Ownership and Control of Firmware in Open Compute Project Devices

Elaine Palmer (erpalmer@us.ibm.com), Tamas Visegrady (tvi@zurich.ibm.com), and Michael Osborne (osb@zurich.ibm.com), IBM Research Division

9 November 2018

1 Introduction

A country music song made famous by Garth Brooks in 1990 declares, "I've got friends in low places," noting that one can always rely on ordinary people to help a friend in need. Firmware is the friend in the "low places" of data centers. It runs in servers, memory subsystems, storage systems, cooling units, communications controllers, power management systems, and other devices. These systems and subsystems rely on firmware to verify the soundness of the hardware, to transfer control to subsequent software, and, in many cases, to operate the hardware directly. Firmware typically has full access to the resources of a system, such as volatile and non-volatile memory, processors, coprocessors, voltage regulators and fans. What, then, if firmware were to become irreparably modified, whether by mistake or malice?

2 Firmware Ownership in the Open Compute Project

The Open Compute Project (OCP), defines itself as "a collaborative community focused on redesigning hardware technology to efficiently support the growing demands on compute infrastructure."[1] Two OCP projects, "Security"[2] and "Open System Firmware"[3] incubation projects, have identified security as critical to the resilience of the compute infrastructure. As these projects attempt to make the firmware in OCP devices as open and secure as possible, the concept of ownership repeatedly arises. Ownership establishes the authority to initialize and update firmware in a device.

The goal of this paper is to provide tutorial information about firmware ownership as requested by members of multiple OCP projects. Firmware ownership affects the overall security of OCP devices, which, in turn, affects the security of the compute infrastructure in which the devices are deployed. This paper describes secure and efficient methods of establishing, representing, and transferring ownership. It provides detailed examples of ownership transfers throughout the lifecycle of a device. Finally, it relates these examples to OCP's tenets of efficiency, scalability, openness, and impact.

The information herein is based on the authors' decades of work in designing and implementing ownership in a broad range of security devices, from smart card chips to servers.

3 The parties involved

Consider a simple example of a data center that procures and deploys a thousand identical new devices. The devices arrive with firmware that is functional, but outdated. After first installing the devices, the data center staff must update the firmware, and continue to update it, as new versions of the firmware are released, throughout the life of the device. When the device is ultimately taken out of service, it is sent to a reclamation center, where it is stripped of useful parts, and the remaining parts are scrapped. In this simple example, there are only three parties involved: the initial manufacturer, the data center operations staff, and the reclamation company.

A more realistic example involves more parties, each with their own responsibilities and concerns, such as

- suppliers who furnish component parts to the device vendor
- original design manufacturers (ODMs) who assemble the components before the devices are rebranded by the device vendor
- independent vendors who write the firmware
- testing facilities that test the device and its firmware
- third party evaluation agencies who review the security of the firmware
- the data center's staff who configures the devices (e.g., is power saving mode always enabled?)
- the chief information security officer's staff, who determine and audit the security configuration of the devices (e.g., is encryption always enabled in storage media?)
- the data center customers (e.g. is my application key adequately protected in this hardware security module?)

Each of the parties has a vested interest in the configuration and security of the device firmware.

4 The rights and privileges of the firmware owner

In this paper, we use the term “owner” (others use use “administrator”, “officer”, or “authority”). The firmware owner is not necessarily the one who purchased a device, and may not even have physical possession of it. Nor does the firmware owner necessarily hold the intellectual property rights to it. Instead, the owner controls what firmware is allowed to run on a device. Consequently, the owner controls its security. Ron Minnich, a software engineer at Google and co-chair of the Open Compute Project on Open System Firmware, observes, “If you don’t own your firmware, your firmware owns you”.

The owner establishes ownership of a device by installing a cryptographic signature verification key or certificate into the device, along with the first version of firmware. In the simplest case, the device uses that key to verify the authenticity and integrity of the firmware (see Figure 1). For example, before selling its devices, a vendor, in its role as the owner, installs its own signature verification key and firmware into them. The devices are deployed, and later, when the firmware is outdated, a new version, digitally signed by the vendor, is presented to each device. The device attempts to verify the digital signature, and, if the verification succeeds, installs the new version on the device. If the verification fails, for example, if the wrong party signed it or if the firmware was modified after it was signed, then the device rejects it.

