OPEN

Compute Project

Hardware Secure Boot

OCP Security workgroup

EDITOR: Bryan Kelly, Microsoft Corporation

CONTRIBUTORS:
Yigal Edery, Kameleon
Joe Foster, Microchip
Ahmed Abbas Hassan, CyShield
Bryan Kelly, Microsoft
Nate Klein, Google
Jubin Mehta, Facebook, Inc.
Alberto Munoz, Intel Corporation
Rajeev Sharma, Open Compute Project
Wojtek Powiertowski, Facebook, Inc.
Eric Spada, Broadcom, Inc.
Brett Henning, Broadcom, Inc.
Osman Koyuncu, Nuvia Inc.
Ben Stoltz, Google

Revision History

Revision	Date	Guiding Contributor(s)	Description
1.0		Bryan Kelly, Microsoft Corporation	Initial Release

Executive Summary

The environment. In cloud data centers, the composition of the cloud server consists of a variety of processing devices and peripheral components such as, accelerators, storage devices. These devices typically all run firmware, which can reside internally in the device or externally to the device.

The problem to be solved. Until this document, servers had no standardized, open, and consistent mechanism to provide authorization of the devices that comprise the server. For example, does a network adapter still contain the initial firmware that was installed by its manufacturer? Has the device been reflashed or updated with firmware from another entity? Cloud service providers cannot rely on procurement agreements or attestation alone to ensure the integrity of the products they buy. These products must have protection mechanisms that enforce integrity during design, development, manufacturing, testing, shipping, provisioning, installation, and operation. Allowing unauthorized code to execute on the device would create innumerable possible attacks.

The Open Compute Project solution. This document presents a design for enforcing firmware integrity on the components within a server. In this design, components are required to have certain capabilities to enforce firmware integrity and maintain trustworthiness across the product life cycle.

The platform components must:

1. Provide a mechanism for securely anchoring a root of trust public key.
2. Verify the device firmware digital signature using the anchored public key
3. Provide a mechanism for revoking previously signed firmware

This document identifies required and optional functionality for platforms and attester devices. Feedback on version 1.0 of this document is invited, especially from vendors implementing it.

1. Purpose

This document is intended to guide the implementation of expected behavior for secure boot for hardware products, and the implementation maps to OCP Security workgroup standards/solutions.

2. Audience

The audience for this document includes, but is not limited to, system and system component designers, security information and event management (SIEM) system developers, and cloud service providers.

3. Scope

1. Authenticating all mutable firmware in system devices (flash for BIOS, BMC, microcontroller(s), CPLD, etc)
2. Existing standards/protocols:
    - 2.1. NIST SP 800-193 “Platform Firmware Resiliency Guidelines”
    - 2.2. NIST SP 800-57 “Recommendation for Key Management”
    - 2.3. NIST SP 800-147 “BIOS Protection Guidelines”
    - 2.4. NIST SP 800-147B “BIOS Protection Guidelines for Servers”
    - 2.5. NIST FIPS 186-4 Digital Signature Standard (DSS)

4. Syntax and conventions

In this document, the words “shall”, “must”, “shall not”, “must not”, “should”, “recommended”, “should not”, “not recommended”, “may”, and “optional” follow the conventions in IETF BCP 14 (RFC2119, RFC8179) 2119. The roles “attester”, “verifier”, and “reference integrity measurements” are defined in the draft Reference Terminology for Attestation Procedures.

Definitions

Throughout the OCP specs, the terms Secure Boot and Measured Boot are used to describe required capabilities. Since these terms can have varied interpretations, this section defines the meaning intended for these in the OCP specs.

Secure Boot is the mechanism that verifies the integrity of every code being loaded, before it's allowed to execute. This includes, for example, checking code for proper signature by an approved signer. Secure Boot is considered successful if the integrity check passed, and fails if it didn't. Note that failure in Secure Boot does not necessarily imply "bricking" the device, but rather following a recovery policy which defines what to do if the check fails. This spec covers in detail the expected behaviour of secure boot.

Measured Boot is used to describe the mechanism that collects measurements (hashes) of every code being loaded, and reports them in a secure way via the process of attestation. Measured boot lets the verifier of the attestation process (which normally happens after the code was executed) to decide what is considered good/bad and apply relevant policies. The OCP attestation spec covers the attestation protocol as well as other aspects of measured boot.

