SECURE BOOTLOADER FOR ARM MICROCONTROLLERS

Pavol Prítel
Doctoral Degree Programme (1), FEEC BUT
E-mail: xprite00@stud.feec.vutbr.cz

Supervised by: Radimír Vrba
E-mail: vrbar@feec.vutbr.cz

Abstract: Embedded systems are single purpose devices, where software updates are often omitted or very limited in comparison with personal computer systems. Such systems are typically using microcontrollers with limited memory and computing power. Enabling software updates on an embedded system brings some security issues that need to be taken into account. As soon as there is a possibility of software update, the device is exposed to risk of malicious or not original applications being executed on the device. In this paper a reference implementation of bootloader with security features is described. As a result of using asymmetric cryptography, the memory footprint and start-up time is increased significantly. The contribution of this work is in finding the right algorithm that will offer trade-off between level of security and used resources of targeted microcontroller.

Keywords: elliptic curve cryptography, cipher, bootloader, microcontroller, embedded

1 INTRODUCTION

Security took place in today's personal computer world. Confidential data is stored on everyone's computer and is protected by passwords, biometry, encryption and other security mechanisms. Security is a standard that end user expects from the device. Different situation is in the domain of embedded systems, where lack of security is significant. Embedded systems are often single purpose devices executing specific application that does not need an update for a longer period of time during the product life cycle. Firmware is rarely without any bugs, even worse, these bugs are often found after the device is already deployed in the field. Such a bug can be fixed by factory recall of all devices. Another solution, economically more acceptable is the integration of bootloader into system. By separating application itself from boot process, the system is enriched with possibility of software updates. On the other side, the system is exposed to malicious and non original applications, which can be intentionally loaded by an attacker. Therefore, security is a key parameter of any bootloader. Bootloader is a piece of software responsible for loading applications. Bootloader typically establishes a communication with host expecting an application image to be sent and then hands over the execution to the final application. The most valuable feature of bootloader is the capability of downloading an application using communication protocol that fits the specific application. In other words, the bootloader logic is separated from the communication protocol used to download the application image. Any of well-known protocols can be used depending on the field of operation. Industrial embedded systems will most likely use protocols specific for industrial area such as CAN, Ethernet. Simple standalone devices will most likely use USB, UART, SPI, I^{2}C or any other interface. Bootloader described in this paper is targeting ARM Cortex-M microcontrollers. As a result of possible attacks analysis, several security features were implemented. The key security feature of the bootloader is authentication of downloaded application using asymmetric cryptography.

The chapter 2 describes the basic boot process of microcontroller and security recommendations for bootloader. Chapter 3 introduces the key algorithm in the bootloader security chain, chapter 4 shows the result of the digital signature algorithm benchmark.

2 BACKGROUND

After the reset of Cortex-M microcontroller, the interrupt vector table is located at address 0x00000000. The first two entries in the interrupt vector table contain initial value of stack pointer and reset vector. After the reset, processor sets up MSP (Main Stack Pointer) and PC (Program Counter) registers with these values and starts executing the first instruction on the address pointed by PC register. This boot sequence is shown in figure 1 and is common for all ARM Cortex-M processors [2].

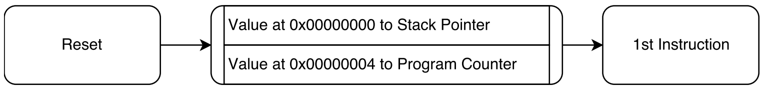
Reset
Value at 0x00000000 to Stack Pointer
Value at 0x00000004 to Program Counter
1st Instruction

Figure 1: Boot sequence on reset

As the after reset location of interrupt vector table is at address 0x00000000, the bootloader must have IVT at this location. This ensures the bootloader is the first application executed by the micro-controller. Reset boot sequence is shown in figure 1.

