An Introduction

to Secure Boot

PRPL-SBW001

REVISION HISTORY

Version	Date	Authors	Description
0.1	Feb 2023	Eric Valette (Orange)	Initial draft for Whitepaper based on Eric's work at Orange
	May 2023		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.
0.7	June 2023	Onur Zengin (MaxLinear), Matthias Hofmann (MaxLinear)	Consolidation for WG review
0.8	July 2023	Security WG	Minor editorial changes
1.0	Aug 2023		Initial public release

1 Contents

1 Contents 3
2 Important Notices, IPR Statement, Disclaimer, And Copyright 5
2.1 ABOUT PRPL 5
2.2 THIS MAY NOT BE THE LATEST VERSION OF THIS PRPL DOCUMENT 5
2.3 THERE IS NO WARRANTY PROVIDED WITH THIS PRPL DOCUMENT 5
2.4 EXCLUSION OF LIABILITY 5
2.5 THIS PRPL DOCUMENT IS NOT BINDING ON PRPL NOR ITS MEMBER COMPANIES 6
2.6 INTELLECTUAL PROPERTY RIGHTS 6
2.7 COPYRIGHT PROVISIONS 6
2.7.1 INCORPORATING PRPL DOCUMENTS IN WHOLE OR PART WITHIN DOCUMENTS RELATED TO COMMERCIAL TENDERS 6
2.7.2 COPYING THIS PRPL DOCUMENT IN ITS ENTIRETY 7
3 Acronyms 7
3.1 ACRONYMS 7
3.2 TERMS 8
4 Introduction 9
4.1 OBJECTIVES OF THIS DOCUMENT 9
5 Bootloader and Secure Bootloader 10
5.1 WHAT IS A BOOTLOADER, HOW DOES IT WORK 10
5.2 SECURITY CONSIDERATIONS FOR THE BOOTLOADER 11
5.3 WHAT IS ROOT OF TRUST (RoT), WHAT CONSTITUTES A CHAIN OF TRUST (CoT)? 12
5.4 SECURITY CONSIDERATIONS PERTINENT TO OPERATORS 12
5.4.1 MAIN SECURITY BUILDING BLOCKS 12
5.5 COLLECTED OPERATIONAL CONSTRAINTS 13
5.5.1 DEVICE REFURBISHMENT PROCESS 13
5.5.2 FIRMWARE UPGRADE AND FIRMWARE ROLLBACK MECHANISMS 13
5.5.3 SPECIFIC BOOTLOADER KEYS FOR DEVELOPMENT GATEWAYS 14
5.5.4 . FIRMWARE UPGRADE PROCEDURE SHALL BE RESISTANT TO POWER OUTAGE 14
5.6 SECURE BOOTLOADER ARCHITECTURAL CONSIDERATIONS 15
5.6.1 SECONDARY PROGRAM LOADER (SPL) 15
5.6.2 PRPL COMPLIANT BOOTLOADER (U-BOOT) 15
5.6.3 NAND FLASH 15
5.6.4 eMMC 16
6 Boot Stages 16
6.1 SECURING ALL BOOT STAGES 16
6.2 RECOMMENDED BOOTLOADER IMPLEMENTATION STRATEGY 17
7 Key Storage 17

7.1 CRYPTOGRAPHIC KEYS AND ALGORITHM REQUIREMENTS FOR THE BOOTLOADER 18
8 References 18

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 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) The 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
CPE	Consumer Premise Equipment
CoT	Chain of Trust
RoT	Root of Trust
UBI / UBIFS	Unsorted Block Images. A Linux kernel flash-handling layer that manages wear-levelling and bad blocks. UBIFS refers to a UBI File System.