In the context of OCP specs, and to avoid confusion, we explicitly avoid using any other industry terms such as **verified boot**, **trusted boot**, etc. These terms tend to be interpreted differently by different people, and are usually a source of confusion.

Guiding Principles

Secure boot shall be designed in a way that secure boot integrity cannot be compromised by mutable code.

Secure boot code execution path shall be designed in a way that authentication cannot be compromised by hardware glitching, such as voltage manipulation or altering of efuses.

Secure boot must authenticate mutable firmware with a hardware protected reference, hence the protection mechanism of mutable code shall be immutable.

Mechanisms that control the secure boot or configuration of secure boot shall be attestable. For guidance on attestation, review Attestation of System Components document.

Secure boot implementation shall permit complete transfer of ownership or control of the device. Example: invalidating supplier keys with customer keys etc.

Secure boot shall be designed in a way that permits for key revocation or key rotation of the signature verification key.

Secure boot shall permit secure provisioning and secure update of the device.

Secure boot shall perform authentication of mutable firmware code.

Secure Boot can optionally implement device unique encryption for confidentiality protection of firmware images. Confidentiality protection does not negate the need for digital signature and integrity protection of firmware.

5. Secure Boot Requirements

The first stage of secure boot is an immutable source (for example: mask ROM) and shall enforce authentication of the next stage mutable bootloader.

The authentication mechanism shall use either a public key, hash of a public key, table of public keys or table of hashes (for example: SHA2 of public key), provided the key or hash table is immutable and not dependent on mutable code or measurements. They are therefore referred to as hardware rooted, or root keys.

No local secrets such as an asymmetric private key shall be used to authenticate any stage of device.

Each boot or reload of a device shall enforce authentication of mutable code by the immutable source.

A means for revoking or re-keying authentication keys or hashes of keys shall be provided. For example, by revoking or invalidating keys stored in One Time Programmable (OTP) memory by changing the state of revocation designated OTP memory.

Key hashes or table hashes shall have a minimum strength of SHA2-384,

Hashes and mutable code digests shall support a minimum of SHA384 and follow FIPS 186-4 4.2 on the security strength of the hash function.

RSA and ECDSA are the recommended Digital Signature Algorithms for signature generation, and follow the strengths in the CNSA Guidance.

It is recommended that any secure boot public keys or hash digests of key tables stored in OTP for the digital signature verification provide a mechanism of error detection to protect the integrity of the anchored keys or digest, for example: carry parity bits in OTP that are verified by the immutable source enforcing secure boot.

It is recommended that sufficient space be provisioned for more than one device secure boot root key or root key digest. To support transfer of ownership and renewal of secure boot keys. The Root key is the immutable reference described in paragraph two above.

Immutable source shall not permit any unauthenticated mutable code to execute on the device prior to successful digital signature verification of the code.

Unauthenticated ROM patching and unauthenticated code shall not be permitted.

Boot and secure boot authentication failures shall not render the device unrecoverable, however all recovery paths shall enforce digital signature verification of mutable code.

It is recommended that devices provide a mechanism for logging secure boot failures.

If encrypted firmware images are supported, decrypted firmware shall only be accessible from internal memory while performing firmware security operations such as digital signature verification.

6. Revocation and change of ownership Requirements

- Revocation of secure boot keys should be designed in a manner that does not permit unwarranted change of ownership (theft) of a device. Example; a vulnerability in firmware that is exploited should not allow an attacker to change OTP in a manner that allows programming of a new key and revocation of the current key.

- Manufacturers shall not maintain master unlock keys or hashes, escrow security, backdoor keys or recovery keys. Manufacturer keys should get completely replaced by customer keys, or in the instance of dual signature devices, the manufacturer keys should become secondary signatories to the primary customer keys.

- In devices designed with dual signing enabled, both signatures shall be equally enforced. Example: A device that supports a manufacturer signature and customer signature, both signatures shall be used. The manufacturer's signature shall not override customer signature, and vice versa.

- If additional OTP memory or similar protected storage is provided for key revocation and new key or hash programming, it shall be possible to program multiple keys into a device, whereby no further key adding is possible.

