Towards Protected Firmware Verification in Low-power Devices

Yong-Hyuk Moon and Jeong-Nyeo Kim
Hyper-connected Communication Research Laboratory
Electronics and Telecommunication Research Institute (ETRI)
Daejeon, Republic of Korea
email: {yhmoon, jnkim}@etri.re.kr

***Abstract*—It is barely conceivable to ensure the security state of a device without a trusted computing base. However, a hardware security module is not provided in most low-power devices. This paper presents a new design approach, which can securely verify a current state of firmware at a booting time utilizing untrusted components. We discuss a Memory Protection Unit (MPU) enabled memory access control to ensure that memory regions of a bootloader are not accidentally compromised from unintended access. Further extensions of the suggested approach are also addressed for achieving the enhanced security confirmation.**

Keywords-firmware verification; memory protection; device security.

I. INTRODUCTION

Secure booting is a fundamental security technique of computing devices and recently become a mandatory option for protection of computing tasks and resources. However, most Microcontroller Units (MCUs) of low-power devices do not contain a hardware security module functioning as a Trusted Computing Base (TCB). The commodity MCUs may not provide sufficient chip-level protection. It is difficult to validate if a device is correctly programmed as intended. Further, devices are highly vulnerable to a simple piece of exploits since run-time verification of code and data is performed on the uncertain assumption that a verification process may be trustable.

To tackle this limitation, we discuss a feasible design approach, which can confirm a current security state of a device with the existing untrusted components. Our primary contributions can be summarized as two aspects: *i*) we first suggest how firmware verification can be performed by a custom bootloader; and *ii*) we then discuss an MPU-enabled memory protection scheme, which guarantees the reliability of firmware verification by controlling code and data access to the bootloader. In addition, the proposed design approach has been partially implemented and tested as a prototype software modules on devices working with Advanced RISC Machine (ARM) Cortex M3/M4 for checking its validation.

The remainder of this paper is organized as follows. Section II briefly reviews the conventional approaches for maintaining device security. We discuss a new design approach for firmware verification and bootloader protection in Section III. Section IV provides further extensions on the proposed design. Finally, we conclude the paper in Section V.

II. RELATED WORK

Recent lines of research related to device security are reviewed and their issues are discussed in this section.

A. Secure Booting

A built-in Read Only Memory (ROM) is a minimal requirement for designing and implementing secure booting at small-footprint devices. Once some ROMs of MCUs are masked during manufacturing, further modifications are not allowed for bootloader protection [1]. Alternatively, a custom bootloader can be loaded from some blocks of flash memory. However, it is difficult to prevent an accidental erasure or modification of the bootloader and its related configurations and secure materials from unintended access. This directly implies that the genuine of firmware or operating system working at a device cannot be guaranteed.

B. Remote Attestation

To revalidate a programmed firmware at a device, software attestation schemes have been widely proposed [2]. One common assumption is that a remote verifier is trustable and secure communication is established between a prover and a verifier [3]. However, a prover's trustworthiness remains unclear and manipulated checksum functions may not be complicated enough against a guessing attack. Another limitation is that this approach tends to focus on verifying the integrity of working codes only [4]. Moreover, code verification is performed at a pre-defined interval of time in a verifier-driven manner. Therefore, attackers may have more chances to compromise devices.

C. Memory Protection

Sancus [5] is a memory access control scheme based on program counter, so that a new hardware implementation is required as an extension of MCUs. This approach also depends on a specifically modified C compiler and a TCB. Similarly, Smart [6] uses a special hardware-controlled memory for a secure key storage and allows that ROM-resident code only access to the keys. For execution-aware memory protection, TrustLite [7] uses an MPU built in a secure System on a Chip (SoC) and the on-chip memory is required to store MPU configurations. One critical drawback of this scheme is that authenticity and integrity of a secure loader cannot be verified at a booting time.

 ISBN: 978-1-61208-582-1

III. PROTECTED FIRMWARE VERIFICATION

We suggest one feasible design approach to security designers and system programmers for ensuring firmware protection without any hardware modifications.

A. Memory Construction

Figure 1 shows an example of memory layout, which is used in the proposed protected firmware verification. In this approach, we assume that the cryptographic computations, such as key derivation, firmware encryption, key wrapping, and signature creation can be completed prior to loading a custom bootloader and a firmware to a flash memory.

To construct such memory layout, two offline processes, such as *i*) encrypting a firmware and *ii*) signing a firmware are required as depicted in Figure 2. In the first phase, a symmetric key generator creates a Firmware Encryption Key (FEK) and we derive a Confidentiality Root Key (CRK) from a given Production Unique Key (PUK). We then encrypt an original firmware image with the FEK and using the derived CRK, we also wrap the FEK based on the Advanced Encryption Standard (AES) [8] for containing the integrity information of FEK. In the latter, an authenticity key generator creates a key pair and compute a firmware signature based on the Elliptical Curve Digital Signature Algorithm (ECDSA) [9]. Through the above steps, we have the encrypted firmware, AES-wrapped FEK, ECDSA public key, and firmware signature as security materials for firmware protection. Those data are finally allocated to flash memory regions.

