECURE BOOT STEPS

A generic secure boot process relies on a hardware Root of Trust (RoT). An immutable code block verifies and authenticates the next bootloader stages leveraging the RoT. This ensures the Chain of Trust (CoT) is not modified without authorization. A gateway may include a number of earlier bootloader stages prior to U-Boot. The secure boot flow steps after successful security checks are listed below:

- Kernel and rootfs may be bundled together or be separate images. U-Boot verifies the next stage.
    - When kernel and rootfs are bundled together, rootfs will be verified by U-Boot alongside the kernel before execution is passed to the kernel.
    - When rootfs is a separate image: once the kernel runs, it verifies and/or decrypts the root filesystem utilizing solutions such as dm-crypt and dm-verity.
- U-Boot verifies the DTB to ensure it remains unmodified. DTB can be part of the FIT image, which can be verified at the same time as kernel verification.
- U-Boot puts the DTB into memory to provide the kernel access to it. If there is an initrd available in the system resources, U-Boot also verifies and loads the ramdisk into the memory.
- U-Boot then hands over the execution to the kernel.

- The kernel applies platform-specific security policies and optionally checks the integrity of subcomponents.
- Finally, the firmware initializes the user-space and prpl LCM components, then verifies the authenticity of other critical components.

For modularity reasons, a gateway may employ more than one DTB. The DTB to be used is chosen by the U-Boot parameters and the DTB is verified as a part of the image that it is included in.

Please note that we do not address bare-metal virtualization-based architectures in the initial release of this document. If the use case arises, we will address this case separately. Also, any architecture-specific details that may be missing are included in the designated documentation pages for each specific architecture which we listed in the Introduction section.

The following sections address the flow outlined above.

4.2 U-BOOT FUNCTIONAL PROPERTIES

U-Boot is part of the CoT. Hence, it needs to support capability to facilitate signature verification of consequent images in the chain unless those images have already been verified by earlier boot stages. Decryption can be optionally supported for when the images loaded are not in cleartext. This verification functionality requires access to high value assets such as K_RoT<PRPL-SB001 >.

In addition to secure boot functionalities, U-Boot may be required to have image upgrade capabilities, e.g., for refurbishment of CPE. Multiple copies of authentic images that are accessible both in successfully executed verification chain, and in case of the failed verification chain, are required. Requirements related to these copies are addressed in prpl Secure Boot Requirements <PRPL-SB001>. Generic layouts that could be used as models of how these copies are laid out on either an unmanaged NAND flash or an eMMC flash are listed in the next section.

For software upgrade functionality the gateway may utilize the K_RoT or another key asset to authenticate the received image prior to writing the image into the target flash partition. In addition to authenticity, other checks such as Secure Version Number check (SVN) can optionally be performed to ensure the image to be flashed is considered to be secure. This authentication prior to writing into flash can be done over the entire image or separate blocks depending on the system resources.

Refurbishment operation is an operator- / OEM-specific operation that can be done anywhere in the field or in an access-restricted facility. The update functionality may be performed directly to the flash. For increased security, an additional update partition or bank can be allocated in the flash for temporarily storing the image, before writing it to the actual partition. Details are implementation specific.

During any of the Secure Boot operations, U-Boot may utilize Trusted Execution Environment (TEE) features. This could be either to provide secure access to assets or performing certain cryptographic operations as part of the secure boot flow. U-Boot should be able to delegate trusted operations to TEE.

5 Partition Layout

5.1 MULTIPLE STAGE LOADERS

The reason for having more than one stage of boot loaders is to ensure that the boot process is as reliable and available as possible. Bootloaders earlier to U-Boot are simple and small pieces of software that can initialize the hardware fast, perform a basic health check, validate and load other bootloaders into memory. U-Boot performs more complicated tasks, such as loading drivers and initializing peripherals, to ensure that the operating system can be started appropriately.

5.2 OTHER BOOTLOADERS

Embedded architectures are expected to show different properties and features. These differences may lead to the boot process being segmented into various smaller bootloaders. Each of these bootloaders perform specific steps of the hardware initialization and bring-up sequence. U-Boot itself can be handled as a sub segment of a larger bootloader blob as well.