- Revocation of the current key or firmware manifest requires the device be booted from a signed and authenticated image with one or more of the following attributes to inform revocation flows:
    - A flag and metadata instructing key to add or key index to activate, and key or index to revoke.
    - A flag and metadata indicating new key to be added or key index to activate
    - A flag and metadata indicating key index to revoke or key to deactivate.

    - A flag and metadata indicating numeric value of manifest to revoke.
- It is recommended the revocation may be multi-phased across activation and deactivation images, where key or manifest activation occurs before revocation of the previous version. To prevent latent A/B redundant image failures, or recovery failures after revoking keys, revocation processes may include updating redundant and recovery images.

- Revocation of active keys, shall only be performed if there remains one valid key. In this circumstance, the final key in the revocation cycle becomes persistently valid when all other keys or hashes that have been revoked.

- Revocation of secure boot keys shall not be mixed with anti-rollback revocation OTP for firmware components. It is permissible to use key revocation and key rotation with anti-rollback. See section: 10. Revocation and Anti-rollback

7. Required Algorithms

Measurements and Digests

For mutable code digest/fingerprint generation:

- SHA2-384 or higher

Secure Boot Signatures

For Signature generation follow strengths in: CNSA Suite Guidance one or more of the following algorithms:

- RSA PSS PKCS#1 v2.1, 3072 bit key length or higher
- ECDSA 384 bit key length or higher

Signature generation shall follow:

- Digital Signature Standard (DSS) FIPS 186-4

Encryption Algorithm

For flash encryption algorithm: AES 256-bit

Strength of Algorithm

For Key digest generation:

- SHA2-384 CNSA Suite Guidance

*Entropy, random bits, symmetric keys, and private asymmetric keys **must** be generated within the device itself, in a hardware security module, or locally, in a device with the following properties:*

- *Follows recommendations in NIST Special Publication 800-90A Rev 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators*
- *Follows recommendations in NIST Special Publication 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation*
- *Follows recommendations in NIST Special Publication 800-133 Recommendation for Cryptographic Key Generation*
- *Complies with Annex C: Approved Random Number Generators for FIPS PUB 140-2, Security Requirements for Cryptographic Modules*
- *Validated at overall level 2 or higher under FIPS 140-2 SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES or Security Requirements for Cryptographic Modules, FIPS 140-3*

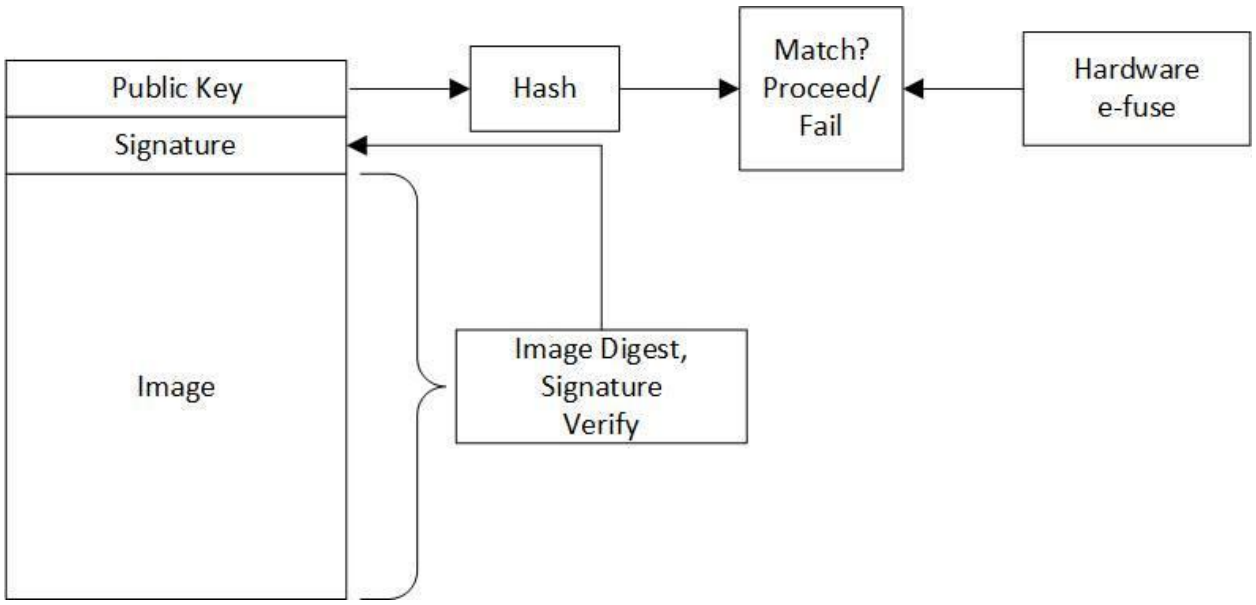
Follows the guidance in the Commercial National Security Algorithm Suite regarding quantum resistant algorithms and key sizes.