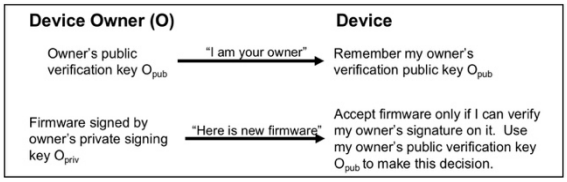

Figure 1

That example, while straightforward, is not common. Instead, it is more common for the owner to sign other keys (not the firmware itself), and those other keys are then used to verify the firmware. (See Figure 2) This key hierarchy allows the device owner to delegate the authority to other parties, who sign firmware using *their* keys.

Such a hierarchy is in use today in servers which implement secure boot. In those systems, the firmware owner is typically the system or platform manufacturer. Systems are shipped with default firmware and a key hierarchy pre-installed. This initial configuration also controls whether to allow another entity to take over ownership, either through physical presence or authenticated remote configuration services.

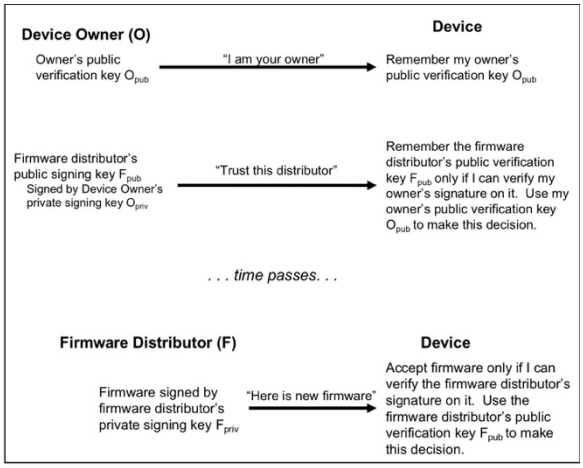

Figure 2

5 The problem to be solved

Attackers attempt to take control over devices in order to install or execute malware. As a device owner, an attacker can replace or augment legitimate firmware with malware, then use it to control the device or to install additional malware. Even devices which support secure boot are programmed to accept firmware from their owners. The problem, then, is how to prevent an attacker from establishing ownership.

Although there are many issues around establishing ownership, two key ones that we address in this paper are

1. How can a secure device be initialized with its very first credentials?

2. Once initialized, how can ownership of the secure device be transferred to another party?

6 Initialization – The Origin of the Device’s Universe

How can a secure device be initialized with its very first credentials? There are at least three common ways: 1) imprinting, 2) installing temporary transport keys and initializing later, and 3) establishing permanent keys during manufacture.

6.1 Imprinting in the Field

Imprinting allows the first initializer of a device to establish the device's identity and membership in an organization. Device: "Hello server, I am device ABC, my public key is $ABCK_{pub}$ and I want to become part of your system." Server: "Hello device ABC, you are hereby part of the XYZ server owned and managed by XYZCo." The device may create its own initial keys, or they can be generated externally and injected. Those keys must be certified, lest the device be indistinguishable from other devices that are outside of the organization. Further attempts to imprint the device are either allowed (after wiping all secrets) or forbidden (by blowing a fuse or setting an unmodifiable bit).

It is important that the chain of custody, and the certifying and imprinting operations be physically and logically secured. If not, then attackers can trick the certifying operation to certify a software clone or to certify keys controlled by the attackers. Device: "Hello certifying server, I am device ABC (but I'm really a software clone pretending to be real hardware)". Or, Device: "Hello certifying server, I am device ABC (but I'm really hardware with a hacker-controlled key inside)". The certifier may not be able to tell the difference, when it's the very first step in establishing ownership.[4, 5]

In an insecure supply chain, attackers can use stolen, but authentic hardware that they have imprinted to impersonate legitimate hardware. "Hello Bank of New Currency, I look just like all your other devices. You can trust me because I'm real hardware." The device, of course, is indeed real hardware, and it may report that it's running legitimate firmware, even though it's running an attacker's firmware.[6]

6.2 Temporary Transport Keys

Similar problems exist in the Internet of Things (IoT), where thousands of low cost devices are manufactured, initialized with applications, and personalized with the user's information. There, devices are initialized with a secret transport key that is common across a large batch of devices, and is known only to the manufacturer and the next organization to process the device. When devices have limited processing and memory resources, initialization using a secret key is fast and easy. Using this technique requires physical and logical protection of the secret transport key. Some organizations use public / private key pairs instead of shared secret keys.