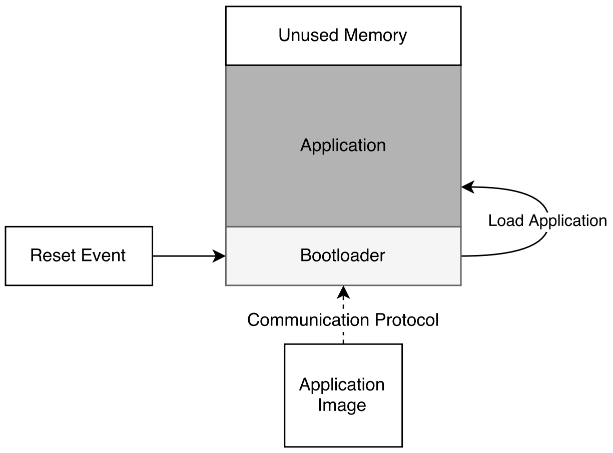

Figure 2: Bootloader functionality in general

From bootloader point of view the application is just stream of bytes that needs to be loaded at the right memory address. The information of hand over address is missing in the application image itself. ARM microcontrollers can relocate the IVT location at run-time using VTOR register. Typically, applications have their own interrupt vector table stored in RAM. The location of the table needs to be known to the bootloader, therefore the table contains entry point (Reset Handler) of the final application. Passing corrupted application IVT location is a security breach that can lead to undefined behavior of boot process if not detected. The simplest solution is to require the applications to be linked with fixed IVT location. This prevents from executing code that was not authenticated. Another solution is that the host sends the IVT location for each application separately. In this case, the IVT location needs to be included in digital signature so the bootloader can fully authenticate the application.

2.1 Security Recommendations

2.1.1 Debug Interface

In order to protect the data stored in the processor, debug access should be always disabled once the device is operational in the field. ARM Cortex-M microcontrollers use JTAG or SWD debug interface that can be disabled. By disabling debug interface permanently, an attacker loses the ability of modifying the code stored in internal memory.

2.1.2 Memory Protection

The application image is serialized and sent to the bootloader using communication protocol. The bootloader has full read/write access to all on-chip memory. The host sending an application to bootloader may request a write to memory which is used by bootloader itself. Allowing this write command could cause the bootloader corruption, which is a security issue. Before any data is written, the memory protection checks whether the address range is accessible by external write commands. Commands addressing protected memory must be rejected. For security reasons, the memory access to memory mapped peripheral registers and bootloader memory space is protected. As a result, bootloader only accepts memory write commands to memory reserved for application. Logic of memory protection is shown in figure 3. An invalid application image may use a memory that should not be used. For example memory in which the bootloader is stored. Any access to memory mapped peripheral should also be restricted. Bootloader must perform a memory range check before any write operation is executed. All memory write operations to restricted memory space must be rejected.

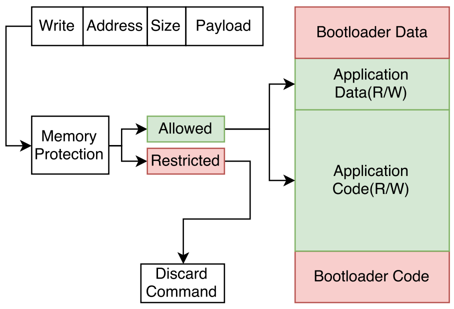

Figure 3: Memory protection in write command

2.1.3 Public Key Table

Asymmetric encryption algorithm is used for authentication of application image. Bootloader should only use public keys that are trusted. This can be achieved by storing the keys in memory that cannot be overwritten by an attacker [3]. Reference implementation uses a key table. The key matching is done before any authentication of application is performed. In case of key mismatch, the authentication process is interrupted and evaluated as unsuccessful.

3 DIGITAL SIGNATURE ALGORITHM

Asymmetric cryptosystems are widely used for digital signatures. The most popular algorithm is RSA, which relies on the hardness of factoring of large integers. RSA keys are prime numbers with recommended size at least 2048 bits [4]. The keys used by RSA are large in comparison with keys used by cryptography algorithms based on elliptic curves, which is the crucial parameter for embedded systems with limited memory resources. ECDSA significantly reduces the key size, while the same level of security is achieved. Figure 4 shows the authentication chain, where the application provider owns secret private key used to sign an application image. Application and generated signature are delivered over the communication channel to the target microcontroller, where bootloader is in charge of authenticating and executing the application. Bootloader owns well-known public key that is used to authenticate the application image delivered by application provider. Signature generation and verification are time critical operations, where only verification is executed on the resources limited microcontroller. Many large integer multiplication operations are involved in elliptic curve cryptography algorithms and these operations are not supported as a native instructions on microcontrollers. In the bootloader implementation, the microECC open source library was used for ECDSA.[1].