8. Secure Boot Flow

Secure boot enforcement shall be immutable. The secure boot immutable source is responsible for authenticating the first stage mutable code. Once authenticated the first mutable code can be responsible for authenticating the next stage of boot. The acceptable digital signature verification algorithms can be found here: Recommended Algorithms

Embedded Public Key Flow

In the following example, the ROM is the immutable Root of Trust and follows the below sequence in authenticating a firmware image using a digest of the root key as an immutable anchor in hardware fuses:



In the above example:

1. A hash of the Firmware Signer's public key ($FWSignK_{pub}$)is provisioned in immutable efuses at provisioning. This public key is the root key anchored in hardware, and used by ROM for comparison to the public key provided with the firmware image for digital signature verification.
2. ROM reads mutable metadata, the Firmware Signer's public key and first stage mutable boot loader from a known location on mutable storage into internal SRAM. Note: the ROM reads both key and image into internal SRAM to avoid time of use time of check attacks. Note: No interfaces or external ports should allow access to this region of internal SRAM while digital signature verification and secure boot is occuring.

3. After ROM hashes the Firmware Signer's public key now in SRAM, it compares the hash value to the e-fused digest of the provisioned Firmware Signer's public root key.

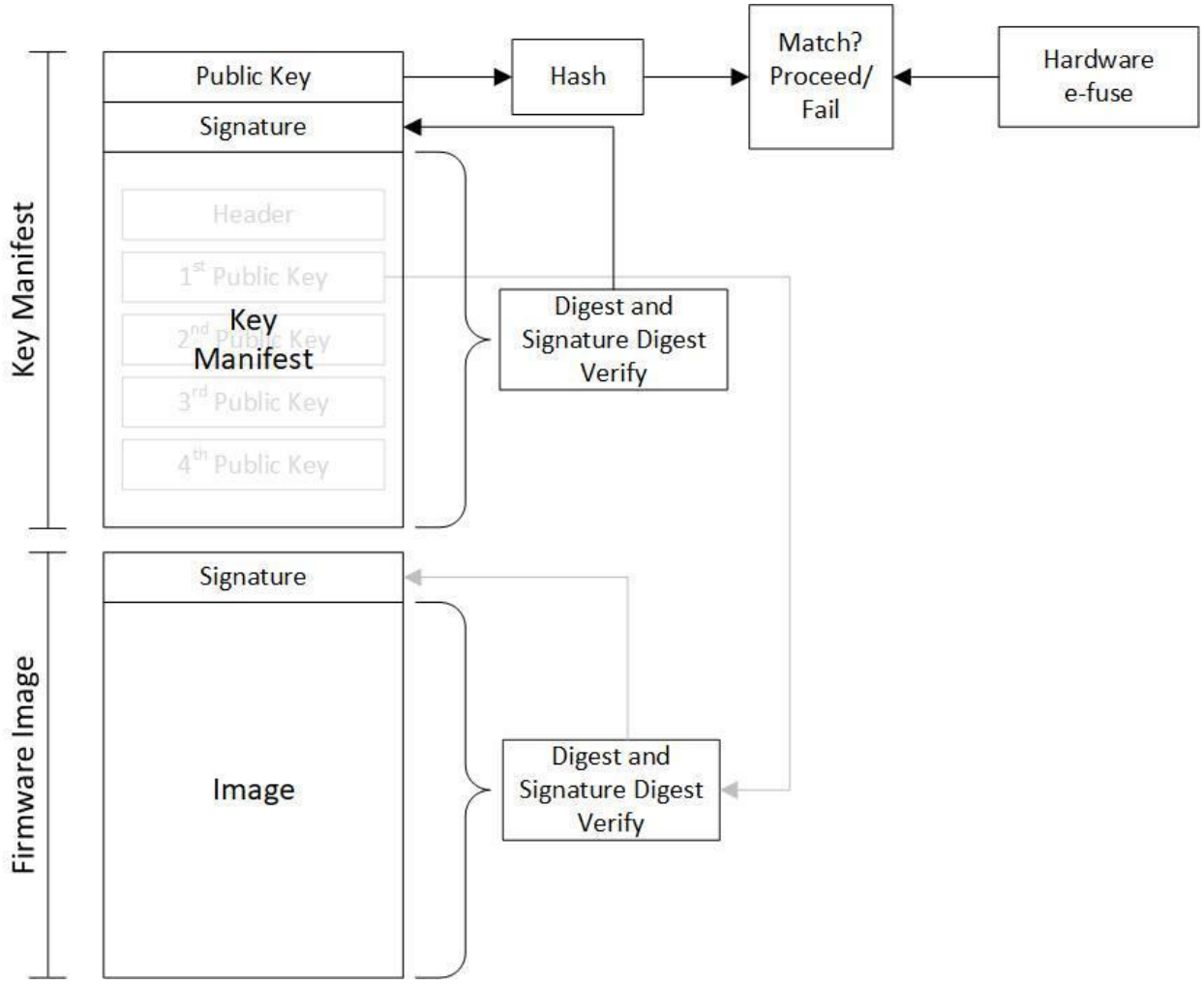
   If matching; the ROM uses the Firmware Signer's Public Key in SRAM to verify the signature on the first stage bootloader, taking a digest of the first part of mutable code and comparing it to the digest used for the signature.

