prpl Secure Boot Requirements

PRPL-SB001

REVISION HISTORY

Version	Date	Authors	Description
0.1	August 2022	Eric Valette (Orange), Brendan Black (Vodafone)	Import to prpl Template, reorganise, minor edits
0.2	September 2022	Brendan Black (Vodafone), Matthias Hofmann (MaxLinear)	Comments / revisions from prpl Security WG with contributions from the following member companies (in alphabetical order): AT&T, Deutsche Telekom, DISH, MaxLinear, Orange, SoftAtHome, WNC, Verizon, Vodafone.
0.3	March 2023	Brendan Black (Vodafone), Matthias Hofmann (MaxLinear)	Shorten introduction chapters and move generic information into a separate document. Update requirements numbering. Editorial clean-up & Clarifications. Discussed in prpl secure boot Tiger Team
0.9	March 2023		Review version for Security WG
1.0	May 2023		Public release of v1.0

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 organisation 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 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 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 PRPL member companies, is not binding in any way upon 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 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) PRPL Requirements numbers, where applicable, must not be changed from those within the PRPL Documents;
(b) A prominent acknowledgement of PRPL must be provided within the Commercial document identifying any and all PRPL Documents referenced and giving the web address of 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 PRPL.

3 Acronyms

3.1 ACRONYMS

FW	Firmware
WAN / LAN	Wide / Local Area Network
RO	Read-Only
RW	Read-Write
LZMA	Lempel-Ziv-Markov chain algorithm
CPE	Consumer Premise Equipment
CoT	Chain of Trust
RoT	Root of Trust

UBI	Unsorted Block Images. A Linux kernel flash handling layer that manages wear levelling and bad blocks
OTP / SOTP	One Time Programmable memory. A memory that could be written only once during the manufacturing process. It can contain code, data, keys. SOTP adds additional security features so that it can only be accessed by trusted applications.
HSM	Hardware Security Module is a physical computing device that safeguards and manages digital keys for strong authentication and provides crypto processing.
TEE	Trusted Execution Environment. A standard which creates an isolated environment that runs in parallel with the regular operating system, providing security for the rich environment. It is intended to be more secure than the User-facing OS. ARM TrustZone is an implementation of the TEE standard.
Uboot / U-Boot	Das U-Boot (subtitled "the Universal Boot Loader" and often shortened to U-Boot) is an open-source, primary boot loader used in embedded devices
IEK	Image Encryption Key
TRNG	True Random Number Generator
PRPL	(The) prpl Foundation

3.2 TERMS

Firmware	The firmware is the software embedded in the gateways and stored on the flash. It is the sum of all the partitions on the flash.
Partition	Sub-set of the flash that holds a part of the firmware. A partition can hold a filesystem, or any other raw data, such as the bootloader, the kernel...

Image	A flat file or binary entity that can contain the raw or descriptive data defining & containing the total or partial layout of the flash or disk, including kernel, initramfs & filesystems.
Flash memory	Flash memory is non-volatile, solid-state computer memory that can be electrically erased and reprogrammed. Sometimes referred to as simply flash.
Operating System	The software that manages the sharing of the resources of a computer.
File System	A method for storing and organising files and the data they contain.
Wear Levelling	A method for prolonging the service life of some kinds of erasable computer storage media, such as flash. Flash memory media have individually erasable segments, each of which can be put through a limited number of erase cycles before becoming unreliable. Wear levelling attempts to work around these limitations by arranging data so that erasures and re-writes are distributed evenly across the medium.
Bootrom	Bootrom (or Boot ROM) is a small immutable mask ROM or write-protected flash embedded inside the processor chip. It contains the very first code which is executed by the processor on power-on or reset.
Degraded Mode	Operator Specific mode, wherein the device starts up with limited functionality (e.g., no LAN access, to just allow an upgrade / recovery).

3.3 DEFINITIONS OF REQUIREMENTS TERMS

The definitions of MUST, SHOULD and MAY in this document are as follows:

MUST / SHALL: A functional requirement which is based on a clear consensus among PRPL Service Provider members and is the base level of required functionality.

MUST NOT / SHALL NOT: This function is prohibited by the specification.

SHOULD: Functionality which goes beyond the base requirements and can be used to provide vendor product differentiation.

MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item.

Note: these definitions are specific to PRPL and should not be confused with the same or similar terms used by other bodies.

Please note: in case a requirement will be deleted in future versions of this document, its Requirement Number will still stay in the updated version of the document, the text of the requirement will be removed and replaced by “N/A”. This is done so that each requirement stays unique across upgrades of the document.

4 Introduction

This document presents high level specifications and requirements for a gateway bootloader implementation that are derived from an operator's internal bootloader specifications. It will explain how to use classical open-source bootloader and chipset hardware to correctly establish a Root of Trust that will expand security after the boot phase.

This Specification is contributed to the prpl Foundation under the same RAND-based terms as defined by the IPR Policy of the Broadband Forum.

4.1 Objectives of this document

Due to hardware constraints given by the various chip suppliers' boot and secure boot implementations, the early steps in booting a gateway which are closely hardware related are out of scope for this document.