According to the requirements listed in prpl Secure Boot Requirements document <PRPL-SB001> the scope of this document does not address any bootloader stages prior to U-Boot as long as the layout is handled in a prpl compliant way.

5.3 FLASH LAYOUT

We provide a generic flash layout for each of the two possible flash use cases. In the first part of this section, we study the generic layout for an un-managed NAND flash device. In the second section we illustrate the generic layout for an eMMC managed device. A generic flash layout is expected to include the partitions/images listed below and is described in more details in the annex 8.1:

- Earlier bootloaders (Out of scope)
- U-Boot , its inactive Copy and U-Boot environment
- Manufacturing Data
- Kernel, Rootfs and their inactive Copies.
- Rootfs-data for development purposes.

- Secure Storage. This partition can serve for other secure objects and trusted storage implementations. It has its standalone security controls to provide security assurance. Secure boot does not verify this partition.
- And finally, LCM data where container and user space application binaries can be managed separately.

Kernel / Rootfs is shown separately in the diagram, however it is possible to have these partitions combined depending on the operators' upgrade strategy. Prpl does not enforce any requirement on these images being separate or together.

The gateway may utilize two separate implementations pre-rootfs loading. Init ramdisk (initrd) and initramfs. Either of these can be chosen by OEMs. Where initrd is preferred, it must be compliant with the secure boot flow employing relevant security controls and best practices that prpl addresses. Where initramfs is preferred, it is considered part of the kernel image.

Prpl recommends Flattened uImage Table <FIT> for the marked images for FITs flexible design and security features. Please see the relevant section in <PRPL-SBW001>.

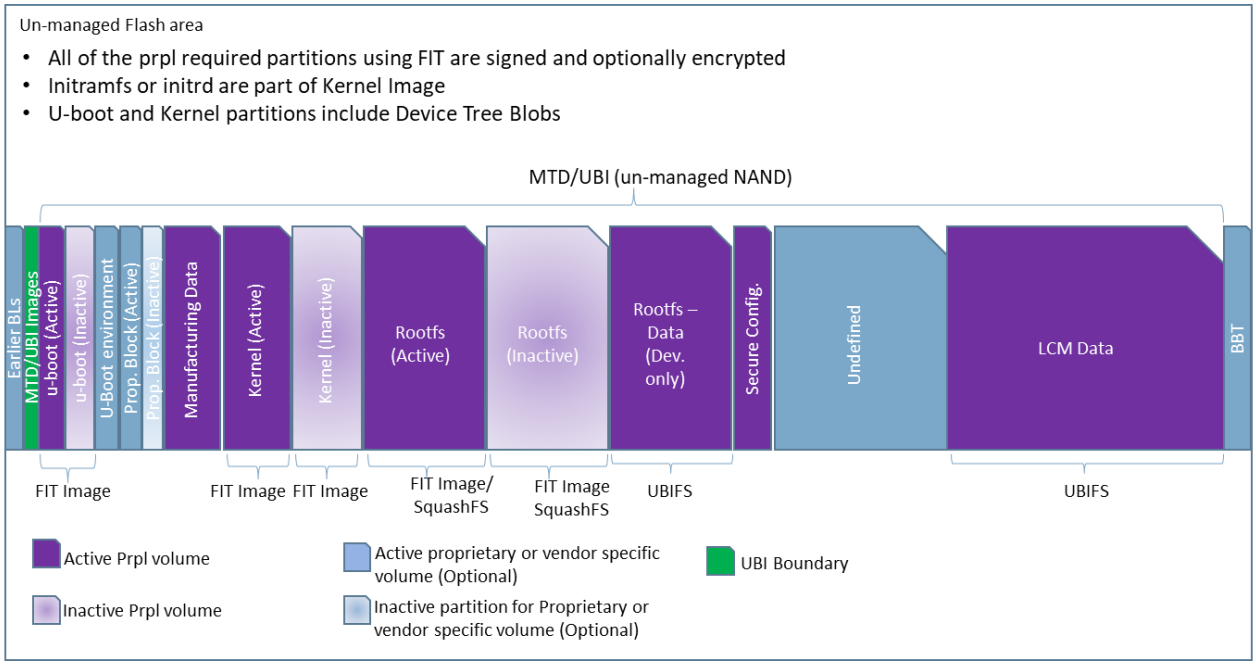
Prpl encourages the use of encryption on the images that are in CoT. U-Boot will verify and decrypt the kernel and the rootfs. Alternatively, the kernel can use dm-verity and dm-crypt as a standard way if the later images in the chain are to be authenticated and decrypted (block-by-block).