   If not matching, ROM fails over to the next boot device and looks for a public key or hash table. Upon failure, the device shall log the failure in accordance with the specification: Recovery Specification, and seek to enter a recovery flow as per the specification and NIST 800-193.

Key Manifest Flow

A typical implementation of secure boot using the above methodology would provision a Device Update Public Key ($DevUpdtK_{pub}$) in immutable OTP at provisioning and have a key manifest as the first part of mutable code. The ROM would therefore authenticate the key manifest using the Device Update Public Key, following the above-mentioned sequence, then use a Firmware Signer's public key ($FWSignK_{pub}$) from within the key manifest to verify the first part of mutable code.

The block diagram below depicts the manifest flow:



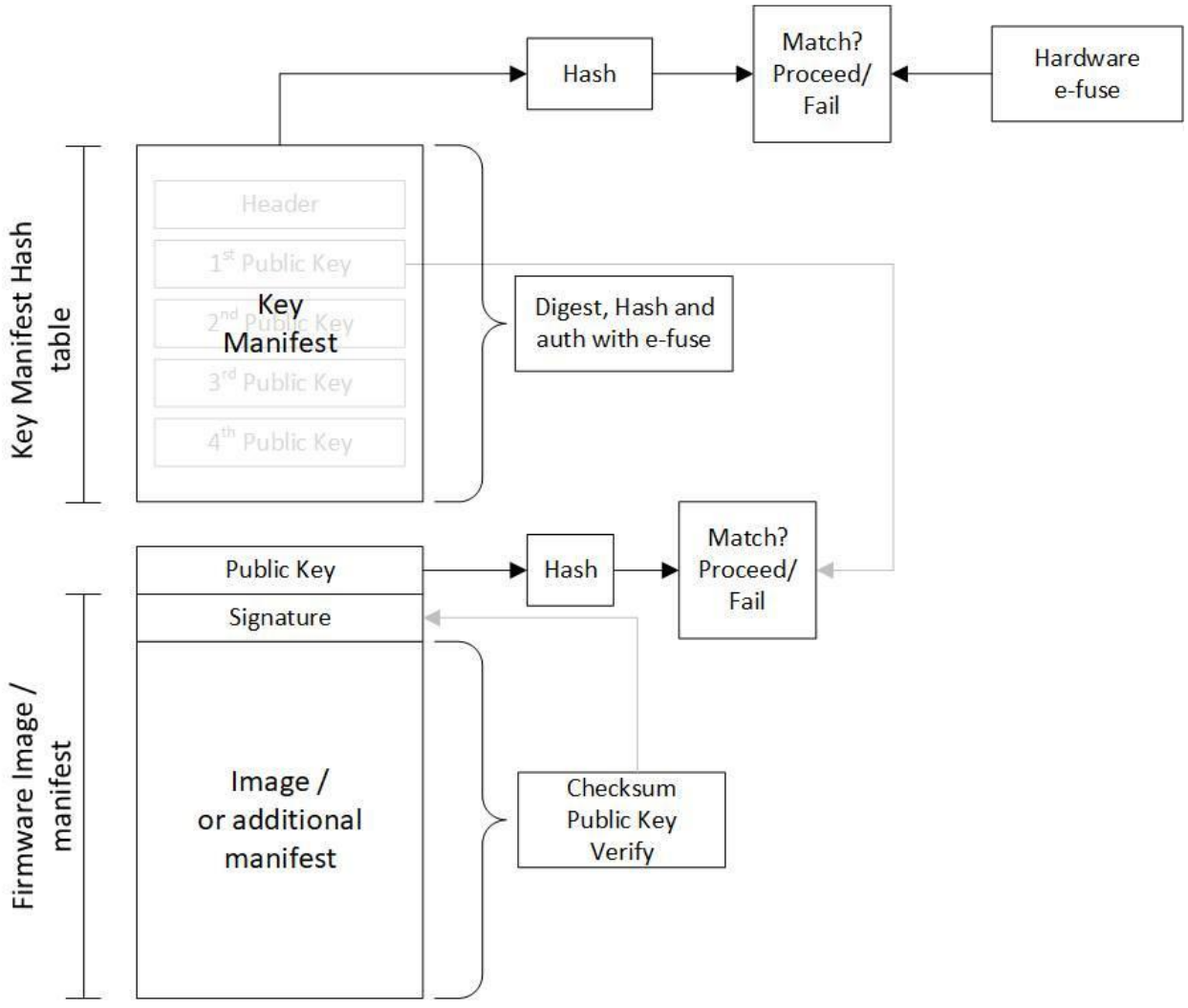
There are a few advantages of using the above key manifest approach:

- The flow promotes using hardware anchored keys less frequently, and only signing key manifest with the private keys for the hardware anchored public keys. It is good practice to store all private keys securely in a Hardware Security Module (HSM) and restrict key usage. One advantage of the manifest approach is the private keys contained in the offline HSM that correspond to the public keys in hardware are used less frequently. The keys contained in the manifest are used for firmware signing, and therefore used more frequently. The immutable "Root keys" in hardware efuses are used only for signing key manifests, which would be updated less often.
- The mutability and depth of the key manifest permits time based revocation of keys in the manifest.

- The flow allows for the partitioning of offline HSMs, whereby HSM containing keys for firmware signing can be accessible to secure build systems while the HSM containing keys for signing key manifests can be further air gapped.

Manifest with Hash Table Flow

Similarly, a hash table with digital signature authentication by ROM can exist on flash. Typically the hash table method consists of a list of digests of asymmetric public keys. The digests in the hash table are then used to compare against keys provided with mutable code as signatory of the mutable code. If the digest of the key matches the digest of an unrevoked key in the manifest, then the key is used to verify the mutable code digital signature. This flow is a take on the above asymmetric verification. An example of the implementation is as follows:



In all cases the Immutable Root of Trust must be used to verify mutable code and generate measurement for verification. Secure boot implementations should not rely on firmware only update authentication mechanisms, as vulnerabilities in firmware can be exploited and then persisted.

