Software Implementation of Secure Firmware Update in IoT Concept

Lukas KVARDA, Pavel HNYK, Lukas VOJTECH, Marek NERUDA

Department of Telecommunication Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Technicka 2, 166 27 Prague, Czech Republic

kvardluk@fel.cvut.cz, hnykpav1@fel.cvut.cz, vojtecl@fel.cvut.cz, nerudmar@fel.cvut.cz

DOI: 10.15598/aeee.v15i4.2467

Abstract. *This paper focuses on a survey of secure firmware update in the Internet of Things, design and description of safe and secure bootloader implementation on RFID UHF reader, encryption with AES-CCM and versioning with use of external backup flash memory device. In the case of problems with HW compatibility or other unexpected errors with new FW version, it is possible to downgrade to previous FW image, including the factory image. Authentication is provided by the UHF RFID service tag used to extract unique initialization vector of the encryption algorithm for each update session. The results show slower update speed with this new upgrade method of approximately 27 % compared to older one, using the only AES-CBC algorithm.*

Keywords

Firmware versioning, IoT, security.

1. Introduction

Concept of the Internet of Things (IoT) is very topical and popular theme as can be seen on forecasts made by Gartner research company [1]. According to this estimate, a number of connected devices will be 31 percent higher than last year, which means 8.4 billions of connected IoT devices in 2017. Applications for the consumer segment will represent 63 percent (5.2 billion) of the total number of applications in use, for instance, digital set-top boxes, game consoles, smart TVs and PCs. The rest, 37 percent (3.1 billion) are devices like commonly used commercial security cameras or various kinds of intelligent sensors which will be most in use by businesses. According to forecast, connected devices in 2020 (20.8 billion) mentioned by Gartner last year and this year's estimate, which is 400 million smaller, we can assume, that current rising trend is slowly approaching the imaginary ceiling. However, we still talk about the huge amount of devices, where security should not be underestimated.

There are still a number of factors contributing to the deterioration in IoT security. One of them is the fact, that many IoT device producers use the same Software Development Kit (SKD). The problem may occur if the company that produces the SDK releases a new version riddled with bugs, then all devices using this SDK will be vulnerable just through these bugs. If the same software is used, it will not depend on the device type. The second one is that the security measures are often neglected by users, such as keeping default device name and password, or unconcerned with regular updates of firmware (FW). The last major problem is that manufacturers of IoT devices neglect or does not solve security at all. As we mentioned in [2], this year, the manufacturers also try to get the lowest possible cost of device, short time to market and lower product maintenance costs when deploying firmware update mechanisms. This approach leads to the fact that many of devices are not fully tested in time and for the security issues. For these reasons, there is a need for a frequent update of FW with a new version that will fix bugs.

Enormous and still growing number of IoT devices makes it impossible to implement the FW update by sending the device back to the manufacturer or by sending manufacturer technical support which would be solving the update directly on device in the field [3]. If these options are used for FW update it costs too much money, and take up too much time. Therefore, these update methods are inappropriate, and a remote FW update is becoming increasingly used instead. The ability to re-uploading newer versions of FW to deployed devices without major costs, has in addition to fixing bugs other benefits, such as allowing

service and support, which can significantly extend lifecycles of device. According to IEEE [4], these services should be a matter of course today.

On the other hand, this way of updating brings us the risks that security experts have warned us for a long time. Unfortunately, in most cases, security measures are neglected, which may lead, for example, to:

- threats to intellectual property,
- unauthorized device control,
- product cloning, revenue reduction, and trademark damage,
- compromised device communication,
- MITM (Men in the Middle attack) and
- access to another system.

The need for IoT security is illustrated by the recent massive DDoS attacks by Mirai botnet [5] on DYN servers that took down Twitter, Spotify, Netflix, GitHub, Amazon and other websites. These attacks were led by IoT devices such as digital video recorders, home routers, security cameras, etc. According to ESET research suggests [6], at least 15 percent of home routers are unsecured from the estimated global number of 105 million devices and therefore there is a huge threat of infecting malware such as Mirai.

Another disturbing attack was demonstrated by researchers who managed to take control of Jeep and were able to turn off the engine remotely while driving on the highway. To protect both driver and passengers secure update of vehicle electronics is essential [7] and [8]. Because of these and many other threats, the risk of a cyber attack on the FW update process should not be underestimated.

2. Today Update Secure Solutions

Not everyone underestimates the security risk when updating FW; there are companies offering a ready-made solution. Atmel company, for example, offers the demo kit to demonstrate a secure FW update over the Ethernet. The demo kit uses the ATSAM4SD32 microprocessor (MCU), dual bank Flash firmware update and authenticated encryption AES-GCM (Galois/Counter Mode) mode support. Atmel also offers security at a hardware level called CryptoAuthentication devices. These devices can handle secret keys protection, key generation and their management. Also, it solves security of communication bus between the MCU and the crypto device by authentication, thus protecting against MITM attacks. Compared to others, crypto devices have another significant advantage, which lies in the design layout. Atmel uses the active metal shield of package to protect the chip itself, which prevents the use of side-channel attacks, but also detects if someone tries it.