OTP	One Time Programmable memory. A memory that could be written only once during the manufacturing process. It can contain code, data, keys.
SOTP	SOTP adds additional security features to OTP so that it can only be accessed by trusted applications.
TEE	The TEE is a secure area of the main chip of a connected device that ensures sensitive data is stored, processed and protected in an isolated and trusted environment.
SPL	Secondary Program Loader. ROM code is the first thing that loads (and executes) other programs. SPL is the second thing that loads (and executes) other programs.
U-Boot, uboot	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.
PRPL, prpl	(The) prpl Foundation
SoC	System on Chip, a SoC integrates the main components including the processor itself in a single chip. In the case of a Gateway this is the Network Processor.
MTD	Memory Technology Device
eMMC	Embedded Multi-Media Card

3.2 Terms

Firmware	The firmware is the software embedded in the Gateway 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, initrd & filesystems.
Flash memory (flash)	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, ROM bootloader) 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.

4 Introduction

This paper introduces the subject of Secure Boot as a companion to the prpl Secure Boot Requirements document [REF].

The document will describe what a secure boot process is, and which prerequisites need to be considered. It further explains the major principles, how they could be applied to all prpl-compliant products, and to align on the implementation efforts of a prpl compliant bootloader. This document will explain how, by starting with a typically hardware SoC-based Root of Trust, a Chain of Trust can be provided (even with an Open-Source bootloader) by leveraging chipset hardware. The resulting boot process launches execution of authenticated and authorized Firmware (FW).

4.1 OBJECTIVES OF THIS DOCUMENT

This document explains how:

- a prpl-compliant bootloader leverages assets located in SOTP to securely boot a process from the SoC's ROM, then typically continues with one or more chipset-specific bootloader(s); and,
- a boot sequence can use keys from SOTP to authenticate and eventually decrypt the encrypted stages of the boot process from power on or reboot.

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 guidance for best industry practice complementary to the prpl Secure Boot Requirements [REF] document, and aims to explain the concepts listed there.

Hardware configuration of a gateway defines the Secure Boot Process' Root of Trust assignment and allocation. OEMs and Operators may have alternative designs that prefer simpler or more complicated key management schemes, hardware capabilities and trust party setups. Choice of Non-Volatile memory type (flash memory) may also impact the implementation on the prpl compliant boot loader for secure boot. We address the options and relevant requirements in prpl Secure Boot requirements [REF] document. For a gateway that aims to comply with prpl secure boot requirements, it is recommended to consider:

- Threats to the gateway and risks to the related businesses;
- Usage of the OTP, SOTP, fuses, and secure storage used to store cryptographic credentials, for example key storage;
- Hardware features and components that are dedicated for trusted execution of critical operations;
- Strategy for patching and upgrading; and,
- Recovery, refurbishment, and graceful failure cases.

An overview of these items is provided in the Device Security Requirements [REF-Dev-Sec] document.

5 Bootloader and Secure Bootloader

We will start this chapter by describing the purpose of a bootloader and why it is an essential security element for the firmware integrity.

5.1 What is a bootloader, how does it work

When powering on a Gateway the software that is performing all steps necessary to put the hardware of the Gateway into a defined state, and perform other operations required, so that execution can be handed over to the main OS or Firmware, is called bootloader.

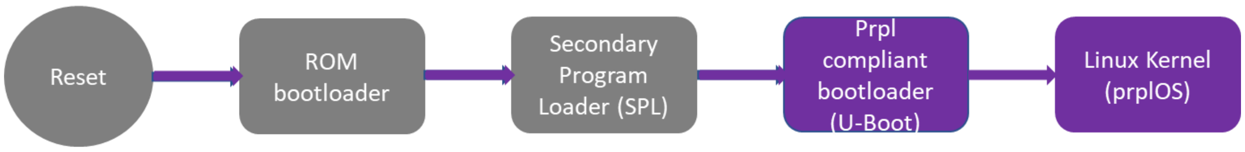
A bootloader typically consists of multiple distinct parts that all serve a specific purpose. At a minimum 2 parts exist:

- A hardware specific part that initializes the parts of the hardware required for the boot process, for instance initializing the DRAM.

- A hardware-independent part.

On embedded system devices, the usage and availability of prevalent open-source solutions have formed a de-facto standard in terms of expected features, supported architectures and functional specifications: The most common silicon independent bootloaders in the embedded world are barebox and U-boot. Most proprietary bootloader solutions in embedded devices are forked from these two projects. That is why, this document addresses the principles over [U-Boot] as a basis of this discussion.

