Security Software System Design and Implementation for Microcontrollers Based on TrustZone

Liang Liu, Jun Ma, Cuiping Zhang, Ting Chong, Haifeng Zhang and Yuanyi Dong

ABSTRACT

With the rapid growth of microcontroller applications, especially in the IoT field, more and more devices have the requirements of interconnection, the most important thing which should be taken into account is the security. The TrustZone feature developed by ARM based on ARMv8-M architecture provides the hardware basis for security among devices. Based on this, software is needed for building a secure platform to satisfy the application security requirements. Device secure boot scheme can resist software malicious modification, firmware update scheme can apply security patches in time, secure storage scheme can manage the application confidential assets and can resist malicious modification, secure crypto scheme is used to ensure the secure data transmission. This paper gives some software security features when design and implement secure system microcontroller software.

KEYWORDS
LoT, ARMv8-M, TrustZone, Microcontroller, Security, TEE.

INTRODUCTION

The rapid development of IoT applications brings great convenience to people's lives, but it also brings many security risks. Therefore, the security of device operation and the data security provided by the device are particularly important. Private data is stolen or modified, the transmitted data is illegally intercepted, and the running software was maliciously tampered, such attacks are prominent problems for IoT devices. How to build a secure and reliable system on a microcontroller device with limited cost resources is an urgent problem to be solved. This requires a close combination of hardware and software, the hardware provides the underlying security technology support, on this basis, the software implements the secure system. There are already some application schemes, such as the combination use of a master chip and a secure chip to implement a security system, but it has large chip area, high power consumption, low performance and high cost features. Based on the ARMv8-M TrustZone technology, this paper gives the basic security elements on how to build a
secure system on a single, high performance and low cost chip, and then gives complete software system design architecture.

SECURE BOOT

As required by the PSA (Platform Security Architecture) specification introduced by ARM, a secure system must have a secure boot feature. The boot process of the device can usually be divided into multiple boot stages, corresponding to multiple software modules.

The boot process of each stage uses a standard algorithm to verify the software module of the latter stage, ensuring that the software of the next stage has not been maliciously modified, ensuring the integrity of the software module, and ensuring that the device is booted by authorized software. The whole process can resist the loading and running of unauthorized software until the application is launched. The entire boot verification process will form a trusted chain, as shown in Figure 1.

Figure 1. Boot verification link.

Usually the signing process is started after compiling each software module. The RSA private key is used for signing, there are several steps as follows:

(1) Calculate the hash value of the image to be signed.
(2) Encrypt the hash value using the RSA private key to generate a signature.
(3) Attach the signature and the RSA public key corresponding to the RSA private key to the tail of the original image.

When the signed software image is in the middle of the boot chain, in addition to the public key used for the verification of this image by the upper level, the image tail also includes the hash value of the public key owned by next level image. The Hash value is used to verify the validity of the public key attached to the tail of the next level image, and it participates in the calculation of the signature, so the integrity of the hash value is guaranteed by the result of the verification of the upper level image, as shown in Figure 2.

Figure 2. Image layout after signing.
The verification is the reverse process of the signing. The verification uses public key coming from the tail of the image to be verified. The public key is visible to the outside, therefore, the validity of the public key needs to be guaranteed, the verification process is as follows:

1. Calculate the hash value of the public key attached at the image tail to be verified.
2. Compare the calculated public key hash value with the public key hash value saved in current image tail.
3. If the public key is valid, use the public key to decrypt the signature.
4. Calculate the hash value of the image to be verified.
5. Compare the hash value of image with the content after decryption.
6. If the hash values are consistent, the image is valid.

Usually in a complete chip design, there is ROM space to store immutable boot code. This code is the first part executed after the device is powered on, it initiates the verification of the next level software modules stored on flash. The hash value of the root public key used for verification is usually stored in an OTP storage medium during the factory provision, and this hash is used to verify the validity of the public key attached to the tail of the image to be verified. The immutable boot code along with the root public key hash is the trusted root in the secure boot process.

In the microcontroller application that uses the ARMv8-M architecture, the core is in a secure state at the beginning of system reset, and all verification processes are completed in a secure environment to ensure the validity of the secure boot verification.