In the following layouts, we placed a Secure Storage before Kernel and Rootfs related partitions so that there is room for OEMs to size these partitions according to business needs. These partitions require a fixed maximum size in order to avoid accidentally overwriting other images in the flash. Secure storage partitions are encrypted and signed based on their own security policies and storage object structures. We do not enforce additional security controls over these partitions.

The partition marked as Rootfs-data is only for development firmware. Production Firmware in the field should not include this partition.

5.4 UBI UN-MANAGED NAND FLASH

Un-managed flash requires management against issues such as flash wear-out and bad block management. The illustration below allocates a Bad Block Table (BBT) at the end of the layout to have the registered persistently in the Flash. Prpl recommends using Unsorted Block Images (UBI) to have a standardized way of mitigating risks related to these issues.



UBI is specifically designed for flash memory devices. Earlier bootloaders are not considered part of the UBI device because they are only responsible to interpret how U-Boot is formatted with UBI standard. Therefore, earlier bootloaders chain the bootloaders up until U-Boot from a raw flash device, bypassing the UBI layer entirely.

UBI volumes can be dynamically sized to exactly fit each image.

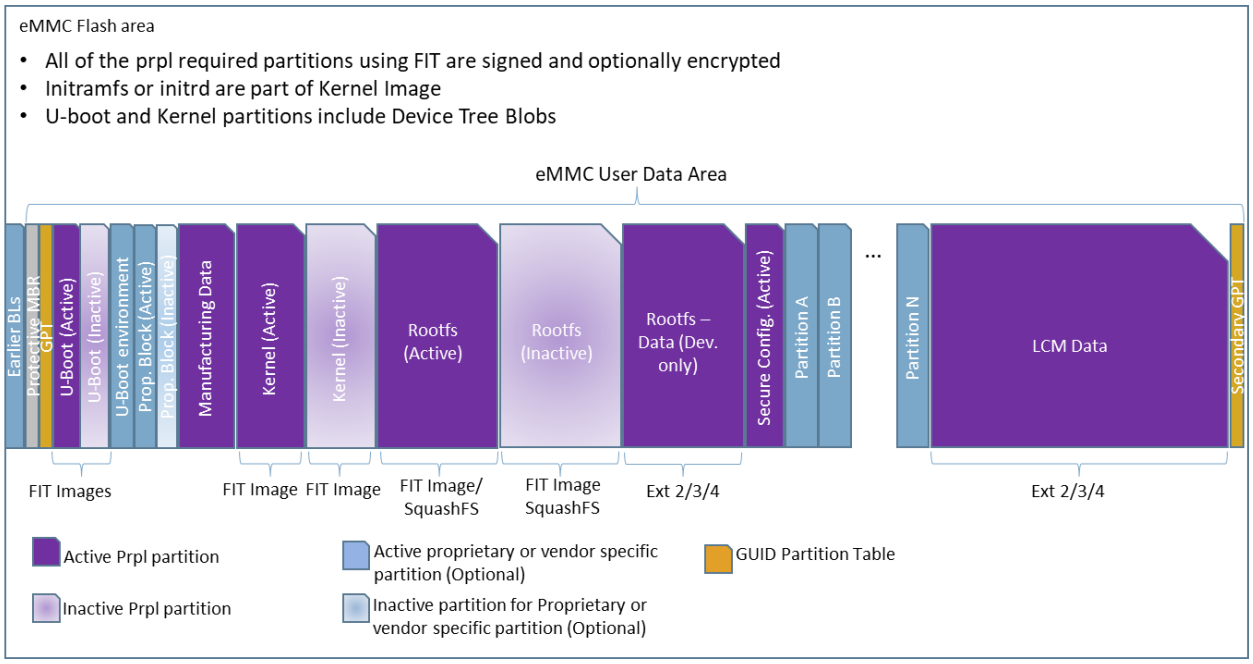
5.5 EMMC MANAGED NAND FLASH

In the following diagram we provide an example flash layout for eMMC flash. Issues such as bad blocks and flash wear-out are managed by the controller of the eMMC. For this reason, this layout does not require a BBT partition. UBI requirements are reduced in this flavor. We utilize GPT to have a flexible partition design. We also employ an “extended (ext)” file system for the advantages that the file system provides.

GPT partitions are resizable, but resizing partitions may cause complications, so it is advisable to overprovision the sizes for upgradable items.

A backup GPT header is usually present at the final LBA even if it is not represented in the diagram below.

While U-Boot, Kernel and Rootfs are in FIT format, the rest of the prpl images can be in a format that the OEM may choose.



6 Compliance Statement

See prpl Secure Boot Requirements (PRPL-SB001) Chapter 7.

Req.	Description	Compliancy
PR-01	On un-managed NAND, the Flash MUST be partitioned in a way that allows a given firmware to perform wear leveling on each filesystem using as many available flash blocks as possible.	Compliant
PR-02	Using two logical firmware banks.	Compliant
PR-03	A given logical firmware bank MUST contain at least a kernel Image, and a root file system image. The initramfs image SHALL also be part of the logical firmware bank requirement if use of initramfs is selected for the build. The root file system image and kernel images SHALL be located in their own separated logical flash partitions. Use of initramfs is optional	Compliant
PR-04	The bootloader MUST be available in at least two separated logical flash partitions.	Compliant
PR-05	Every flash image containing code that may potentially be executed on the gateway MUST be signed.	Compliant

PR-06	Every flash image containing code that may potentially be executed on the gateway SHOULD be encrypted.	Compliant
PR-07	The flash SHOULD contain a dedicated separate partition containing all configuration parameters, including the ones selected by the end user. There MAY be a backup configuration partition for data redundancy. Upon a Firmware upgrade the end user configuration SHOULD NOT be changed.	Compliant
PR-08	If the dedicated configuration partition exists, the bootloader SHOULD have the ability to erase / reset the configuration database	Compliant
PR-09	Flash partitions on raw NAND flash MUST start and stop on sector boundaries	Compliant
PR-10	The flash partitioning MUST allow later addition of new partitions	Compliant
PR-11	In the case of raw flash device usage, firmware size flexibility MUST be supported in the logical firmware banks. For this reason, on raw flash devices UBI is recommended.	Compliant
PR-12	A portion of the flash MUST be used to store the manufacturing data.	Compliant
PR-13	The header or footer content SHALL be documented by the hardware manufacturer if not strictly compliant with open-source implementation and in particular any deviation from original open-source code.	Compliant
PR-14	Each root file system image SHOULD be a squashfs file system image	Compliant

7 References

[PRPL-SB001] Secure Boot Requirements (PRPL-SB001) https://prplfoundation.org/wp-content/uploads/2023/05/prpl-Secure-Boot-Requirements-v1.0.pdf

[PRPL-DSR001] prpl Device Security Requirements https://prplfoundation.org/wp-content/uploads/2018/04/prpl-Device-Security-Requirements-v1.0.pdf

[PRPL-SBW001] An Introduction to Secure Boot (PRPL-SBW001) https://prplfoundation.org/wp-content/uploads/2023/08/prpl-Secure-Boot-Whitepaper-v1.0.pdf

[FIT] Flat Image Tree (FIT) https://U-Boot.readthedocs.io/en/latest/usage/fit/index.html

8 Appendices

8.1 PRPLOS REFERENCE PARTITIONS

Partition/volume names are used by prplOS system to properly identify various element on emmc/flash:

- Various secure boot stages will identify next stage using that name
- Secure upgrade will also rely on those names to determine where to write images
- Identify secure manufacturing data
- Various mount points for data

For GPT, typically used on eMMC, it is desirable to overprovision partitions compared to typical size to account for future additional features as resizing is more difficult compared to UBI volumes. On UBI, volume size can be made to fit exactly the content.

Only elements directly referenced by prplOS are listed below: any number of additional partitions/volumes can be added as long as they don't reuse the same names.

Additional partitions having an upgrade logic can follow the same active / inactive scheme as kernel or rootfs to allow for switching.

Partition name	**Purpose**	**Expected path in filesystem**	**Format**	**Typical size**
u-boot-1	Main U-Boot copy	N/A	raw partition binary-encapsulated in FIT	1 MB
u-boot-2	Second U-Boot copy	N/A	raw partition binary-encapsulated in FIT	1 MB
mfgdata	Manufacturing data	N/A	raw partition Device tree	1 MB
kernel-active	Initially active copy of kernel (being executed)	N/A	raw partition binary-encapsulated in FIT	
kernel-inactive	Initially inactive copy of kernel: backup if kernel is corrupt or target for writing upgrades	N/A	raw partition encapsulated in FIT	

Partition name	Purpose	Expected path in filesystem	Format	Typical size
rootfs-active	Initially active copy of rootfs (being executed)	/	raw partition squashfs encapsulated in FIT	100 MB
rootfs-inactive	Inactive copy of rootfs: backup if rootfs is corrupt or target for writing upgrades	N/A	raw partition squashfs encapsulated in FIT	100 MB
securestore	Store non-volatile configuration	/cfg	ext4 / ubifs	16 MB
lcm_data	Storage for LCM application (OCI images + data)	/lcm	ext4 / ubifs	> 1GB
rootfs_data	default openWRT overlayfs partition. **SHOULD NOT BE PRESENT ON PRODUCTION**	/overlay	ext4 / ubifs	20 to 32 MB

8.2 Implementation Guidelines

8.2.1 U-boot Encapsulated In Fit

Prpl reference partitions recommendation advocate for a U-Boot encapsulated in FIT for following reasons:

- It allows U-Boot to be handled through prplOS upgrade system. Indeed, the upgrading system needs a generic way to understand the versioning of an item and a generic way to authenticate it
- It allows a non-proprietary generic way of authenticating and encrypting the U-Boot.

That requires the previous stage of the boot chain to be able to either directly handle FIT format for authentication or authenticate it through a custom way that would leave the FIT information intact.

As it may not always be possible for the previous boot stage to handle FIT format, depending on the chipset available boot chain or capability, U-Boot may be left authenticated in the trust chain previously available.

In that case standard prplOS upgrade won't be able to handle the U-Boot stage and this would require a customized implementation.

As U-Boot has built in FIT support, another possibility would be to use an already existing proprietary signed U-Boot stage to validate an additional "pivot" U-Boot stage encapsulated.

8.2.2 MFGDATA UPGRADE RECOMMENDATION

It might be desirable to have a mechanism to upgrade mfgdata on the field depending on manufacturer or operator needs.

As mfgdata is critical for the CPE operation, great care must be taken to upgrade it in a secure fashion. For this, it is advisable to provision a mfgdata-inactive partition to allow upgrading an unused version, check its integrity and then switch with the active version.

As mfgdata format is flexible, it can contain per CPE unique data or data authenticated by various parties that may need to be signed. The mfgdata authenticity must also be kept. It is not the purpose of this documentation to describe exactly how the upgrade should be performed as the process will greatly vary depending on operational needs.

8.2.3 OVERLAYFS PARTITION DISABLING FOR FIELD DEPLOYMENT

OverlayFs is really handy for development purposes.

But for real field deployment, security requirements would mandate that no executable could be permanently modified unless it can be authenticated. This implies that OpenWRT overlayFS on filesystem root cannot be allowed to use rootfs_data partition as a backend. Then any non-LCM data would need to be written in the securestore /cfg path.

8.2.4 U-BOOT ENVIRONMENT MANAGEMENT

The U-Boot environment holds various variables that are used to configure the system. It is usually kept as a persistent storage partition / volume that is copied to RAM when U-Boot starts. As this is fine for development purposes, it can pose a security risk if an attacker were to modify specific variables in the stored persistent environment.

To prevent this, it is advised to adopt one of the two following strategies:

- If there is no need to permanently modify the U-Boot environment for operation, it is advisable to disable persistent storage for the U-boot environment and only rely on the default loaded from signed image of U-Boot itself.
  The U-Boot environment can optionally be kept in RAM across non-electrical reboot to allow certain variables to be updated. Some chipsets use dedicated registers that can be used as an alternative.

- When using U-Boot persistent storage, the whitelisting / validation of environment variables (CONFIG_ENV_FLAGS_LIST_DEFAULT) from U-Boot should be used. That way only variables meant to be modified can be loaded from persistent storage. That will prevent security-sensitive variables from being over-written by something loaded from persistent storage.