The figure below describes a simplified typical bootloader operation sequence for the prpl compliant bootloader:


Reset
ROM bootloader
Secondary Program Loader (SPL)
Prpl compliant bootloader (U-Boot)
Linux Kernel (prplOS)

1) Upon a reset triggered by Software or by hardware (i.e., power on) the gateway starts to execute the immutable (secure) ROM bootloader. The reset vector points directly to the ROM bootloader to ensure an unmodifiable, small codebase is executed upon the reset being released. The ROM bootloader configures the SoC and necessary peripherals to load and executes the Secondary Program Loader (SPL). On typical gateways the SPL will be read from flash and copied to RAM. Typically, this will be SRAM/Cache inside the SoC, in rare cases it might be DDR.
2) For a gateway on which secure boot is implemented and enabled, the ROM bootloader will check the signature of the SPL and may decrypt it in case it is encrypted before it hands over execution to it.
3) The SPL will then initialize all the hardware that is necessary for the next stage bootloader (i.e., U-Boot) to be loaded and executed. This typically involves initializing the DDR and may include initializing the Trusted Execution Environment (TEE). Once the next stage bootloader has been loaded into RAM, for a gateway on which secure boot is implemented and enabled, the SPL will check the signature of the next stage bootloader (U-Boot) and decrypt it in case it is encrypted before it hands over execution to it.
4) The main purpose of U-Boot is to locate the Linux kernel as well as the root filesystem that belongs to this kernel in the device's flash. It then loads the Linux kernel into RAM and authenticates, decrypts, in case it is encrypted, and decompresses it. The same can be done with the root filesystem for the Linux kernel; alternatively, the Linux kernel will, once it is running, authenticate, decrypt, and decompress the individual files needed on the fly.
5) Once the Linux kernel image has been authenticated and copied to RAM. Execution control is handed over to the Linux kernel together with a set of parameters, most important the one that allows the Linux kernel to find its root filesystem.

5.2 SECURITY CONSIDERATIONS FOR THE BOOTLOADER

The Gateway is an important piece of Hardware, by which an operator provides services like “Access” or “Video” to its subscribers. Correct and predictable function of the Gateway is of high importance for the operator to allow giving a reliable service to the user.
This leads to some primary security considerations for the bootloader:

- The bootloader should be built around a Chain of Trust.
- All firmware components that are in the Chain of Trust are verified and authenticated.
- Ensuring that the router always boots a verified and authenticated state.
- Optionally denying reverse engineering of firmware, meaning:
    - Optionally encrypting all relevant software and data on flash.
- In the case of a running CPE being compromised, ensuring that possible changes to the firmware components running currently, that are in the Chain of Trust, are not persistent.
- Ensuring the refurbishment process does not compromise boot security.
- Ensuring that with simple hardware modifications (e.g. soldering of wires to enable a console or debug port, desoldering/resoldering of flash) the security aspects like integrity and confidentiality are not compromised.

The Gateways Hardware and Software must protect confidential credentials, which in the hands of an attacker could allow breaking the security, from unauthorized exposure and usage.

5.3 What is Root of Trust (RoT), what constitutes a Chain of Trust (CoT)?

Root of Trust:
The Root of Trust (RoT) is generally a public key that is responsible for authenticating the first code that is to be run (typically the SPL) using public key cryptography.

Chain of Trust:
A Chain of Trust (CoT) will ensure that each software component that will be loaded has not been tampered with. In practice, this means that each new Software component that is loaded will be authenticated, typically by the previous stage.

5.4 Security considerations pertinent to operators

The key elements for the overall Gateway security are described in the prpl Device Security Document [REF-Dev-Sec]

5.4.1 Main security building blocks

The main components required for a secure boot process are:

- Asymmetric and symmetric cryptographic Keys, X509 certificates used for encryption and signature of bootloader stages and firmware.
- Bootloader and Firmware image signing and encryption which is done as part of the build system as well as on the Gateway firmware signature verification and decryption done by the bootloader.
- All bootloader stages will deny transferring control to the next stage of the boot process if the signature of the next stage image cannot be verified.
- When doing a firmware upgrade, once downloaded, before flashing, a signature verification of the firmware to be upgraded to, should be performed.