This document does not intend to limit innovation but provides requirements for best industry practice to provide a secure boot process that allows building on a Chain of Trust that is based on the boot ROM code of the silicon.

Another purpose of the document is to define requirements for the PRPL specific part of the boot process that needs on one side be secure, and on the other side needs to allow it to adapt to the various needs of the different operators for example, a secure refurbishment process.
The exact hardware instantiation may change the final technical solutions chosen:

- Having a secure OTP location with hardware encryption / signature validation capability for direct and secure access, would highly simplify the design regarding key management.
- NOR, NAND flash, with or without controller will drastically impact the file system type selection and the need for implementing wear levelling in software.

To enlighten these difficulties, this document will contain a dedicated chapter for constraints and another dedicated chapter for requirements that may slightly change depending on hardware capabilities.

For the impatient, this document explains how to use:

- A Secure One Time Programmable (SOTP) storage that is now available on most gateway chipsets.
- A bootrom sequence able to use keys from SOTP to decrypt and authenticate the intermediate bootloader stages that in turn will use other SOTP elements to decrypt and authenticate next stage bootloaders.
- Trusted Execution Environment (TEE) that could be based on Trusted Firmware or on a dedicated hardware security processor when available.

Furthermore, the document includes requirements on key handling.

The document has a companion whitepaper on how bootloaders work, what comprises a secure bootloader and why it's needed at all, and which other functionality of a gateway is dependent on bootloader functionality that has to be considered for a secure boot process.

Please note:
Some software licences (Like GPLV2) may require publishing some or all parts of the encrypted code. The Operator who provides the gateway with the software to the end user will control and be responsible for this aspect.

4.2 MAIN BOOT STAGES

In this chapter we will give a brief overview and define names for the main stages of the boot process of a gateway.

Boot stages	Description
ROM bootloader	Immutable boot code that is run at cold or hot boot of the silicon
Intermediate bootloader stages	Preparing the configuration, bringing up the essential blocks of the silicon e.g., DDR, TEE, etc…
Last stage “PRPL compliant” bootloader	Bootloader (u-boot) that includes the PRPL specific bootloader requirements and allows customization by / for the Operator.

5 Functional Description and Context

5.1.1 Summary of collected operational requirements

This chapter summarises high level requirements. They were derived from practical experience in the field. More detailed requirements will be given in chapters 7, 8, and 9.

The Operational Requirements are numbered “OR-xx”

- **OR-01** The bootloader code and the complete firmware code running on the gateways MUST be protected against unauthorised modification and forgery.
- **OR-02** End user MUST never be granted access to any bootloader command on production boxes.
- **OR-03** The bootloader execution MUST NOT reveal information about the hardware or software that compromises security, or that can be collected by the end user.
- **OR-04** Hardware-invasive attacks MUST NOT provide an easy way to identify code running on the box, and SHALL NOT allow access to confidential cryptographic assets / secret keys. Public keys used for authentication SHOULD be protected from modification by such attacks.
- **OR-05** The gateway’s firmware upgrade process MUST be resistant to power outage throughout the firmware upgrade process.
- **OR-06** The gateway’s code MUST provide a means to reset a broken configuration database.
- **OR-07** The gateway’s code MUST provide means to replace a broken operational firmware which prevents a 'regular' boot process. A recovery MUST be possible if the gateway can no longer connect to an ACS.
- **OR-08** For an Operator the 'last stage' PRPL compliant bootloader SHOULD behave the same way on a given gateway generation, even if the gateways are supplied via different sources.
- **OR-09** The last stage PRPL compliant bootloader that loads and starts the main OS's (usually Linux) kernel SHOULD be u-boot.
- **OR-10** In order to allow modification and additional features, the last stage PRPL compliant bootloader MUST be upgradable in the field.
- **OR-11** Intermediate bootloaders SHOULD be upgradable in the field to patch (or apply bug fixes) or add new features to the gateway.
- **OR-12** The secure bootloader MUST support a secure refurbishment process.
- **OR-13** A mechanism enabling the prototyping of new features directly on target hardware MUST be available via specific development (DEV) gateways.
- **OR-14** A DEV device MUST NOT use the same keys as a Production PROD device.

5.2 Collected functional requirements

In this paragraph we will translate the operational requirements from the last chapter into functional requirements.

The Functional Requirements are numbered “FR-xx”:

- **FR-01** The gateway’s boot process MUST use a Chain of Trust (CoT) that protects the gateway’s boot process from reset till the main OS kernel will be booted and provide the required security. The Root of Trust (RoT) MUST be protected by a hardware secured mechanism.
    - Note: Hardware secured mechanisms could be as follows: Read-only bootrom that verifies next boot stage, key(s) stored in SOTP, secured boot engine, dedicated security processor, ...
- **FR-02** Each flash partition containing a firmware image or executable code image located in a rewritable part of the flash MUST also contain a signature for the image enabling to validate the authenticity of the code.
- **FR-03** Each flash partition containing a firmware image (kernel, initramfs or rootfs) or executable code image located in a rewritable part of the flash SHOULD be encrypted to prevent analysis of the content.
- **FR-04** The portion of flash signed MUST include not only the firmware image itself but also any mandatory information in the signature header that may be used by the bootloader itself like firmware type, firmware version, etc...
- **FR-05** On production firmware, the bootloader MUST NOT output anything and MUST NOT listen on a serial console even if the serial port pins are not soldered.
- **FR-06** The gateway’s bootloader code MUST prevent firmware regression to a less secure firmware compared to the one that was previously run successfully on the gateway; this MAY be overridden by an authorised source i.e. Operator ACS if physically possible.
- **FR-07** The last stage “PRPL compliant” bootloader MUST not boot a firmware that has a lower version compared to the last firmware's security version (SVN). An SVN is incremented whenever a new firmware fixes significant security vulnerabilities / bugs or enhances security functionality.
- **FR-08** The gateway’s last stage “PRPL compliant” bootloader MUST verify firmware signature at boot time and refuse to load the firmware in case of a failed signature verification.
- **FR-09** During normal boot all bootloader stages MUST NOT emit ethernet frames unless solicited where required by the Operator.
- **FR-10** The last stage “PRPL compliant” bootloader MUST be able to handle multiple boot images (e.g., rescue, nominal, backup, etc.) and MUST NOT impose restriction on their location in the flash except those imposed by the flash itself (e.g alignment on flash block boundaries).
- **FR-11** In case the last stage PRPL compliant bootloader does not find any valid firmware image, it MUST still be able to load a rescue firmware image via a device refurbishment process like 'reflash from LAN', USB DFU, or other supported process. This process MUST NOT compromise security.
- **FR-12** If the Hardware supports the feature to recover damaged bootloaders, it MUST be possible to recover damaged bootloaders via a device refurbishment process like 'reflash from LAN', USB DFU, or other supported process. Security rules MUST apply.
- **FR-13** The boot loader code MUST be able to detect the push on any combination of buttons.
    - Note: The PRPL LLAPI defines buttons and a common API to detect buttons. The last stage “PRPL compliant” bootloader MUST be able to detect buttons pressed and based on operator defined requirements trigger an action (e.g., selection of image to be booted).

    - Note: The same device tree parameters SHOULD be used between last stage “PRPL compliant” bootloader and Linux.
- **FR-14** It MUST be possible to use a button combination to start a regular firmware in a mode where it simply resets the configuration database and reboots.
- **FR-15** The last stage “PRPL compliant” bootloader MUST be able to indicate when it is forced, without prior end user action, to boot in recovery or fallback mode (see figure in PR-02), for example:
    - No valid N-1 firmware image found (e.g., this can happen when flashing of a new firmware gets interrupted).
    - Only N-1 Firmware found valid (bad block on flash preventing firmware signature verification) However, as a new firmware is already deployed, it means that previous firmware SHALL only run in degraded mode.
        - Where security version numbers are in use, the decision to run in degraded mode can be based on whether the security version number is acceptable to run. This status is indicated to the user.
    - The last stage “PRPL compliant” bootloader cannot verify the only available bootable firmware.
    - The last stage “PRPL compliant” bootloader MUST have the ability to communicate to the firmware that it has to run in this degraded mode.
- **FR-16** The last stage “PRPL compliant” bootloader MUST be able to indicate when flash content layout seems to be incoherent for instance:
    - No valid image found. Meaning firmware cannot be upgraded until a new image is flashed.
    - The preferred image was not able to be booted, the bootloader MUST indicate to the OS what the cause was.
- **FR-17** Each stage of the bootloader and kernel / firmware MUST have the ability to indicate the stage that has been reached in the boot process via LED / LCD or other indicators defined via device-tree or plugin at that stage. The indicators MUST NOT be hardcoded in the boot loader**.**
- **FR-18** The bootloader execution time MUST be both secure and fast because it negatively impacts the overall boot time (even if bootloader execution time is usually negligible).
- **FR-19** A device Refurbishment Process SHOULD be initiated from a last stage “PRPL compliant” bootloader.
    - Please Note: Today Operators are having different refurbishment processes. Sometimes they are bound to the capabilities of the Service Provider handling the refurbishment process.
    - A Refurbishment Process MAY be initiated automatically - e.g., by an Ethernet Frame sent by the Refurbishment network, or by specific button presses at certain stages of the boot process, or a combination of both.

6 Key Bootloader Software Architecture Requirements

This paragraph summarises key software architecture solutions deemed necessary to achieve all the previously exposed requirements.