When the number of organizations is relatively small and their identities are known in advance, such a scheme is feasible. However, temporary transport keys are not always an option, because at the time of manufacture, the identity of the "next" organization is not known, nor is it known what quantity of devices the unknown organization will order.

6.3 Permanent Keys at Manufacture

Manufacturers can establish ownership during a once-in-a-lifetime (of the device) initialization step performed in a secure manufacturing facility. There, the device generates its first key pair. Then, the manufacturer digitally signs and installs a certificate containing the device's unique id and public key, and information about the manufacturer. Trusted Platform Modules are initialized this way.

One such device is the IBM 4767-002 PCIe Cryptographic Coprocessor,[7] a device that has been evaluated at level 4 under the Federal Information Processing Standard (FIPS)140-2[8]. Its design allows one general-purpose device to be programmed and updated by multiple authorities for widely different security applications. The IBM 4767-002 initialization step takes place after the module has already been encapsulated in its tamper-responding enclosure.

When security is the utmost concern, initializing a device with its permanent and secure identity at the time of manufacture is preferred. In a device with tamper protection, the device can assert its identity, its configuration, its owner(s), and its manufacturer, and it can protect itself, even before it leaves the manufacturing facility.[9]

6.4 Hybrid techniques

The techniques listed above can be combined to meet the needs of the manufacturer, the stage of manufacturing and device's next destination. For example, a batch of devices might use a shared secret transport key to get them from a chip fabricator to an adapter vendor, but later, the adapter vendor injects unique key pairs and matching certificates. In another example, a vendor might install permanent keys at manufacturing, but those keys control only one portion of the firmware. The remainder of the firmware is controlled by the customer, who imprints secondary keys in the field.

7 Transferring Ownership

As noted in section 3 above, any one or a combination of parties may furnish the firmware or keys for a device. How then, can a device identify all the parties that may control it throughout its lifecycle?

7.1 Functional Requirements

In our work, we have identified common functionality required to implement the concept of firmware ownership and ownership transitions. This functionality is required by a range of devices, from smallest to largest, such as smart card and passport chips, Trusted Platform Modules, hardware security modules (HSMs), baseboard management controllers, service processors, large cloud servers and their subsystems.

In this section, we describe a subsystem that meets these requirements using a simple state machine (in hardware or firmware), persistent registers or memory locations, and digital signature verification operations.[10] Other implementations are possible.

7.1.1 The Minimum Requirements

In order for a device to validate the origin and integrity of firmware it is expected to run, it must

a) remember and write protect the firmware owner's public key (or a list of keys), and

b) verify the digital signature of firmware that was signed (elsewhere) using the owner's private key.

The owner's public key is typically stored in an X.509 certificate. Note that we assume that a public key algorithm is used, because, in addition to signature verification, it provides non-repudiation of operations. The public key must persist through power cycles. Read protection of the public key is not required, but write protection is. It must be protected from malicious or inadvertent changes that would allow the wrong party to become the owner.

7.1.2 Requirements that support transfer of ownership

The minimum requirements in 7.1.1 above assume that a device, once owned, remains in the control of that owner. However, in our experience, such a scenario is unlikely. Instead, as a device progresses through its lifecycle, its ownership and the keys inside it will change. Consider, for example, a device as it transitions from a fully open manufacturing test floor, to a final test stage, and then to a courier for transport to its final customer installation. The first keys used on the open test floor have little to no security requirements. Anyone can update them and digitally sign tests to exercise the device (assuming signature verification is even enabled at that time). At the final test stage, operators install production keys to test that firmware signature verification works prior to shipment. Finally, at the customer's data center, the customer may replace the production keys with ones with stricter controls and pedigree.