5.5 Collected operational constraints

The next section will list some best practices relevant to secure boot. The practices listed below are relevant to the Secure Boot Process and are based on Operator Business' constraints and limitations.

5.5.1 Device refurbishment process

To allow the re-use of devices which are returned to the Operator, or in some cases to allow a refurbishment process triggered by the customer, a device refurbishment process is required.
The refurbishment process and the specific operations that are performed to the Gateway are usually specific to the needs of the Operator. In practice a device refurbishment process on the Gateway is triggered while the Gateway boots. It is a part of the bootloader operation. Detection of the specific trigger event and starting the refurbishment process will then be handled within U-Boot.
Device Refurbishment is usually performed in a non-secure environment especially when it is subcontracted or performed by the end user. To still have an authentic device, it is important that device refurbishment activities are performed securely, ensuring that none of the operations performed during refurbishment will flash unauthorized software or configuration to the Gateway.
The security mechanisms put in place for a firmware upgrade should be used as part of the refurbishment process.

5.5.2 Firmware upgrade and firmware rollback mechanisms

Mass upgrade operations may result in errors or large-scale problems which may force an operator to cease the upgrade operation. This action may result in a partial upgrade in deployed gateways. Some sporadic errors may also cause issues in deployed gateways long after the upgrade is completed.

Because of these reasons it is advisable to implement a rollback mechanism.

A rollback mechanism allows reversion to an older version of the Firmware. This older version of the Firmware may either be in the Gateways flash already or may be downloaded as part of a Firmware upgrade – in this specific case a Firmware downgrade.

Vulnerable rollback mechanisms may create a downgrade attack surface. That is why downgrading a vulnerable version of the FW must be denied by the Anti-Rollback controls. We address the anti-rollback requirement in [REF] section 7.

5.5.3 Specific bootloader keys for development gateways

Secure boot designs rely on critical assets such as RoT Keys and optional critical assets such as encryption keys and other crypto material. Access to these keys must be extremely restricted and in case these are used in the development phase, the OEMs and Operators may face a key compromise risk.

When these critical assets are compromised, everyone in the possession will be able to generate their own bootloaders and firmware for the Gateway. During the boot process the Gateway will not be able to detect that this Firmware that might contain malicious code was not authorized by the Operator.

Accidental disclosure is also an issue which can be seen in the market.

The Kerckhoff principle that says that a cryptosystem is designed to be secure even if everything about the system is public knowledge, except the key, emphasizes that the production assets must be well protected against disclosure.

Key owners of the system are advised to implement non-production keys for development environments. This ensures segregation of production keys and limits the risk of key compromise. We have listed the requirements for compliance in [REF] section 9.

5.5.4 . Firmware upgrade procedure shall be resistant to power outage

Multiple Firmware upgrade strategies exist like loading a full image to RAM before flashing it, loading the full image to a dedicated partition, or upgrading in smaller blocks. Each of these mechanisms has their specific strength and weakness.

There are possible errors that may happen during the Firmware upgrade process. These errors may be caused by:

- Power and network outages
- Bugs in upgrade code
- Invasive attacks on the gateway
- Process flow-related mishandlings
- Hardware failures
- Non-Compliant implementations

An upgrade process must be resilient to these possible errors. The requirements on the upgrade resilience are listed in [Ref-Dev-Sec] section 2.3.

Depending on the gateway’s available volatile and non-volatile memories, operators are encouraged to implement an upgrade flow that is resilient and easy to recover from in case of a failed upgrade attempt. Requirements in the [REF] Section 8 address these error cases.