6.1 SOFTWARE DESIGN REQUIREMENTS

The Software Design Requirements are numbered “SD-xx”

- **SD-01** All mechanisms which upgrade the firmware MUST share the same algorithms for flash management as they will operate on the same flash location. The firmware upgrade mechanisms could be provided by the last stage “PRPL compliant” bootloader “Reflash on LAN”, or other local upgrade mechanisms like USB DFU boot features, or a firmware upgrade via ACS.
- **SD-02** In case bad block handling of the flash is not managed by the flash itself or a specific controller, bootloaders MUST use bad block handling when writing or reading from that flash.
- **SD-03** In case wear levelling of the flash is not managed by the flash itself or a specific controller, all bootloaders MUST handle wear levelling when writing to that flash.
- **SD-04** The intermediate bootloader MAY consist of multiple stages. They are executed before the last stage PRPL compliant bootloader. The intermediate bootloader MUST initialise the hardware for the last stage PRPL compliant bootloader. The PRPL compliant last stage bootloader is typically the code block where operator specific functionality gets implemented - the intermediate bootloader stages are specific to the silicon and are typically provided by the silicon supplier.
- **SD-05** The intermediate bootloader stages MUST focus on 'early boot' requirements and security aspects, and be able to load the PRPL compliant bootloader code (authenticate and decrypt).
- **SD-06** For designs using controllerless NAND flash, the flash management algorithm supported by both the bootloader and the Linux kernel MUST be the same. UBI is the preferred implementation option.
- **SD-07** In the case where a TEE is in use, the TEE MUST be booted and operational before the last stage PPRL compliant bootloader loads the Linux kernel and/or does any external interface usage like LAN. TEE can be booted at power on in parallel with the intermediate bootloader stages.

7 Partitioning Requirements

The high-level requirements collected in the previous chapters, will be translated into low level requirements that should be easily verifiable via basic tests. This chapter focuses on Partitioning Requirements

The Partitioning Requirements are numbered “PR-xx”

7.1 FLASH PARTITIONING

7.1.1 GLOBAL PARTITIONING STRATEGY

Given the actual cost of NAND flash per MB, the flash storage SHOULD be large enough to store two complete firmware images. This simplifies the firmware upgrade mechanism and minimises impact to the end user during a firmware upgrade, as the box will not need to reboot twice.

The following set of requirements, aims at describing the high level view of the global flash layout:

REQ ID _ TITLE	PR-01 – On controllerless NAND, the Flash MUST be partitioned in a way that allows a given firmware to perform wear levelling on each filesystem using as many available flash blocks as possible.
DESCRIPTION	
If the number of blocks used to perform wear levelling for a given file system is limited to the physical blocks that effectively belong to this file system, this may result in very rapid flash wear out. Spreading the writes for a given filesystem on the entire flash is much more efficient when possible but this requires a logical block layer abstraction. For controllerless NAND flash, the UBI layer allows this for any file system located in UBI volumes on a given UBI device.	

REQ ID _ TITLE	PR-02 – Using two logical firmware banks.
DESCRIPTION	
In this model, when upgrading the gateway’s running firmware version N into firmware version N+1, the currently running firmware writes the new firmware image in a non-active bank and once the operation completes it makes the bootloader aware of the change so that it will use the bank containing the firmware N+1 instead of firmware N at next reboot. The firmware SHALL be stored in two or more logical firmware banks. Two main approaches can be used to store the images in different partitions. Some of the known industry best practices are listed below.	

1. Identical Backup - Both the partitions store the same latest upgraded firmware on commit (N), OR

2. Last Good Image Backup - Primary bank stores the upgraded firmware (N) and the other partition stores the previous working firmware (N-1,...) even after commit.

In either case, there is a separate 'commit' step following the upgrade request, which SHALL update Anti-Rollback Id if supported, and / or the firmware in the backup bank based on mode selected. This commit can be carried out remotely after the operator determines that the upgraded firmware is working fine for a desired "observation period."

Committing an update indicates that an upgrade image is feasible and accepted by the platform owner. Organisations may have different metrics to accept this success according to their internal policies. Hence, the commit operation MUST be decoupled from the bank upgrade operation.

please see drawing below:

Action	Identical Backup		Previous Backup		Comments
	Primary	Backup	Primary	Backup	
					System is booting from 'Primary' always until it fails, then from Backup
Factory	V	V	V	V	Initial version from factory
Upgrade	V+1	V	V+1	V	Upgrade request of new firmware
Commit	V+1	V+1	V+1	V	Commit request can come whenever Operator decides new image is sound, could happen at any time sooner or later. Anti-Rollback ID is written to OTP etc at Commit apart from any image backup – we boot with 'Primary'
Upgrade2	V+2	V+1	V+2	V+1	A 2nd upgrade request for firmware
Commit2	V+2	V+2	V+2	V+1	Commit request can come whenever Operator decides new image is sound, could happen at any time sooner or later. Anti-Rollback ID is written to OTP etc at Commit apart from any image backup

REQ ID _ TITLE	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.
DESCRIPTION	

As during a firmware upgrade it may be required to change the Linux kernel image (new Linux kernel version, security or bug fixes in the same Linux kernel), having a single kernel image used by two firmware banks would prevent changing the kernel in the field or at the risk of bricking the box when we rewrite the kernel to replace it by the new version if a power cut occurs.

Because the Linux kernel image will be loaded by the last stage "PRPL compliant" bootloader, and the root file system image will be mounted by the Linux kernel only later on, the requirements regarding their precise flash image properties may be substantially different as the driver code for flash handling will be quite different.

Placing the Linux kernel in the root file system creates a dependency on the last stage "PRPL compliant" bootloader: it MUST be able to read & mount the root filesystem, so putting the kernel image in a separate partition removes this dependency.