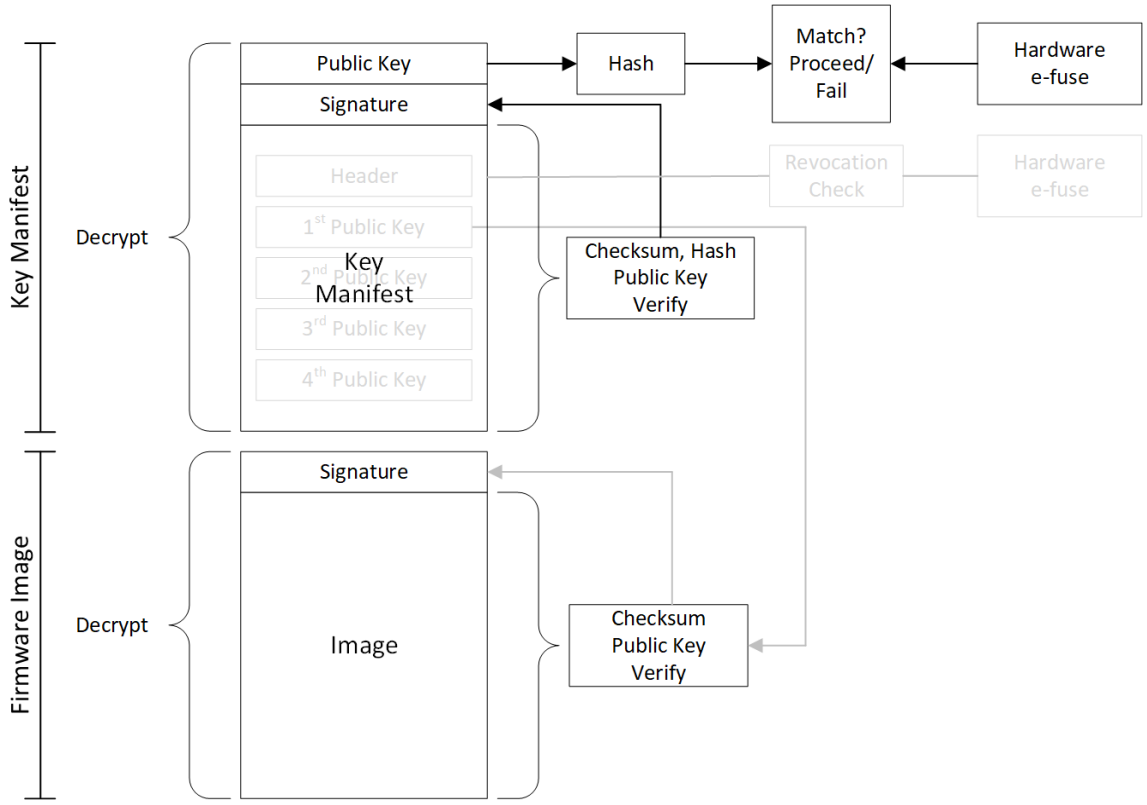
Note: It is recommended that the keys used for authentication of mutable key chain structures and code be extended into the attestation measurements.

It is recommended that keys anchored in hardware sign other keys, which then sign firmware. This is the purpose of the proposed manifest/key chain flow.

9. Confidentiality

Flash encryption does not negate the need for authenticated secure boot, however it can improve the security posture of a device and offer a layer of defence against targeted attacks on firmware. Flash encryption through a device unique key can provide confidentiality of flash contents.

When a device supports flash encryption, the device ROM would copy flash contents into SRAM decrypting the firmware block as it copies into internal memory and before following the authenticate flow described in: Secure Boot Flow



Flash encryption should be configurable as immutably on or off in hardware. Decryption occurs then digital signature verification.

10. Revocation and Anti-rollback

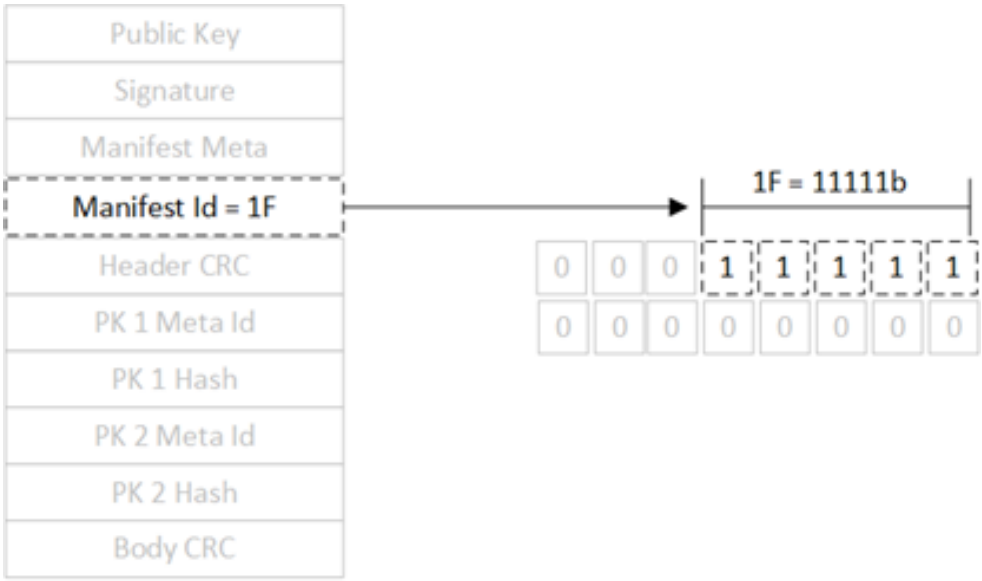
Key revocation assists in restoring integrity to a product should the keys or hash table become compromised. In certain circumstances key revocation can also be used to assist with anti-rollback where firmware cannot be trusted to guarantee rollback prevention. Example implementations of anti-rollback include monotonic id or security version numbers in firmware, key manifests and hash tables.