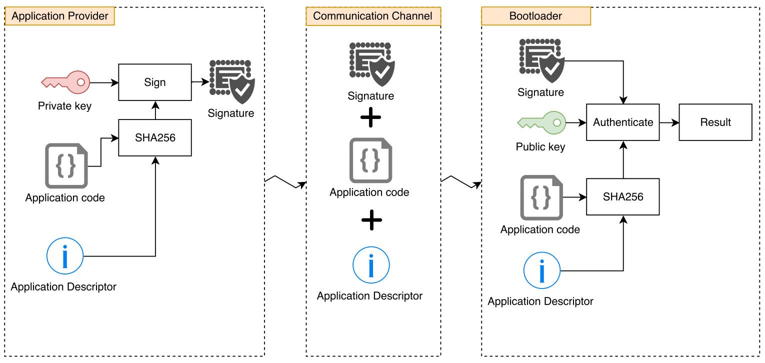

Figure 4: Application authentication using digital signature

4 ECDSA BENCHMARK

There are basically two time critical operations in authentication process. Majority of Cortex-M microcontrollers are not equipped with hardware cryptography accelerators. As a result, even simple hashing function like SHA256 can take a significant processor time, especially when hashing a large payload. The absence of hardware cryptography accelerator block results in increased boot time of final application. Another time critical operation is signature verification. Microcontrollers with hardware accelerator for elliptic curve cryptography are rare. As a result, most of the embedded applications are using the software implementation. Very large integer multiplication was found as the main bottleneck of elliptic curve cryptography, as long as these operations cannot be computed by native instructions. The performance evaluation was executed on three different microcontrollers.

Testing the ECDSA library built with different level of optimization show the advantages of using highly optimized assembly language code. Pure C implementation of the VLI multiplication is up to 4 times slower than implementation with optimized assembly code. Signature verification is the most time consuming operation in the authentication chain. Therefore, the benchmark is focused on this procedure. Table 1- 3 shows the result of benchmark, where the same algorithm was built with different level of optimization. All processors are running at their maximum frequency, which is a typical use case.

Table 1: Cortex-M0 24 MHz Verification Time

Optimization Level	Signature Verification Time [ms]
C Language - None	2466
C Language - Low	2144
C Language - Medium	1669
C Language - High	1219
Assembly Language	733

Table 2: Cortex-M3 48 MHz Verification Time

Optimization Level	Signature Verification Time [ms]
C Language - None	1588
C Language - Low	1430
C Language - Medium	1014
C Language - High	638
Assembly Language	489

Table 3: Cortex-M4 144 MHz Verification Time

Optimization Level	Signature Verification Time [ms]
C Language - None	298
C Language - Low	273
C Language - Medium	187
C Language - High	108
Assembly Language	92

5 CONCLUSION

Reference bootloader for Cortex-M microcontrollers was developed considering several security risks. The key security feature is ECDSA for authentication of loaded applications. Results from benchmark show that ECDSA is suitable for targeted processors in context of bootloader, where the time-consuming authentication is performed only once during boot time. The open source library microECC offers highly optimized assembly language implementation of the most time critical operations used by ECDSA. As a result, processors without hardware cryptography accelerators can use this algorithm with a reasonable processor time. Bootloader prevents from malicious software to be executed on target processor, but the application image data transfer between host and bootloader is not encrypted. With additional security level, the boot time would be increased as well. Encryption requires a mechanism of generating shared secret key such as Diffie-Hellman. Therefore implementation of this feature is being prepared as a future work.

ACKNOWLEDGEMENT

The author would like to thank Ministry of Industry and Trade of the Czech Republic in project FV10562 SYMONPRO Systems for Process Monitoring using modern tools for optimization.

REFERENCES

[1] K. MacKay, ECDH and ECDSA for 8-bit, 32-bit, and 64-bit processors, Available online: https://github.com/kmackay/micro-ecc

[2] J. Yiu, The Definitive Guide to ARM R Cortex-M3 and Cortex-M4 Processors, ISBN:9780124080829

[3] M. Hunter, Using the Kinetis Security and Flash Protection Features, http://www.nxp.com/assets/documents/data/en/application-notes/AN4507.pdf

[4] E. Barker and Q. Dang, Recommendation for Key Management, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf