Design aspects for Upgrading Firmware of a Resource Constrained Device in the Field

Poonam Pingale, Kalpana Amrutkar, Suhas Kulkarni

Abstract— **Upgrading firmware of a resource constrained embedded device installed in the field is the necessity of the present day embedded systems. Taking device to the manufacturer or manufacturer reaching to the field becomes difficult once the embedded device is installed. The feature of in-field firmware upgrade in the embedded device makes the task of upgrading firmware easy. Although, the process of upgrading gives rise to certain issues related to safety and security. In spite of lot of research to address these problems, safety and security of upgrade process remains challenging. This paper discusses the architectural design aspects to develop an in-field firmware upgrade system for embedded device based on ARM platform using Ethernet medium. Our implementation approach for ARM based device provides solutions to most of the safety issues occurring during in-field firmware update. The security is achieved using lightweight encryption algorithms PRESENT and CLEFIA. The performance comparison for the two algorithms is presented. Our architecture provides feature of normal working of device when firmware update is done whereas many such systems halt their normal functionality. The previous firmware image is also retained and can be switched back called as Rollback firmware in our approach.**

Keywords— in-field firmware upgrade, bootloader architecture, safety, security, PRESENT, CLEFIA

I. INTRODUCTION

Embedded devices are found in every field controlling our day to day activities of life. The period of firmware development for these embedded devices depends upon the critical nature of the system. The firmware may contain bugs which arise after system installation. Once the system is installed on the site, it becomes difficult for fixing such bugs and modifying the firmware. Also upgrading firmware of device is needed when a new feature is to be added to the device improving the functionality of the system. The need for upgrade arises for compatibility of a new hardware peripheral of the device. The in-field firmware upgrade proves a boon making this task smoother. The time to market of the system can also be decreased by developing a version fulfilling the requirements of the system and then upgrading the firmware later for more features [1].

The feature of upgrading firmware in field requires a bootloader residing in the device flash memory. Most of the times bootloader controls the upgrade process when the update is sent to the embedded device. It also takes care of selecting and switching of firmwares in the flash memory [2]. The upgrade is carried out sometimes by application itself where as bootloader performs switching.

Poonam Pingale, RMD Sinhagad School of Engineering, Pune, India

Kalpana Amrutkar, RMD Sinhagad School of Engineering, Pune, India.

Suhas Kulkarni, Wavelet Technologies Pvt. Ltd, Pune, India.

The concern about the bootloader is its memory size which is an overhead for the flash memory of embedded device in which both application firmware and bootloader code are placed. Boot loader design is also constrained for RAM (Random Access Memory) usage so it should utilize minimum RAM as well as flash [3].

Many problems are observed during firmware upgrade of the device, making the whole process unsafe and unsecure [2]. The correct version of incoming firmware, authenticated sender of firmware, trusted destination device, integrity are major challenges for successful in field firmware upgrade system. The firmware can be modified by an attacker while it is transmitted over the media. For instance if the attacker modifies the firmware of a sensor node to convey wrong data to server then the system dependent on sensor data may misbehave. The sender or transmitter of the new firmware can be malicious who can send the incorrect firmware image. The end device can be changed and allowed to receive the upgrade making device authentication necessary.

The present day applications such as smart grid or home automation, IoT (Internet of Things) are ruling the world. The applications like IoT need robust security mechanisms as smallest part of the world will also be connected to the internet; its in-field firmware upgrade with high security is a major aspect for today's world [4]. The paper will discuss the deign problems in initial sections and later the implementation of our system. The solutions offered by us are presented further solving most of the issues.

II. DESIGN PROBLEMS

The major problems occurred while upgrading firmware of a device can be divided into three main categories i.e. architecture of bootloader, safety of upgrade and security of overall process.

A. Bootloader Firmware Architecture

Bootloader is the most important part for in-field update [5]. The major constraint on the Bootloader is that it should be very compact. The bootloader memory footprint present in the flash of device microcontroller decreases the area available for the actual application. There are different approaches in which bootloader can be implemented [2]. In first approach, bootloader performs all the firmware update related tasks like receiving updates from manufacturer, validating it, updating the flash memory and finally switching to the updated firmware. This makes bootloader bulky, consuming more flash space giving reduced memory for application. When bootloader performs upgrade, runtime functionality of device will be stopped. In second method, bootloader selects and switches to application firmware and update related tasks are performed by application. This makes the bootloader

978-1-5090-0774-5/16/$31.00 © 2016 IEEE

lightweight and application memory is increased. We have selected this method as it is advantageous retaining the normal functionality of the device while device firmware is updated. Validation of received firmware is carried out without disturbing system functionality.

B. Safety of Firmware Upgrade

The problem faced is the Safety of firmware upgrade process. It may fail at many stages during upgrade due to power failure, network failure, firmware version issues etc. This may stop the normal working of device and loss of working firmware image. The possible ways to address these problems are memory partitioning, communication protocol stack, packet acknowledgement etc. [2].

C. Security of Firmware Upgrade

Integrity and confidentiality are the major issues concerning security of devices [2]. The security can be achieved by various means like using encryption algorithms [1], digital signatures, hash functions, message authentication codes etc. The existing technologies used for security are not suitable due to their bulkiness for embedded device which is resource constrained device with low power consumption and low computation power.

The traditional encryption algorithms require more computational power and memory due to huge SP (Substitution-Permutation) boxes along with complex calculations involving number of rounds. Algorithms like AES (Advanced Encryption Standard), DES (Data Encryption Standard), Blowfish have been widely used in current systems [3]. They are not efficient for the resource constrained embedded devices. Many light weight algorithms like PRESENT, CLEFIA, HIGHT (HIGh Security and light weigHT), IDEA (International Data Encryption Algorithm), SEA (Scalable Encryption Algorithm), TEA (Tiny Encryption Algorithm) etc. are emerging, designed specially to work with limited resource embedded platforms addressing the security issue. Table I gives the comparison of block size and key size of few algorithms. ECC (Elliptical Curve Cryptography) is an upcoming technique providing better security than symmetric techniques using secret key which is a public or shared key algorithm used for embedded security. The traditional algorithms using public key like RSA have large key size whereas ECC provides a smaller key size and same security level [6].

The light weight algorithms PRESENT and CLEFIA have gained ISO/IEC standard ISO 29192-2 for lightweight cryptography [7]. CLEFIA and PRESENT prove to be efficient in software as well as hardware [8]. CLEFIA uses feistel network in which the input block undergoes division in sub blocks and further processing on them [9-10]. It uses 128, 192 or 256 bits key and 18, 22, 26 rounds for key expansion respectively. PRESENT follows the SP network with 31 rounds and key size being 80 or 128 bits [11]. More the key size more will be the effort needed for the attacker to hack it. Our system involves implementation of lightweight algorithms PRESENT and CLEFIA algorithms on ARM Cortex M4 platform along with their comparison.

TABLE I. COMPARISON OF LIGHTWEIGHT ALGORITHMS

Algorithm	**Block Size (Bytes)**	**Key Size (Bytes)**
CLEFIA	16	16/24/32
PRESENT	8	10/16
IDEA	8	16
HIGHT	8	16

III. ARCHITECTURE DESIGN AND IMPLEMENTATION

We have designed a complete firmware upgrade system for ATSAM4E8C Micro-Controller. Fig. 1 shows the overall view of the system. ATSAM4E8C is a 32-bit Micro-Controller operating at 96 MHz clock frequency based on ARM Cortex M4 core. The firmware is sent by a PC software application developed in the IDE (Integrated Development Environment) named Qt Creator which is cross platform framework. Software application communicates with end device via Ethernet which is a well-established existing network. The applications like Internet of things use Ethernet to connect the devices to centralized servers. So this media helps the integration with current systems easy. IPv6 protocol in IP (Internet Protocol) layer and UDP (User Datagram Protocol) protocol in transport layer facilitate firmware upgrade of large number of devices connected in the network simultaneously. Ethernet media provides high speed and low latency in communication than other existing media. The Firmware will be broadcasted over the network which is updated by the device.

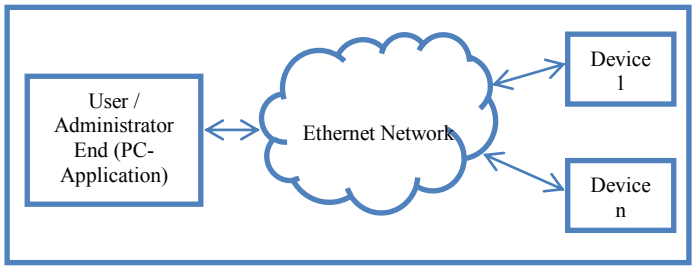

Fig. 1. System Overview

A. Features of Proposed System

- *Roll Back*: The very crucial feature provided in our system is rollback of firmware. It means that the administrator can switch between previous firmware version and new updated version. Normally firmware upgrades the latest version by replacing the current firmware version, if the update fails due to any reason there is loss of working firmware and device may become un-operational. Our approach will make both current working and previous version of firmware copies available all the time and they can be switched as per requirement. When the update is received it will be stored in flash by current working firmware. After

successful download and validation user can shift to new firmware image and if required again back to current firmware using this rollback feature.

- *Runtime Functionality of Device:* The existing systems stop their Run Mode functionality whenever firmware upgrade is started. Most systems perform the upgrade task in bootloader where the runtime functionality of the device is stopped and device enters in to stand-by mode during upgrade process. In the proposed solution, the upgrade is carried out without halting the normal device operations.

B. Addressing the Problems

The overall system design handles most of the issues discussed above. The safety of upgrading the firmware is addressed by partitioning of flash memory, application layer protocol design and bootloader architecture design and so on. The security of the firmware upgrade system is achieved using encryption algorithms. Two lightweight algorithms PRESENT and CLEFIA are implemented in our system. The results of comparison of both are presented in following sections.

- *Partitioning of Flash Memory*: The 512KB Flash memory of ATSAM4E8C is divided in such a way that bootloader gets minimum area i.e. smallest single sector of 8KB and each application gets maximum space of 192KB. The total application area is 384KB which is equally divided in two parts i.e. region A and region B as shown in Table II. One sector of 48KB is reserved for the configuration parameters such as current working firmware copy, status of the update process. This information will be used by bootloader firmware while selecting the firmware to boot; it also forms a communication link between application and bootloader. If region A has current working firmware then region B will have previous version and vice-versa. The received update will be downloaded in region B when current working firmware is A.

- *Application Layer Communication Protocol*: The application layer communication protocol design for the system takes care about various aspects so that the problems could be handled. If the update process fails in the intermediate stages of communication, fresh update is started. The standard Ethernet stack is used with UDP protocol at transport layer. The UDP though being less reliable than TCP (Transmission Control Protocol) facilitates the communication in case of large number of devices with less latency. CRC (Cyclic redundancy check) ensures the integrity and detects the errors incurred during transfer. A firmware read version command informs about the current version status avoiding the error due to version mismatch discarding the update before starting.

TABLE II. FLASH MEMORY PARTITION

Physical Flash Address	Contents	Size
0x00400000	Bootloader	8KB (small sector 0)
0x00402000	Reserved for bootloader future use	8KB (small sector 0)
0x00404000	Configuration Data	48KB (large sector 0)
0x00410000	Application A	192KB(sector 1,2,3)
0x00440000	Application B	192KB(sector 4,5,6)
0x00470000	Future Use	64KB

- *Bootloader Firmware Implementation*: Bootloader is a lightweight approach occupying minimum footprint in Micro-Controller Flash. The main purpose for adapting this approach of architectural design is to keep device in operational mode during firmware update. Bootloader uses the configuration information while switching to firmware after the update is successfully downloaded.

- Application Firmware Implementation: Current working application firmware checks if the firmware update from administrator connected in Ethernet network is sent. The application accepts this update depending upon the status of the other region firmware. For instance, A is working application then if status of region B indicates that it is empty or no previous update process is running then current application in region A will allow the update to be downloaded be downloaded in region B. The flow of application working and bootloader working is explained in fig 2. The decryption of incoming packets is performed by working application which uses the key stored in the RAM. The CRC verification is performed for every received packet and packet is discarded if CRC does not match terminating the process immediately. A new fresh update process has to be started if process fails due to CRC mismatch. Final complete firmware CRC check is also included which is computed after all packets are received. This ensures more security as the modifications in firmware by the attacker after administrator transmits the upgrade, can be detected. The update will not be successful if CRC does not match and administrator has to send the firmware update again. The configuration information in Flash is updated after update completion in order to inform bootloader about the status. The Roll back feature will be used to select the application firmware in region A or region B.

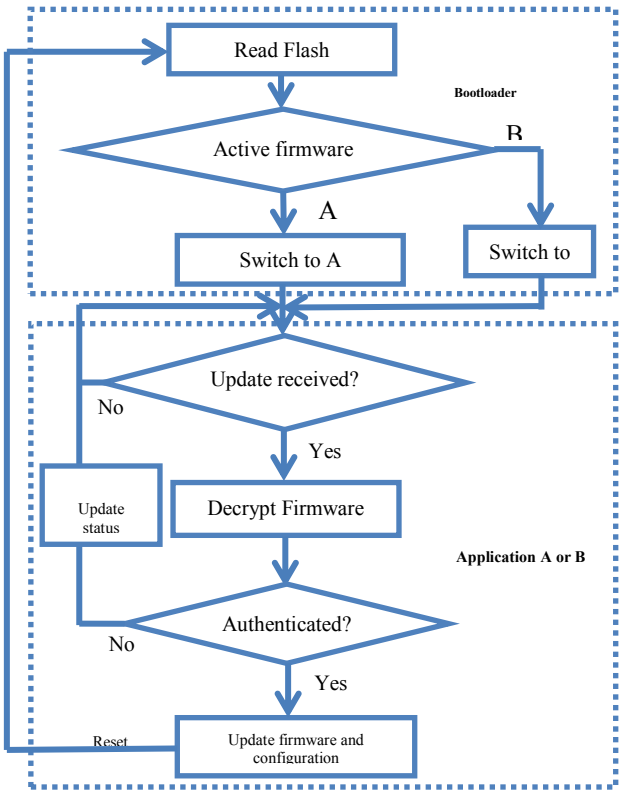

Fig. 2. Bootloader and Application Implementation

- *Implementation of Encryption Techniques for Security:* Security is achieved by encrypting the firmware image data before transmitting. The job of attacker becomes difficult when he sees encrypted data instead of plain data. Encryption of data introduces confusion and diffusion properties which are barrier against different attacks like linear, differential etc. As the traditional encryption algorithms provide sufficient security for defense but are not suitable for constrained devices, we have implemented lightweight encryption algorithms to reduce the computation overhead and memory overhead. Algorithms which are accepted as standards for limited resource embedded systems i.e. PRESENT and CLEFIA are implemented in our proposed system. The paper focuses on their performance analysis.

C. *Experimentation and Results*

The system is tested for different sizes of firmware images and verified that the process does not fail. The time required for update is in linear relation with the size of firmware binary. Firmware update was sent to the device while device was in run mode and it was updated successfully in Flash. Rollback firmware feature is tested after firmware upgrade is done. Further, the encryption algorithms PRESENT and CLEFIA are implemented. The block size for both the algorithms is 128 bits. The key size is different for both 80 bits for PRESENT and 128 bits for CLEFIA. The key is present in the application firmware i.e. RAM for both the algorithms.

The experimentation is carried out for both encryption algorithms which contribute to the security of the firmware update system. The encryption and decryption implementations are on different processors as encryption is done on PC and decryption is done on embedded device. Table III gives comparison of memory requirement and speed of computation. The encryption as well as decryption times are measured excluding the key calculation time. The encryption time and decryption time of both the algorithms are compared by varying the packet size as shown in fig 3 and fig 4. Each packet is of 512 bytes and algorithm processes a block of 16 bytes at a time. CLEFIA performs much faster than PRESENT in terms of encryption as well as decryption time. RAM memory requirement for PRESENT is lesser.

TABLE III. PERFORMANCE COMPARISON OF ENCRYPTION TECHNIQUES

Algorithm	**RAM Memory (Bytes)**	**ROM Memory (Bytes)**	**Encryption time/Block (ms)**	**Decryption time/Block (ms)**
CLEFIA	1576	48	0.0363	0.247
PRESENT	1035	248	0.165	1.514

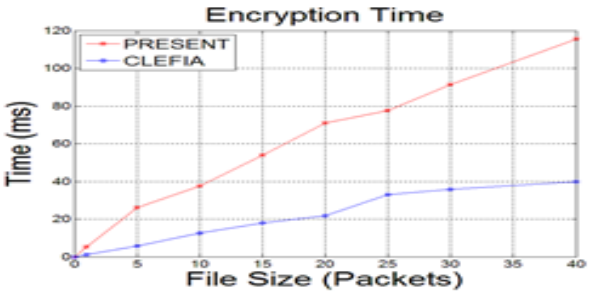

Fig. 3. Encryption Time Comparison

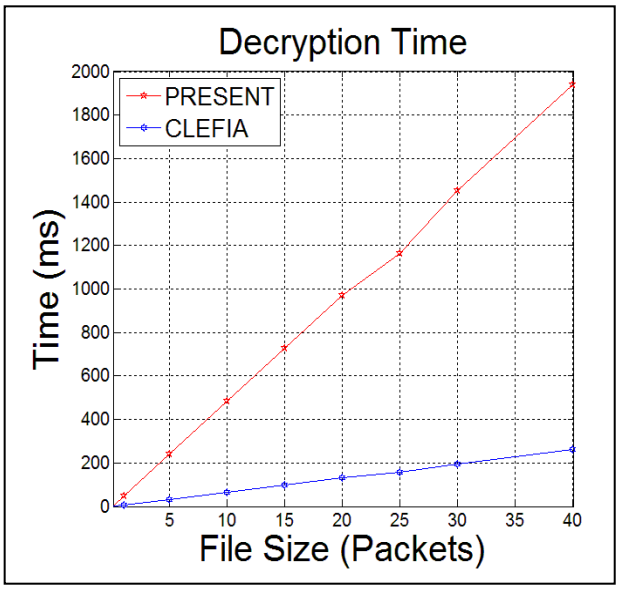

Fig. 4. Decryption Time Comparison

IV. CONCLUSION AND FUTURE SCOPE

The problems observed and current solutions in updating the firmware of the resource constrained embedded system are studied thoroughly. The proposed system design has addressed most of the issues mainly bootloader architecture, safety and security. Rollback firmware, normal working functionality simultaneously with firmware upgrade is the feature of the system. The implementation of the cryptographic techniques provides security to the system. CLEFIA is efficient implementation than PRESENT cipher in software from the obtained results requiring lesser time for encryption and decryption. In future, the comparison of different other encryption techniques, hash functions, digital signatures message authentication codes etc. contributing to the security of system will be done with the existing ones. Also optimization will be done to the existing implementations in order to achieve lower encryption and decryption cycles, memory footprint and better security.

ACKNOWLEDGMENT

The work carried out in this paper is sponsored by Wavelet Technologies Pvt. Ltd. I would like to thank Dr. Udpikar, Wavelet Technologies Pvt. Ltd. for their valuable support. I express my sincere gratitude to the staff of E&TC department, RMD Sinhagad School of Engineering and members of Wavelet Technologies Pvt. Ltd. for their guidance at every stage.

REFERENCES

[1] "White paper Secure Firmware Upgrade System", HCL Technologies, 2014.

[2] Application Note, "Atmel AT02333: Safe and Secure Boot loader Implementation for SAM3/4".

[3] Ian Pratt, Shiyun Zhong "Bootloader design considerations for resource constrained microcontrollers in RFID reader designs", in *IEEE RFID Technology and Applications Conference (RFID-TA)*, 2014, pp. 50-55.

[4] "A new approach to IoT Security, 5 key requirements to Securing IoT Communications", PubNub Inc., 2015.

[5] Jacob Beningo, "Boot loader Design for Microcontrollers in Embedded Systems*", Embedded Systems Conference*, Rev A, 2012.

[6] Sukriti Jalali, "Trends and Implications in Embedded Systems Development", TCS.

[7] Gaurav Bansod, Nishal Raval, Narayan Pisharoty, "Implementation of new Lightweight Design for embedded Security", *IEEE Transactions on Information Forensics and Security*, vol 10, no. 1, pp 1556-6013, 2013, in press.

[8] Charalampos Manifavas, George Hatzivasilis, Konstantinos Fysarakis, Konstantinos Rantos, "Lightweight Cryptography for Embedded Systems - A Comparative Analysis".

[9] Sony Corporation, "The 128-bit Block cipher CLEFIA Algorithm Specification", June1, 2007.

[10] T. and Shibutani, K. and Akishita, T. and Moriai, S. and Iwata, T. Shirai, "The 128-Bit Blockcipher CLEFIA (Extended Abstract)" in *Fast Software Encryption*, 2007, pp. 181--195.

[11] A. Bogdanov et al., "PRESENT: An Ultra-Lightweight Block Cipher", *In Proc. of the 9th International workshop on Cryptographic Hardware and Embedded Systems - CHES 2007*, pp. 450–466.