In our experience, devices large and small, simple and complex, go through these firmware ownership transitions. It is important, then, to generalize the concepts of firmware ownership and secure transition of ownership in an easy-to-implement representation that uses very little memory.

7.1.3 Representing ownership in persistent memory

Our system requires a small number of ownership-representing registers, and certain additional attributes, to describe the current state control, short-term past history, and a designated successor:

- **Current owner**, storing an owner (public key) certificate, certificate hash, issuer + serial number, or an unambiguous representation of the currently authorized owner. We generally assume that the corresponding private key is not present within our system. It would be stored in a secure signing device; we only process authenticated messages.
- **Previous owner**, storing the authentication information about the last accepted owner.
- **Designated successor**, if present, contains a certificate, hash, or other identification of the next targeted user. The device will reject transfer of control to any other designated entity.
- **Reversibility**: a Boolean attribute, which represents the capability of the previous owner to revert the device back to his control.

7.1.4 Commands

At the time of an ownership transfer, the device verifies the digital signature on a command requesting an ownership change. Incoming state-changing commands are authenticated using the current owner's public key. In typical certificate management scenarios, an ownership change command can take the form of a "transition certificate" designating the new owner as an authorized one, digitally signed by the current owner.

7.1.5 State machine

The entire operation of changing ownership can be represented by a simple state transition machine, which uses two inputs to determine a new state: the current state of ownership as described in section 7.1.3, and a command as described in section 7.1.4 determine the new state of ownership. Worked examples of an ownership state machine and common transitions are in section 7.1.7 below.

7.1.6 Extensions

Note that our directly used registers represent only one previous owner. Maintaining a full history of past owners may be added. A one-way chained certificate list could be maintained parallel to our current-ownership information, and validated against it. Since such digitally signed certificate chains may be validated offline, we may let any other entity aggregate past history, while our system only maintains the set of three active certificates.

The system trivially generalizes to multiply-controlled environments, such as those requiring multiple signatures to authenticate critical commands. Such policies could be represented as additional attributes, such as describing maximum and required number of signatures.

7.1.7 Worked Examples

Figure 3 shows examples of transfers of control, from the fictional originator "Republic of Utopia" (UTO) to its counterpart in the similarly fictional "People's Republic of Utopia" (PRU)*. In our example, UTO1 and UTO2 represent different signing keys controlled by the originator, PRU is the signing key of the recipient, and SVC is a service key shared and known to both of the mutually suspicious participants.

1. Control of ownership from UTO to PRU may pass through the possible states, depending on the policy used: Controlled entirely by the "current" owner, either without relevant previous owner (a clean slate), or with a current owner, rolled over from a previous certificate within the same organization. (states [C1] and [C2])

2. Handed over to a service key, allowing reverting to the previous owner. [C3]

3. Handed over to the "successor" (PRU) with no revocation of this handover, while PRU has not yet acknowledged handover. [S1]

   This state transition also represents a fully online ownership transfer, if the identity of the intended recipient is established in an interactive protocol.

4. Handed over to the successor, with the current owner allowed to revert the handover. Note that this state is effectively controlled by both certificates, with two states. [C4 and S5]

5. The designated successor, PRU, accepts ownership, rolling over its own certificate. [S2]

6. The successor erases the previous certificate, advancing the device into a new "clean" slate. [S4]. Note that the global certificate history remains, and only the current/previous pair is updated.

7. Control is passed to the successor indirectly, transferring ownership to a service key, indicating some indirect information about the designated successor PRU. [U1]

8. Indirect transfer, through registering a service key, without revocation capability. [U2]

   Note that anyone, including the previous owner, may take ownership from this state. Therefore, the lack of revocation is only a policy restriction.