B. Bootloader Protection

Immediately after power-on or reset, the booting code performs an initial system configuration by referring to its header. We assume that a Custom Bootloader (CBL) resides on some memory regions of flash and its code and security materials can be protected by setting lock bits at a flash register. However, locking the booting related memory blocks may not be a strong method of ensuring code and data isolation of the CBL. To mitigate this problem, we adopt a MPU-enabled memory access control to prevent unauthorized access to those memory regions during booting. Moreover, this approach can be applied in protecting code and data memory even after the firmware (i.e., kernel) loading. Due to this reason, the CBL then initializes a MPU according to a predefined policy to protect itself and its related data sections, which are colored in grey during booting sequences and firmware verification as illustrated in Figure 3.

When a Central Processing Unit (CPU) tries to execute a code (i.e., instruction pointer) or access read/write a memory region (e.g., stack), an MPU [10] can enforce these accesses to code and data memory with pre-configured settings. For example, the header, keys, signature, and flash registers can be only accessed by instructions defined in the CBL with a read permission. Moreover, addresses of currently fetched instructions by a CPU core are also checked for validating code regions. It is necessary to define what interrupt handlers can perform hardware processing for booting code; an MPU needs to know which addresses of Static Random Access Memory (SRAM) are allocated to the CBL. These considerations can be made as MPU rules.

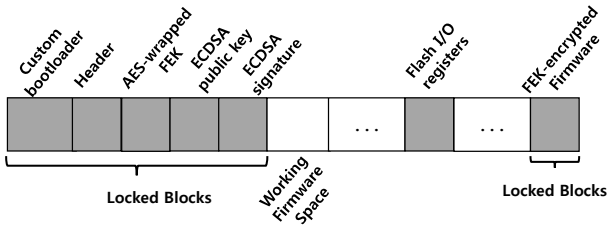

Figure 1. Memory construction for firmware verification

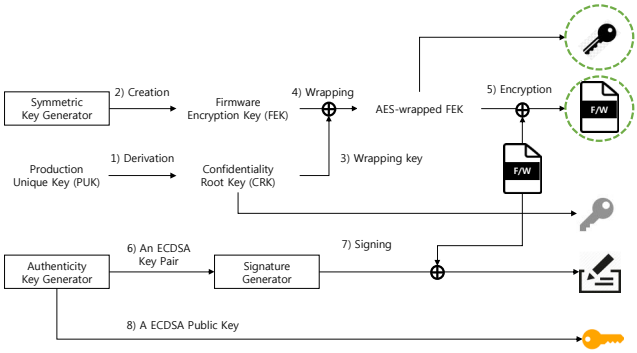

Figure 2. Generation of security materials

C. Firmware Verification

If it is confirmed that a CBL is not compromised and an MPU is activated as intended, a CBL can verify a firmware's security state in terms of confidentiality, authenticity, and integrity. The following phases describe how a CBL verifies a firmware only using a One-Time Programmable (OTP) memory under the monitoring of MPU as shown in Figure 3.

i) The CBL tries to obtain a CRK from an OTP memory. An illegal access to a CRK in the OTP memory violates the MPU rules, so that a memory fault can be detected by an MPU. After that the CBL unwraps a FEK based on the AES cryptographic algorithm with the CRK. If the FEK turns out to be available, the CBL can decrypt the protected firmware. The above process is effective to avert firmware cloning.

ii) The CBL calculates a digest value of firmware, which can be compared to the original one in an OTP memory for checking the integrity of decrypted firmware. Further, the firmware digest and an ECDSA public key are utilized to compute a new signature of the decrypted firmware according to the ECDSA. If the generated signature is equal to the contained one (see an ECDSA signature in Figure 1), the CBL accepts that the decrypted firmware is authentic. As a result, the CBL can copy the decrypted firmware to a particular memory space for a working firmware and delegates its control to the working firmware.

In the aforementioned phases, validation of CRK, FEK, and ECDSA public key can be confirmed by a simple hash comparison using an OTP memory. Moreover, the security state of updated firmware can be verified in the same way as above by adding a newly computed ECDSA signature of a new version of firmware into a differencing data package encoded by the VCDIFF standard [11].