5.6 Secure bootloader architectural considerations

5.6.1 Secondary Program Loader (SPL)

The Secondary Program Loader (SPL) is used to initialize the SoC of the gateway to allow it to interact with the peripherals necessary for the boot process. Meaning to initialize the hardware to allow the prpl compliant bootloader to be loaded, authenticated, and executed. It may initialize the TEE so that the prpl compliant bootloader can make use of it.
This process is different for each SoC manufacturer and linked very closely to the SoC specific architecture.
While prpl has defined security requirements for the boot process, securing the SPL remains part of the SoC supplier's responsibility and domain.

We aim to abstract out this complexity and guide the reader to a common prpl compliant bootloader to:

5.6.2 Prpl compliant bootloader (U-Boot)

The prpl complaint bootloader makes use of U-Boot.
While the SPL will initialize the SoC to be able to load the prpl compliant bootloader the early stages of the prpl compliant bootloader may need to initialize hardware necessary for it to perform its actions.
Dependent on SoC and SPL architecture it may have to:

- Initialize DRAM and flash access if not already done by an earlier boot stage.
- When needed, load vendor specific firmware that may be needed for instance to allow refurbishment via Ethernet, or to manage LED's, displays etc.
- U-Boot may handle parts of the firmware upgrade procedure.
- Locate a good version of the next stage of the boot process, like the Linux kernel, and load it to RAM.
- Validate the next stage (may also decompress and decrypt it) before it starts executing it.

5.6.3 NAND flash

NAND Flash memory media have individually erasable segments, each of which can be put through a limited number of erase cycles before becoming unreliable. Some specific flash blocks can also suddenly become unreadable, unwritable or have bad CRC. Consequently, on designs with controllerless NAND, each software part trying to read/write to or from the flash needs special algorithms to correctly handle the flash peculiarities like wear levelling and bad blocks management. The bootloader itself must read the flash while trying to load a Linux kernel image in memory therefore, such flash management code must be embedded.

The very first code that is executed cannot be located in just any flash block. A flash block error in this case would brick the gateway. NAND flashes have a special characteristic for the first block that is guaranteed to be always valid. This is the area in flash where the part of the bootloader (typically SPL) is located, which can perform the bad block handling task.

Bad block handling may already be implemented in the bootrom in which case constraints on the size and location of the SPL can be less restrictive. Typically, the first stage boot loader is located in flash block 0.

Because the firmware upgrade mechanism will be achievable both by the bootloader "Reflash on LAN" feature and by the Gateways' firmware upgrade code defined by TR069, they MUST share the same approach for flash management as they will operate on the same flash locations.

Besides MTD, typically UBI and UBIFS are used to partition the flash and provide a filesystem inside the partitions.

5.6.4 eMMC

eMMC is a flash storage device made up of NAND flash memory that is controlled by a dedicated controller. This controller takes care of bad block handling and wear levelling of the underlying NAND flash. Both bootloader and applications can access the partitions on the eMMC, while the controller of the eMMC takes care of handling the flash itself.

eMMC provides some enhanced security features that can be used on the Gateway. Write-Protect to protect partitions against unauthorized modification, and RPMB (Replay Protected Memory Block) a special small area that is protected against replay attack.

6 Boot Stages

It is usual to have several boot stages before launching the software that will pilot the Gateway once fully booted. It is not uncommon to have, for example, to execute three concatenated boot stages. There are multiple factors explaining this:

- The immutable bootrom might load an early-stage bootloader (SPL) in secure memory/internal SRAM and not general DDR for security concerns but also allowing the SPL to handle the DDR configuration. The amount of secure RAM/SRAM is very small, and it may not be sufficient to contain a full-fledged bootloader like U-Boot.
- Chipset vendor proprietary bootloader may not provide all functionality required by the operator. They should be implemented in U-Boot.
- Due to license restrictions, not all boot related code might be suitable to be put into U-Boot.
- Each stage of the bootloader must authenticate the next stage of the boot process before it passes execution to it. We documented relevant principles in [REF-Dev-Sec] Section 4.1.