9. Indirect transfer from service key to successor ownership, with an offline-prepared transition certificate (from service key to successor). [S3]

* Standard entities used as examples in literature related to electronic documents[11]

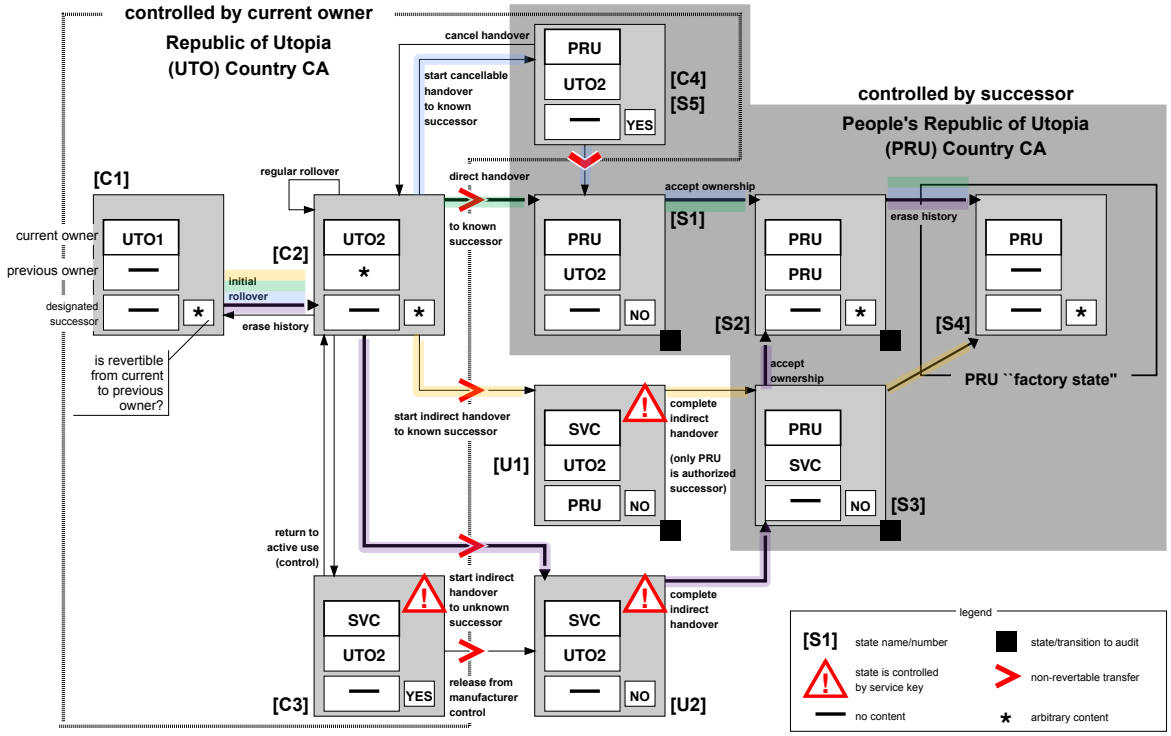

Figure 3

7.1.8 Common ownership flows

Depending on ownership-transfer policies, the system will pass through different chains of states during the transition. Highlighted arrows show typical system transitions for the most frequent scenarios. We mark irrevocable control specially (see the legend) since these state changes correspond to externally newsworthy, auditable events. We similarly mark states controlled by service keys, where ownership is effectively shared.

1. Regular rollover with direct coordination, replacing the control certificate directly [C1, C2, S1, S2, S4] (see Figure 3 green arrows).

2. Regular rollover, with revocation capability [C1, C2, C4/S5, S1, S2, S4] (see Figure 3 blue arrows). While in the [C4/S5] state, the module is effectively controlled by both predecessor and successor. When the initial transfer is rolled back by the originator, ownership returns from [C4/S5] to one of the originator-owned states [C2].

3. Indirect rollover to known successor, without rollback capability [C1, C2, U1, S3, S4] (see Figure 3 yellow arrows). This mode allows offline construction of service-to-PRU transition certificates, while still preventing unauthorized entities taking control of the service-key state [U1].

4. Indirect rollover to any target, through a service key [C1, C2, U2, S3, S2, S4] (see Figure 3 violet arrows). This chain of states may optionally include [C3], if the originator wishes to explicitly mark transfer of ownership to the effectively shared ownership represented by service keys. While functionally the intermediate state [C3] is not relevant, it allows an auditable handover if the originator intends to show explicit start of handover (as an example, if a manufacturer marks a device in inventory as intended for use by a subsequent user).