 ISBN: 978-1-61208-582-1

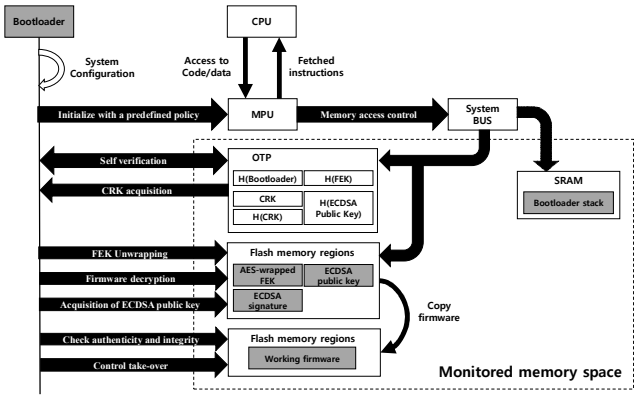

Figure 3. Firmware verification with MPU-enabled bootloader protection

IV. DESIGN EXTENSIONS

This section describes architectural extensions of the proposed protected firmware verification in the following three perspectives.

A. Kernel Level Support

Any privileged task can unexpectedly unlock the memory-mapped registers including flash, MPU, etc. Despite this weakness, some operating systems allow that every task is executed with a privileged mode only. For this reason, it is required that kernel separates user mode tasks from system modules and interrupt service routines (ISR). This required feature can be new to some operating systems but is effective to prevent user mode tasks from accessing privileged instructions. Besides, code and stack regions of each task, interrupt handler, and kernel modules must be monitored by an MPU and memory access violation must be handled as well in an appropriate manner. This MPU-enabled memory protection mechanism can guarantee that, a privileged/user task and an interrupt handler can be restricted from removing or modifying boot related memory regions, even after a firmware is loaded.

B. Secure Memory Loader

Booting codes can be built and activated in a dedicated mask ROM. In this case, we can replace the custom bootloader on flash with special codes, which is called a Secure Memory Loader (SML). One effective way to improve the execution reliability of security-sensitive codes for the protected firmware verification is to reduce the size of the CBL by excluding booting functionalities. If the SML can be precisely defined and limited, more secure and correct invocation of SML and cryptographic computations are within the realm of possibility. Removing or overwriting a SML is beyond the scope of this paper. However, an external verifier would be a better option rather than using an OTP memory for coping with this vulnerable situation.

C. Trustworthy Remote Entity

Custom bootloader's code and data can be attested by a remote verifier to provide an extension option for increased security confirmation if bootstrapping a device must be completed through a trusted server. Besides, a CRK can be received via an end-to-end encrypted network session between a device and a server but this alternative approach would cause more delays than using an OTP memory. After the firmware are loaded, if a memory access violation occurs against the MPU policy, a remote server can exclusively handle such system fault by taking some countermeasures such as remote wipe, network isolation, device recovery, and firmware update.

V. CONCLUSIONS

In this paper, we have suggested a new design approach for protected firmware verification with respect to memory construction, its cryptographic operations, and memory access control. Further extensions as discussed in Section IV will be addressed with respect to implementation and feasibility in our future work.

ACKNOWLEDGMENT

This work was supported by Institute for Information and communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) [2015-0-00508, Development of Operating System Security Core Technology for the Smart Lightweight IoT Devices].

REFERENCES

[1] ATMEL, Application Note, "Atmel AT02333: Safe and Secure Bootloader Implementation for SAM3/4," June, 2013.

[2] Arvind Seshadri et al., "Pioneer: Verifying Code Integrity and Enforcing Untampered Code Execution on Legacy Systems," SOSP'05, pp. 1-16, October 23-26, 2005, United Kingdom.

[3] Y.-H. Moon and Y.-S. Jeon, "A Functional Relationship Based Attestation Scheme for Detecting Compromised Nodes in Large IoT Networks," CUTE'15, vol. 373, pp. 713-721, December 2015.

[4] N. Asokan et al., "SEDA: Scalable Embedded Device Attestation," CCS'15, pp. 964-975, October 12–16, 2015.

[5] J. Noorman et al., "Sancus: Low-cost Trustworthy Extensible Networked Devices with a Zero-software Trusted Computing Base," In USENIX Security Symposium. USENIX, pp. 479-494, 2013.

[6] E. Karim, F. Aurélien, P. Daniele, and T. Gene, "SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust," NDSS'12, February 5-8, USA.

[7] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: A Security Architecture for Tiny Embedded Devices," EuroSys'14, April 13-16, 2014.

[8] J. Schaad and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm," Internet Engineering Task Force (IETF), Network Working Group, RFC 3394, September, 2002.

[9] National Institue of Standards and Technology (NIST), FIPS PUB 186-4, Digital Signature Standard (DSS), July 2013.

[10] ATMEL, Application Note, "Atmel AT02346: Using the MPU on Atmel Cortex-M3 / Cortex-M4 Based Microcontrollers," April, 2013.

[11] D. Korn, J. MacDonald, J. Mogul, and K. Vo, "The VCDIFF Generic Differencing and Compression Data Format," Internet Engineering Task Force (IETF), Network Working Group, RFC 3284, June 2002.

 ISBN: 978-1-61208-582-1