SECURE SOFTWARE UPDATE

Real time software update, especially for online real time software update, is important for IoT devices. When the factory provision is completed, the patch that can be used to put security holes in real time is important for the device normal operation, and it is also an effective means to resist malware attacks. Online software update is also an essential feature in the field of microcontroller applications. During the software update process, there will be occasions such as power loss, software update errors, etc. More should be considered on how to ensure the update integrity.

A reliable method is to divide the Flash memory into the current boot area and update software storage area. The update client running on the device can download the new software from the server to the local update software storage area and notify the device to reboot. During the reboot process, the bootloader will detect whether there is the software that can be updated locally, if yes, then performs software update in a secure environment, as shown in Figure 3.
The update process is as follows:

1. The agent on the non-secure side communicates with the remote server, and downloads the software to be updated to the local storage area.
2. Security domain checks the software metadata information.
3. Save new software to the specified storage area.
4. Notify the device to reboot.
5. The bootloader performs the security verification of the software to be updated during reboot process.
6. After the verification is passed, the bootloader performs software update exchange.
7. Start new software from the start region after the exchange is complete.

During the update process, the contents of the update region and the contents of the start region are exchanged according to the specified block size, and the scratch storage area is used for the exchange, as shown in Figure 4.

The exchange process is as follows:

1. Copy the new image block to scratch area.
2. Copy the old image block to the new image block.
3. Copy the scratch block to the old image block.

Each time a block is exchanged, the current exchange information, such as the index of the exchange block, the current exchange step, and other important contents are recorded in the reserved memory area. If there is a power loss scenario, during next boot, the bootloader gets the information recorded during the previous update process and continues the update process, and this ensures the integrity of the software update.
After the update is complete and the start region is booted, the software should confirm the update by writing a special flag to information recorded area. If it is not confirmed, the previous update process will be rolled back after the next reboot. This is the reverse process of the previous update process, and it returns to the state before the update.

SECURE STORAGE

Secure storage is also an important part of IoT applications. Some device sensitive information, such as communication keys, device authentication keys, passwords, etc., which cannot be stored in a non-secure area in plain text, nor can be modified by non-secure software. These data are often characterized by privacy, integrity, and reliability. Based on the basic hardware security isolation, these data can be stored in a secure storage area in secure world, as shown in Figure 5.

![Secure storage](image)

Figure 5. Secure storage.

Software in the non-secure world cannot access or modify these contents. When updating the secure storage content, the security world needs to operate in an atomic manner. In the process of updating operations, if a special circumstance such as power loss occurs, it is necessary to ensure that the stored content is safe and reliable.

Secure storage involves the following aspects:

1. Unified asset access management.
2. Configure access policy based on user ID and asset ID, such as create, read, write, delete.
3. Compile time built in policy database.
4. Encrypted storage based on hardware unique key.
5. Unified management of file systems, atomic operations for each asset.

In the file system, the flash space can be divided into a series of logical blocks, and the information of the logical blocks and the information of the assets are recorded by means of metadata. The physical location of the logical block, the start position of the data in the logical block, and the remaining usable size in the logical block are recorded in the block metadata. The asset metadata records the logical block where the asset is located, the location of the asset in the logical block, the current size and
maximum size of the asset, and the asset ID, so the asset can be found and read/wrote according to the asset ID. The file system structure is shown in Figure 6.

In the design, two special blocks can be used as the scratch blocks when the block information and file information are updated to ensure the atomic operation of the asset and the integrity of the asset after the power loss during operation. When updating an asset, the updated block information and asset information are first written into the corresponding scratch blocks, if there is a power loss, the original information will not be destroyed, as shown in Figure 7.

After all the information is updated, the scratch block metadata header is updated, from this point, the data in the scratch block becomes the final data, and the old metadata block and the asset data block are erased, that is to say the asset data is updated.

The assets can also be encrypted and stored using a key derived from hardware key to prevent data migration between different devices.
SECURE CRYPTOGRAPHY

In a secure system, there are many scenarios that use encryption and decryption, such as local data reporting after encryption, local data storage after encryption, and secure connection with remote servers. Data encryption is also an effective way to achieve device security management and to prevent data leakage.

The encryption and decryption process involves the key management. In a securely isolated system, some important keys can be injected into the security world. When the non-secure world has encryption and decryption requirements, the security side can be requested to perform the corresponding encryption and decryption process. The secure side can further use the hardware cryptography module to speed up the completion of the request and return the result of the encryption and decryption to the non-secure side, as shown in Figure 8.

![Figure 8. Secure crypto calls.](image)

The keys used for encryption and decryption throughout the encryption and decryption process are always in the security world and will not be obtained or even tampered by the non-secure world. What and how many cryptography algorithms should be implemented in secure world dependents on the application requirements.

SYSTEM ARCHITECTURE

Based on the hardware security isolation and the above basic security software elements, a secure microcontroller application system can be built by adding other security services such as device authentication if necessary. User program such as real time operating system can run in non-secure side; the security side runs the bootloader and the secure software that controls and configures the global system. The secure world provides the necessary secure services for the non-secure world and provides scheduling and control for these services operation, the system architecture is as shown in Figure 9.
Each secure service is a logically independent unit of operation that can be organized and runs in a thread. The secure service is in block state if there is no request, and when there is a non-secure request, the trusted core completes the scheduling of the thread and responds to the secure request from the non-secure side in time. The number of secure services can be appropriately increased or decreased depending on the application. The Trusted core initializes each secure service during system initialization, prepares an environment for non-secure calls, and performs security related configurations such as memory and interrupts.

The secure library interface is generated when the security software is built and will be linked to non-secure software. Non-secure applications can call the library's interface directly, just like a normal function call, the CPU will turn into secure state to execute secure code when the secure library interface is called. This calling mechanism is supported by the ARMv8-M TrustZone hardware extension, which greatly improves the efficiency of the call. And such mechanism makes the software more concise and efficient in design, so it is more suitable for applications in the IOT field.

CONCLUSION

The system can meet the specifications and requirements of PSA and can be applied in resource-constrained microcontroller systems. A single CPU of the ARMv8-M architecture can run this complete software system. To some extent, the cost of the secure system is reduced, and the following security requirements can be met.

(1) Data protection: Sensitive data is stored in a secure space and can only be accessed directly by secure software. Non-secure software can only request security services through specified API, and the secure software will check the validity during access.

(2) Firmware protection: The built-in secure firmware of the device can be stored in a secure space to prevent malicious modification.

(3) Operation protection: For device security control, important operations can be
only operated in the secure software. Some peripheral can be configured the as a secure resource, it can only be accessed by secure world, and this way avoid the malicious operations in the non-secure world.

(4) Secure device management: The data reported by the device and the remote control and authentication of the device can be implemented in an encryption way. The keys used for encryption and decryption can be securely stored in the secure world.

ACKNOWLEDGMENTS

This research was partly supported by the Core Electronic Devices, High-end General Chips and Basic Software Products Projects of China (2017ZX01030204).

REFERENCES

1. Bo Zhao, Yu Xiao, Yuqing Huang, A Private User Data Protection Mechanism in Trust Zone Architecture Based on Identity Authentication. Tsinghua Science and Technology, 2017, 02: 218-225.
2. Joseph Yiu, Whitepaper - ARMv8-M Architecture Technical Overview.pdf, https://community.arm.com/developer/ip-productsprocessors/b/processors-ip-blog/posts/whitepaper-armv8-m-architecture-technical-overview.
3. ARM, PSA-Introductory-Architecture-Overview.pdf, https://www.arm.com/why-arm/architecture/platform-security-architecture/psa-resources.
4. Joseph Yiu, Software Development in ARMv8-M Architecture, https://community.arm.com/developer/ip-productsprocessors/b/processors-ip-blog/posts/embedded-world-conference-2-17---software-development-in-armv8-m-architecture-1755775143.
5. Joseph Yiu, Milosch Meriac, High-End Security Features for Low-End Microcontrollers,https://community.arm.com/developer/ip-productsprocessors/b/processors-ip-blog/posts/ew2017---high-end-security-features-for-low-end-microcontrollers.
6. ARM, Building a Secure System using Trust Zone Technology, https://developer.arm.com/docs/genc009492/latest/trustzone-hardware-architecture. Joseph Yiu, The Next Steps in the Evolution of Embedded Processors for the Smart Connected Era, https://community.arm.com/developer/ip-productsprocessors/b/processors-ip-blog/posts/embedded-world-conference-2016---the-next-steps-in-the-evolution-of-embedded-processors-for-the-smart-connected-era.
7. ARM, Trust Zone technology for the ARMv8-M architecture, https://developer.arm.com/products/architecture/m-profile/docs/100690/latest/arm-trustzone-technology.