   The only difference between [C3] and [U2] is the capability of rollback from [C3], allowing the originator to take back an object to its control, without using the service key.

8 OCP Tenets

The OCP has defined four tenets to follow when designing products intended for the compute infrastructure: efficiency, scalability, openness, and impact.[12]

8.1 Efficiency

While it is not the only way to represent ownership, the storage which retains the persistent state of ownership as described in sections 7.1.3 can occupy less than 6K bytes of storage, even with full X.509 certificates, and accompanying metadata. The command verifier and state machine can be implemented in a small amount of random access memory or in an FPGA, the largest portion of code being the signature verification check.

8.2 Scalability

Despite its simplicity, the concept of ownership as described above can be implemented in a small amount of memory on a low cost embedded processor, as well as on subsystems of large servers. It is small and simple enough to put on each device, or it can be expanded to handle the ownership of a

cluster of multiple devices. It can be extended to include longer audit histories of previous owners or multiple simultaneous owners. Additionally, with signed commands or firmware updates, remote administration is feasible. Depending on the security design of the device, the use of digitally signed firmware updates can eliminate the need for physical presence in all but the most severe failures of devices and loss or destruction of keys.

8.3 Openness

The objective of this tutorial paper is to make the terminology and techniques known to others. Establishing initial ownership depends heavily on device features, the configuration of manufacturing facilities, the supply chain, and the trust level of customers, so it is difficult to recommend any one technique as a standard. However, requiring device manufacturers to document their device initialization procedures is recommended. Some of the technology described herein is patented[10].

8.4 Impact

Device manufacturers can use the techniques described in this paper to improve the security of their supply chains. Establishing ownership as early as possible in manufacturing, while write-protecting the owner's public key, enable the device to require digitally signed firmware updates as soon as possible in the supply chain. Devices with owners can also manage their own firmware update processes.

9 Summary

Firmware ownership is important to device and compute infrastructure security because it determines who is allowed to update the firmware on a device. This paper introduces techniques for establishing initial ownership. It illustrates several examples and models of transferring ownership during a device's lifecycle. Finally, it relates the OCP's four tenets to the concepts in this paper.

References

1. The Open Compute Project, https://www.opencompute.org/about, retrieved on 10/1/2018.
2. OCP Security Project, https://www.opencompute.org/projects/security, retrieved on 10/1/2018.
3. OCP Open System Firmware Project, https://www.opencompute.org/projects/open-system-firmware, retrieved on 10/1/2018.
4. Alex Sotirov, Analyzing the MD5 collision in Flame, https://trailofbits.files.wordpress.com/2012/06/flame-md5.pdf, retrieved on 11/9/2018
5. Richard Chirgwin, How to nab a HTTPS cert for a stranger's website: Step one, shatter those DNS queries, https://www.theregister.co.uk/2018/09/06/certificate_authority_dns_validation/, retrieved on 11/9/2018.
6. Hagai Bar-El, Known Attacks Against Smartcards, https://infosecwriters.com/text_resources/pdf/Known_Attacks_Against_Smartcards.pdf, retrieved on 9/18/2018.
7. IBM 4767 PCIe Cryptographic Coprocessor, https://www-03.ibm.com/security/cryptocards/pciecc2/pdf/4767_PCIe_Data_Sheet.pdf, retrieved on 9/18/2018.
8. Security Requirements for Cryptographic Modules, http://csrc.nist.gov/cryptval/140-2.htm, retrieved on 9/21/2018.

9. Joan Dyer et al., “Building the IBM 4758 Secure Coprocessor”, *IEEE Computer*, vol. 34, no. 10, pp. 57-66.

10. M. Osborne, E. Palmer, and T. Visegrady, “Managing Transfer of Device Ownership.” US Patent 9,967,102 B2, issued May 8, 2018.

11. ICAO document 9303, Machine Readable Travel Documents, Seventh Edition, 2015, part 7: Machine Readable Visas, https://www.icao.int/publications/Documents/9303_p7_cons_en.pdf, retrieved on 9/23/2018.

12. OCP Tenets Explained, https://www.opencompute.org/files/OCP-Tenets-FINAL2-1.pdf, retrieved on 10/1/2018.