When loading program pages on demand, and the root file system code is encrypted, the first user space code executed shall be from a unencrypted location (usually an initramfs loaded in RAM) that will provide setup for the key and mount command to correctly decrypt the root file system partition content.

REQ ID _ TITLE	PR-04 – The last stage “PRPL compliant” bootloader MUST be available in at least two separated logical flash partitions.
DESCRIPTION	
Operators may want to be able to enhance the last stage “PRPL compliant” bootloader functionality in the field. If the bootloader code is split into several stages, only the last stage that contains most of the functional requirements MUST be duplicated.	

REQ ID _ TITLE	PR-05 – Every flash image containing code that may potentially be executed on the gateway MUST be signed.
DESCRIPTION	
The use of a read only filesystem like squashfs does not prevent an attacker modifying filesystem image in the flash. Using a read only filesystem allows using a global file system signature that could be checked at boot time instead of verifying the signature of every executable file before executing it. In particular, the last stage “PRPL compliant” bootloader image, the kernel image and the root file system image MUST be *independently* signed and have each a *dedicated* partition containing both the image itself and its signature block. The exact signature mechanism will be described in chapter 9.1.1. NB: if a read-write filesystem is used to store deployment units for the LCM manager, the read-write filesystem itself cannot be signed but the deployment units themselves (OCI images typically) MAY be encrypted and MUST be signed.	

REQ ID _ TITLE	PR-06 – Every flash image containing code that may potentially be executed on the gateway SHOULD be encrypted.
DESCRIPTION	

REQ ID _ TITLE	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.
DESCRIPTION	
When doing a firmware upgrade, the end user wishes that his personalised gateway setup (e.g WiFi SSID, WiFi scheduling, device naming, …) is preserved.	

REQ ID _ TITLE	PR-08 - If the dedicated configuration partition exists the last stage “PRPL compliant” bootloader SHOULD have the ability to erase / reset the configuration database
DESCRIPTION	
The exact method of signalling to the last stage “PRPL compliant” bootloader is left up to the operator.	

REQ ID _ TITLE	PR-09 – Flash partitions on raw NAND flash MUST start and stop on sector boundaries
DESCRIPTION	
Because the flash is organised as sectors, and because a sector is the erase quantum, a sector cannot be shared between two partitions. In addition, not starting on sector boundaries may have a huge performance impact. Note that for the eMMC flash, this is handled by the flash microcontroller code itself.	

REQ ID _ TITLE	PR-10 – The flash partitioning MUST allow later addition of new partitions.
DESCRIPTION	

REQ ID _ TITLE	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.
DESCRIPTION	
Static partitioning SHOULD be avoided as much as possible as it both potentially wastes space (e.g size of the firmware is smaller than the partition size) and imposes arbitrary constraints on the firmware size (that cannot exceed the pre-allocated size). NB: Using UBI volumes enables the fulfilment of these requirements by destroying and recreating volume at the correct size when upgrading a firmware image or last stage “PRPL compliant” bootloader image.	

7.1.2 Additional flash partitions

While the previous chapter describes the flash partitioning strategy we want to enforce, in this chapter we will describe additional content that may be stored on the flash.

REQ ID _ TITLE	PR-12 – A portion of the flash MUST be used to store the manufacturing data.
DESCRIPTION	

Please see [1] for details regarding Manufacturing Data.

REQ ID _ TITLE	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.
DESCRIPTION	
As it will mainly be used by bootloader and firmware upgrade code that may be hardware specific, we may leave some freedom to hardware manufacturers. Only the signature size and the algorithm used to check it will be imposed for security reasons. It will be described in the security chapter. In case the FIT image format derived from UBOOT is used, any deviation to original code SHOULD be described.	

REQ ID _ TITLE	PR-14 – Each root file system image SHOULD be a squashfs file system image
DESCRIPTION	
Because we want to forbid root file system alteration at run time, we select a high performance read only filesystem type. The option of read-only mounting a normally read-write filesystem type has issues whereby a root user can remount read-write and modify the contents which may lead to a failure during verification on the next system boot.	

8 Firmware upgrade mechanisms

In this chapter requirements are given to ensure that bootloader and firmware upgrade mechanism work hand in hand.

8.1 UNIVERSAL FIRMWARE UPGRADE FORMAT (FUF)

REQ ID _ TITLE	UR-01 – The SWUpdate framework with its single image delivery capability SHOULD be used.
DESCRIPTION	
Even if several methods exist for performing a firmware upgrade, the new firmware image format used SHALL remain the same.	

REQ ID _ TITLE	UR-02 – The Universal Firmware Upgrade Format MUST be designed in a way that ensures that streaming upgrade is possible while still making sure what has been flashed is correct before activating it.
DESCRIPTION	

Rationale: In case of remote update, if we first need to wait for complete image download before starting to flash it, it means all content needs to be stored in memory. This guarantees that an update is initiated only if all parts are present and correct via a single signature verification (that may be put in addition to all individual firmware component signatures). However, on some systems with less resources, the amount of RAM / Flash to copy the images could be not enough to store the image and the normal firmware.

This means that each upgradable item SHOULD be verifiable either before starting to flash it (using signature that will be used also by bootloader while booting) or that the signature of the header or footer can be verified after flashing like the bootloader will do once flashed before trying to activate it.

Both non-streaming and streaming updates are supported.

REQ ID _ TITLE	UR-03 – The image MUST contain a signature for verification. The image MAY be encrypted using symmetric encryption.
DESCRIPTION	

REQ ID _ TITLE	UR-04 – The FUF SHOULD allow provision for a SoC specific secure boot firmware upgrade option in addition to the universal FUF defined here.
DESCRIPTION	

9 Security Features

In this chapter, we will give a more detailed description of the security means used to achieve our security target for gateways. The bootloader is the root of the security chain. If the security cannot be enforced in the bootloader, the rest of the security chain may entirely collapse.

9.1 KEY STORAGE RULES

REQ ID _ TITLE	SR-01 – Symmetric and private keys MUST NOT be stored in clear in flash

DESCRIPTION

Rationale: If an attacker gets root access to the box exploiting an unpatched remotely triggerable security issue, they will try to make their exploit permanent and try to extract valuable data by dumping flash. They may find it easier to unsolder the gateway's flash and remount it elsewhere if they do not manage to find an exploit.

REQ ID _ TITLE	SR-02 – Symmetric or private keys MUST NOT be stored in clear in user space RAM

DESCRIPTION

Rationale: If attackers get access to the box exploiting an unpatched vulnerability remotely, they will try to gain a foothold and exfiltrate valuable assets by dumping compromised process RAM content. Via privilege escalation, they may also try to dump other process memory areas or even kernel memory if the kernel is not correctly configured.

The keys SHALL be stored once the system is fully operational in TEE memory if a TEE is implemented or in Linux Kernel Keyring if no TEE is available.

Note that this has an important coding impact on all user space software using cryptographic keys, and a cryptographic library like *openssl* has a specific SSL engine that MUST then be used (like cryptodev or pkcs11 engine).

Please also see comment in PR-03

REQ ID _ TITLE	SR-03 – If a TEE is available, private keys MUST NOT be stored in the clear in the Linux kernel accessible space RAM, but in the TEE address space. Symmetric keys may also be stored in TEE space where this enhances security.

DESCRIPTION

Rationale: This depends on the value of the symmetric key and the primary asset it protects. For example, an application may use a symmetric key in userspace for session keys, then regenerate the key periodically as per established security guidelines. On the other hand, the key used to decrypt secured / protected objects stored in FLASH MUST reside in TEE space.

REQ ID _ TITLE	SR-04 – Each boot stage MUST have means to both verify signature and decrypt the next bootloader stage.

DESCRIPTION

Rationale: At a minimum the following 2 approaches MUST be taken considered (Operator to choose which to use):

1. Unified chain of trust, not necessarily relying on SoC secure boot and SOTP / TEE solution provided by Chipset vendors; and,

2. SoC Vendor supplied HW Chain of Trust, relying on SoC security capabilities like SOTP, TEE, Security Engines if provided by SoC vendor.

Operator or PRPL customer MUST be able to choose either method if supported on a SoC.

To guarantee the chain of trust, all boot stages MUST be signed, and MAY be encrypted. Signatures must be checked, and decrypted, before the bootloader code gets executed.

REQ ID _ TITLE	SR-05 - Keys used by a bootloader stage MUST be secured as follows:

	Symmetric keys (e.g., decryption keys) MUST be stored in either: a) SOTP; or, b) next stage Image Header in secure wrapped form (e.g., NIST SP800-38F); or, c) FLASH in secure wrapped / encrypted form with a decryption key in SOTP; or, d) In the case of a derived key, the key that it is derived from (e.g., root of encryption) MUST be stored in SOTP or similar secure storage mechanism.
DESCRIPTION	

REQ ID _ TITLE	SR-06 – The integrity of Public Keys MUST be validated before being used when stored clear in flash.
DESCRIPTION	

Rationale: These keys SHALL NOT be altered. In practice this means that either:

- A hash of those keys is stored in non-modifiable chipset memory. The hash SHOULD be verified before using the keys.
- These keys are signed because they are embedded in a partition containing code whose signature is verified.

9.1.1 CRYPTOGRAPHIC KEYS AND ALGORITHM REQUIREMENTS FOR THE BOOTLOADER

REQ ID _ TITLE	SR-07 – Asymmetric operations MUST be cryptographically secure and compliant with guidelines from standards bodies like NIST, ANSSI, ENISA, NCSC etc (As of 2021, these could be RSA-2048 or ECDSA-256 and higher) - Stronger algorithms may be used for higher security.
DESCRIPTION	

REQ ID _ TITLE	SR-08 – Symmetric operations MUST be cryptographically secure and compliant with guidelines from standards bodies like NIST, ANSSI, ENISA, NCSC etc (As of 2021, it could be AES-256) - Stronger algorithms may be used for higher security
DESCRIPTION	

REQ ID _ TITLE	SR-09 – Hash operations MUST be cryptographically secure and compliant with guidelines from standards bodies like NIST, ANSSI, ENISA, NCSC etc (As of 2021, it could be SHA-256) - Stronger algorithms may be used for higher security
DESCRIPTION	

REQ ID _ TITLE	SR-10 – Symmetric signature operations MUST be cryptographically secure and compliant with guidelines from standards bodies like NIST, ANSSI, ENISA, NCSC etc (As of 2021, these could be HMAC-SHA256 or KMAC128 and higher) - Stronger algorithms may be used for higher security
DESCRIPTION	

9.1.2 CRYPTOGRAPHIC KEYS RELEVANT FOR THE BOOT PROCESS: GENERATION AND STORAGE

In this chapter, we intend to generically explain the minimal set of keys that SHOULD be provisioned in SOTP hardware. Some keys described here may be mandatory for securing the intermediate bootloader stages.

REQ ID _ TITLE	SR-11 – Storage of Operator keys on production Line. Operator production keys, or their hash SHOULD be generated by a HSM (Hardware security module) or equivalent secure environment.
DESCRIPTION	

REQ ID _ TITLE	SR-12 – Whenever in a chipset maker boot or SOTP documentation, a key is specified either as an Operator key or an OEM key, the operator MUST have the choice to supply the corresponding key, or to delegate this key generation to the OEM, or to use a mixture of both options
DESCRIPTION	

REQ ID _ TITLE	SR-13 – SOTP SHALL contain the K_ROT public key or its HASH. The Cryptographic algorithms used MUST be cryptographically secure (see SR-07)
DESCRIPTION	
K_ROT is the Root of Trust public key. It is used by the bootrom bootloader to authenticate the 1st bootloader stage loaded by the bootrom. It may be used to authenticate later stages of the boot process. Access to this key MUST NOT be granted to the Main OS's kernel, as this key is meant to be used only by the secure boot process, and then only up to the stage that the kernel is executed	

REQ ID _ TITLE	SR-14 – If K_ROT is stored in Flash, K_ROT MUST be verified against its hash (stored in SOTP) at least once before using it.
DESCRIPTION	
K_ROT can be stored in SOTP or in FLASH - in the latter case, hash of K_ROT is stored in SOTP. The location of the K_ROT MUST be documented.	

If the hash of K_ROT is stored in SOTP, the storage location of the public key itself MAY vary, provided all boot stages that use it can access it. The boot stage at which the hash verification occurs SHOULD be documented.

REQ ID _ TITLE	SR-15 – SOTP SHOULD contain a source key used for key derivation: K_RDK (Root Derivation Key). This K_RDK SHOULD be used exclusively to compute affiliate symmetric encryption keys. This key MUST be protected with care as once leaked it cannot be changed.
DESCRIPTION	
It is advised that derivation be done in the early boot stage. Any leakage of the key into the main OS (Linux) MUST be strictly prohibited by appropriate security controls (e.g., usage in secured RAM only, or usage in a dedicated secured HW block only).	

REQ ID _ TITLE	SR-16 - Where the hardware is capable, the early-stage bootloader SHOULD be encrypted using a symmetric key (see SR-08) K_ROE (root of encryption)
DESCRIPTION	
Starting from the last stage “PRPL compliant” bootloader stage, it could be desirable to use other encryption keys derived from K_RDK (see SR-15) to allow more flexibility.	

REQ ID _ TITLE	SR-17 – The code required to store the keys in SOTP during production SHOULD be provided to the OEM and / or Operator and its usage be thoroughly documented
DESCRIPTION	

REQ ID _ TITLE	SR-18 – The SOTP MAY contain additional keys
DESCRIPTION	
Some SoCs MAY allow multiple keys, key derivation, and key wrapping capabilities in a secure environment like a hardware-based Security Engine and / or TEE, and the flow may be adapted while meeting the Security requirements. Decryption of an Image may e.g., happen concurrently with Signature verification, and the source of the symmetric key may be in more than one place. This would require different privileges for different implementation styles.	

REQ ID _ TITLE	SR-19 - Public keys used in the chain of trust to verify a specific image MAY be part of the image header or stored in flash
DESCRIPTION	

While the K_ROT MUST be locally available in the gateway in a secure manner like SR-13, the other keys in the chain of trust to verify a specific image MAY be part of the image header or stored in flash - The Chain of Trust MUST be authenticated through the levels of keys / certificates. The key used to authenticate an image could be a derived key. If public keys are stored in flash, a security control MUST be available and used to ensure the key has not been tampered with before it gets used.

REQ ID _ TITLE	SR-20 - Image Encryption Keys (IEK) used for decryption, MUST be stored, and used securely.
DESCRIPTION	

An IEK is a key used to encrypt / decrypt an image. IEK MAY be stored encrypted or wrapped in FLASH, it may be stored in SOTP, or it may be unwrapped from a wrapped key stored in the images secure boot header using security HW or a TEE (e.g., AES-KeyWrap function can be used).

Symmetric keys that are stored in the header of an image, SHOULD be either wrapped by a key in SOTP or secure storage, or MUST use a secure key derivation function using K-RDK with a high-quality random IV (initialization Vector / Nonce). A high-quality random source such as a TRNG SHOULD be provided.

9.1.3 SIGNATURE OF BOOTLOADER STAGES AND FIRMWARE

REQ ID _ TITLE	SR-21 – Secure Bootstrap: Each bootloader stage MUST be signed using the appropriate strong private key and MAY be encrypted using the appropriate strong symmetric key. Integrity of stage N must be checked within stage N-1, before decrypting it and running it. (See requirement SR-07 and SR-08 for algorithm and key length)
DESCRIPTION	

REQ ID _ TITLE	SR-22 – Last stage “PRPL compliant” bootloader's parameters: Last stage “PRPL compliant” bootloader SHALL be passed with an indication whether the primary boot loader or the backup bootloader image was used.
DESCRIPTION	

This enables the use of the same binary code for both the primary boot loader and the backup bootloader.

All other or larger arguments MUST be ignored.

It is expected that only the last stage “PRPL compliant” bootloader will be upgraded in the field. On NAND flash, this requires having two images for this last stage. They have no reason to be different and, in fact, it could even be safer to upgrade the second bootloader image after verification of successful boot with a new image. Nevertheless, if the active / primary path image becomes corrupted (e.g., because of NAND bad block), it SHOULD be traced so that a reflash of a working image can be attempted.

9.1.4 LAST STAGE "PRPL COMPLIANT" BOOTLOADER CRYPTOGRAPHIC OPERATIONS

REQ ID _ TITLE	SR-23 - It MUST be possible to have Production (PROD) and Development keys (DEV).
DESCRIPTION	
Development keys for development devices SHOULD be based on a cryptographically distinct set of keys compared to the production keys. This includes K_ROT as well as all other keys used to authenticate or encrypt bootloaders or images.	

REQ ID _ TITLE	SR-24 – The last stage "PRPL compliant" bootloader MUST authenticate the kernel using asymmetric keys as referenced in Section 9.1.1. Authentication failures MUST be handled according to SR-30.
DESCRIPTION	

REQ ID _ TITLE	SR-25 – If the image is not decrypted by an earlier bootloader, the last stage "PRPL compliant" bootloader MUST decrypt the kernel using a strong symmetric key accessible by the last stage "PRPL compliant" bootloader itself (non-TEE case) or request decryption using the TEE (TEE case).
DESCRIPTION	

REQ ID _ TITLE	SR-26 – Root File system authentication MUST be supported by one the following methods in PRPL: 1- Using initramfs and authenticate rootfs on the fly (SR-27) 2 -Authenticate rootfs in RAM and mount from RAM Block device (SR-28) 3. Verify the signature of a filesystem to mount directly Systems SHOULD be able to do any of the above based on operator selection of option
DESCRIPTION	

REQ ID _ TITLE	SR-27 – Root File system decryption SHOULD support the use of initramfs and decryption of rootfs on the fly.
DESCRIPTION	

REQ ID _ TITLE	SR-28 - As an alternative to SR-27, the last stage “PRPL compliant” bootloader MUST support decrypting the whole rootfs to RAM, and mounting from RAM block device - This SHOULD result in faster boot at the expense of RAM usage (dependent on code on flash and code executes on the device) - option as specified in SR-26.
DESCRIPTION	

REQ ID _ TITLE	SR-29 – After authentication, the last stage “PRPL compliant” bootloader SHALL load and start the kernel, informing it about the address of various content stored in DDR it needs to access for completing its initialization (DTB, initramfs image, ...).
DESCRIPTION	

REQ ID _ TITLE	SR-30 – Booting backup kernel and backup rootfs if something goes wrong: After authentication, if for some reason any component of the normal (primary bank) firmware is corrupted (kernel, initramfs or Rootfs), it SHALL be recorded in persistent storage. The last stage “PRPL compliant” bootloader MUST then try to load and start the image from the backup bank, and MUST avoid an infinite boot-loop. We do not want the last stage “PRPL compliant” bootloader to authenticate Rootfs, but need a way to notify the last stage “PRPL compliant” bootloader of previous Rootfs authentication failure.
DESCRIPTION	

9.1.5 HARDWARE SECURITY REQUIREMENTS WITH POSSIBLE IMPACT ON BOOTLOADER CODE

REQ ID _ TITLE	HR-01 - The console port could be physically accessible after soldering but it MUST be fully deactivated (debug access, logs, console) on any production software, including firmware and bootloader. .
DESCRIPTION	

For example in Linux-based systems this can be achieved by hardlinking /dev/console to /dev/null on system init.

REQ ID _ TITLE	HR-02 - Any access to the following on-chip debugs on production devices SHALL be made impossible before the device leaves manufacturing facilities: • JTAG • SWD
DESCRIPTION	

This SHALL be ensured either by securely deactivating the debug features at the chipset level (e.g., by burning a fuse in the chipset) or by grounding the debug pins without any exposed trace on the PCB. In the first case reactivation SHALL either be impossible or necessitate a strong authentication (certificate-based, for instance). As some manufacturing sites may use JTAGS at production time to validate the PCB, it means that the manufacturing code that performs the final security enforcement SHALL disable JTAG.

10 REFERENCES

[1] prpl Secure Manufacturing Data Standard, PRPL-SMD001