6.1 SECURING ALL BOOT STAGES

Cryptographic Key schemes in a secure boot process may vary depending on the silicon architecture. The keys used by earlier stages may not be accessible to the prpl compliant bootloader or any later stage. Keys and cryptographic operations may not be handled by U-Boot components to ensure security. Access controls over these may be decided by a higher privilege level like a TEE or secure enclave.

prpl must enable the following Secure Boot Possibilities for the Second Boot stage:

- U-Boot implementation based 'common' Secure Boot Flow
- SoC Vendor SoC Secure Boot flow if supported by former - This could leverage Boot ROM, SOTP, and/or TEE supported in the SoC implementation.

This allows operators to leverage either of the approaches as best suit their security and operational requirements and provides a possibility for innovation.

Please note that in some solutions a TEE may not be available before the last stage prpl compliant bootloader.

6.2 RECOMMENDED BOOTLOADER IMPLEMENTATION STRATEGY

Chipset vendor, SoC bootloader code is very specific to the chipset for initialising RAM, Flash, eMMC, specific IO controllers, security engine. Errors or corruptions in this code could render a Gateway unusable and difficult to refurbish, therefore this code should only be modified if the impact of not upgrading is critical to the business. Making modification in this type of code would make implementation very SoC dependent and not portable across processors from different vendors.

We recommend the following implementation strategy:

- Modify vendor specific bootloader code only to securely provision the extra cryptographic keys needed if no other mechanism exists for this purpose.
- The prpl compliant bootloader may require an additional bootloader stage that will fulfil the functional requirements described in [REF], if the operator/OEM does not want to make use of the last stage bootloader provided by the silicon supplier and modify it wherever necessary.
- We recommend using U-Boot source code as this last stage bootloader code: it already contains most of the needed features and there are active contributors that implement missing features as well as maintaining the code base.
- This last stage bootloader code should be placed at least twice in persistent storage so that it can be safely upgraded.
- SPL code could be stored at least twice too in case the ROM bootloader supports it (SoC specific)
- In case of raw NAND flash storage, the two U-Boot image copies should be placed in an UBI volume to manage bad blocks and dynamic resizing.
- In case of eMMC flash storage, the various boot loaders should be stored redundant.

7 Key Storage

Cryptographic algorithms based on keys can be either symmetric or asymmetric. In the case of symmetric algorithms, any unauthorized user getting access to a key is considered as a key compromise. Keys are used in decryption of content that needs to be confidential. For asymmetric algorithms, there is a public key and a private key pair. Only the private key needs to be protected as it is the one that is used to sign and decrypt. The public key is used to validate that the signed content is authentic or to encrypt the original content. As the naming indicates, public keys could be known to anybody, and shall be protected from being modified on the Gateway. For this reason, public keys are usually stored in immutable areas such as eFuses or OTP. Securely storing the hash of a public key in an immutable area is one practice for resource limited environments.

7.1 CRYPTOGRAPHIC KEYS AND ALGORITHM REQUIREMENTS FOR THE BOOTLOADER

We addressed the key operations related requirements in [REF] Section 9.1.2. to comply with these requirements the bootloader code should be able to utilise cryptographic keys for:

- Each stage of the boot process to allow verifying the authenticity of the next stage in the boot process. For the first stage, the type of keys used and the algorithms that are necessary, are usually set by the silicon in use. While they are an important part in the Chain of Trust they are fixed and not part of the discussions in this document
- Verifying firmware signature and decryption of firmware before starting decryption of the firmware image in case of reflash from LAN, dedicated Web page, USB upgrade, upgrade via TR069 where U-Boot decrypts the next stage in the CoT.

8 References

[REF] prpl Secure Boot Requirements v1.0 https://prplfoundation.org/wp-content/uploads/2023/05/prpl-Secure-Boot-Requirements-v1.0.

[REF-Dev-Sec] prpl Device Security Requirements Version 1.0 https://prplfoundation.org/wp-content/uploads/2018/04/prpl-Device-Security-Requirements-v1.0.pdf

[U-Boot] The U-Boot Documentation https://u-boot.readthedocs.io/en/latest/index.html