Used in conjunction with key manifest IDs, or hash table offsets, revocation can be achieved with very minimal OTP memory bits. If there are constraints on OTP memory, a single byte can be used to invalidate 8 different keys whether using hash tables, individual keys or key manifests.

Devices generally require more revocation storage, as each independently updatable firmware component may have its own anti-rollback.

Example:

Where key manifests are used in Secure Boot implementation, a manifest header could contain an ID that the ROM verifies against revocation areas in OTP. In this instance, revocation can be achieved by setting a new key manifest ID in the header of a signed key manifest. Upon boot the ROM would authenticate mutable code using the methodology described in the "Secure Boot" section of this document. After successfully authenticating the firmware, the ROM checks the key manifest ID. If the ID is incrementally higher than the latest OTP memory fused area, then the next OTP memory area is programmed one increment higher.



Revocation can only be achieved with a valid manifest id, one increment higher than the current manifest ID, with revocation bits set in the header manifest.

To prevent boot issues on the next reboot of the product, recovery firmware must subsequently be updated. Alternative, A/B methods whereby recovery or backup areas of flash are first programmed with newly signed manifests or images are first tried.

- Devices with multi-stage bootloaders and firmware images that are chain loaded or individually signed should include security version numbers in their signed metadata.
- Devices should support secure and persistent monotonic counters, resilient to rollback attacks, or secure one-time programmable storage for firmware anti-rollback version numbers.
- Devices shall implement an anti-rollback mechanism that prevents older versions of validly signed firmware from running on a given device.
- Devices may implement separate anti-rollback for recovery firmware and production firmware.

11. Device Recovery

Secure boot authentication failures shall not render the device inoperable, nor should it permit the device to load mutable code without verifying integrity and signature.

Devices that fail to locate and load a secure firmware image shall continue to attempt to load over their permitted firmware load interfaces. These interfaces may include: A/B SPI flash images, I2C, PCIe, USB or other bootable interfaces on the device. Failing to load a valid digitally signed image on any or all interfaces shall not halt the device or prevent the device from retrying interfaces. All failed boot attempt events should be signaled or logged in tamper evident log and made available as part of recovery.

Failures in secure boot due to a detected event, or unexpected path detected by code flow integrity techniques such as code checkpoints or code integrity watchdogs may reset the device, but shall avoid halting the device indefinitely. The device may take an extended and varied time to reset as a countermeasure to watchdog reset timing attacks.

Devices using special purpose recovery firmware to bring the device to a recoverable state, shall enforce secure boot digital signature verification over all mutable code loaded into the device, and any firmware downloaded there after shall continue to verify digital signatures. Information on device recovery can be found in the Recovery document.

12. License

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP's request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at
http://www.opencompute.org/participate/legal-documents/.
If you have any questions please contact OCP.

13. About Open Compute Foundation

The Open Compute Project Foundation is a 501(c)(6) organization which was founded in 2011 by Facebook, Intel, and Rackspace. Our mission is to apply the benefits of open source to hardware and rapidly increase the pace of innovation in, near and around the data center and beyond. The Open Compute Project (OCP) is a collaborative community focused on redesigning hardware technology to efficiently support the growing demands on compute infrastructure..For more information about OCP, please visit us at http://www.opencompute.org