Another big company offering the made-ready solution is Texas Instrument with their Crypto-Bootloader. This design allows to securely update an FW of microprocessor directly connected to the network. They use a custom utility to use WAN to access or connect to endpoints. This system is implemented on the ultra-low power MCU MSP430FR5969 containing hardware accelerator for AES-256.

The Czech company Jablotron with the open source BigClown project also deals with IoT security solutions. This is a modular system built on the MCU STM32L083CZ, which, like the previous one, has built-in AES 128-bit encryption machine. Authentication and update of AES security keys are solved here by the Atmel crypto device. Specifically, it is ATSHA204A using the SHA-256 Hash algorithm with the Message Verification Code (MAC) and Hash-based Code Verification (HMAC).

There are also companies entering to market with finished products. One of them is Thales company, which offers HSM (Hardware Security Modules) allowing key generation and safe storage. Unlike previous designs, authentication is done by assigning unique digital certificates generated by HSM to each device separately.

Given that there are many suggestions on how to ensure secure FW updates [9]. It would be wise for anyone who wants to participate in the world of IoT, take these possibilities into account and use them for the new designs. Or, at the very least, design the own security way to keep the FW update process safe. Poulter, Johnston and Cox [10] proposed an implementation of the Secure Remote Update Protocol (SRUP) for IoT devices.

3. Implementation Details

When designing the update process, account should be taken not only of possible threats but other variables, such as memory requirements, computing power, power consumption, security level, and the cost of the proposed system. In our design, we tended more to computing performance, memory, and price at the expense of security when choosing an encryption algorithm, as we apply the FW update to the existing AutoEPCIS UHF RFID Reader in third version of hardware in Subsec. 3.1.

Each cryptographic algorithm should provide three basic features: privacy, integrity, and authenticity. In our previous secure FW update [2], AES-CBC algorithm that provided only the first two properties is used. The authenticity has to be implemented as well. One way to do this is to use digital signatures. From the point of view of computing power and memory requirements, the PIC32MX microprocessor, which is used in AutoEPCIS UHF RFID Reader, should be able to handle more demanding asymmetric encryption (public key, private key). Nevertheless, we have opted out of this option, because it is necessary to use a third-party Certification Authority (CA) to authenticate the user. We want to avoid this solution because of the prices for these services. Although digital signature can be used without CA, then the method can be susceptible to MITM attack. Alternative solution is the use of crypto RFID tag [11].

In our design, the authenticated encryption algorithm in CCM mode is used. This algorithm does not threaten the above-mentioned MITM attack or malleability attack [12]. The sufficient attention must be given to the problem with the safe storage of secret keys encounters since key storage is one of the most critical parts of security. When designing new devices, using of special ICs for safe key storage as mentioned in the Sec. 2. is recommended, because HW security is much stronger than SW security. In our case, we are limited by the existing HW, so we choose to store secret keys into the FLASH memory of MCU, which has protection against reading data from memory. We consider this action to be fully satisfactory for our purposes.

Due to the ease of implementation and utilization of the existing system, we have chosen to use the AES-CBC encryption algorithm. It uses a service RFID tag which is located in the vicinity of the reader and ensures authentication. The RFID tag detects the functionality of the RF part of the reader (whether the antennas are connected) and to measure the Received Signal Strength Indication (RSSI) changes to monitor if there is a change in the environment such as shadowing the tag, coaxial cable damage or antenna movement.

The main idea of a safe FW update is that only the service RFID tag owner (physical security key) is able to update a device by certified NFI (New Firmware Image) from the manufacturer. We ensure that we generate a Pseudo Random Number (PRN) that is stored in the user memory of the tag. The PRN size depends on the size of the service tag user memory. Confidex's UHF RFID tag with the Impinj Monza 4QT chip, which has a 512-bit user memory size is chosen, so the PRN has the same size. From this PRN, 128 bits are randomly selected and declared as an Initialization Vector (IV). This is unique for every other session and must not be repeated. Subsequently, NFI encryption is performed by a secret key and created IV. The secret key is also generated as a pseudo-random sequence of numbers and then programmed in a trusted environment to the MCU along with the bootloader. This encrypted NFI is sent along with SNO (Selected Numbers Order) positions to the device. The big advantage of AES-CBC method is not sending IV, but only positions that will be set up on the side of the device, meaning that even if the attacker gets the data, without appropriate service tag they will be useless. In addition, the PRN tag collection is internal, the device does not publish the memory of the tag, it only loads it into the temp memory and deletes it after updating. So if an attacker wants to get a PRN, he has to be physically in place and read this tag with another RFID UHF reader. He is still able to decrypt the NFI, because he needs the secret key stored in the FLASH MCU, which we consider to be a considerable effort, compared to what he would have obtained in our case. In addition, it would still be possible to encrypt the SNO itself and lock the service tag password, but here we get to the security by obscurity.

We assume that the customer has received a service tag along with the device and therefore both parties have a secret key and PRN. The proposed FW update method is shown in the Fig. 1 and can be divided into five basic steps: The first is to generate the NFI that we want to update the device. Next, we will need a session IV, which is obtained by applying the *PickRandom128* function to an already generated PRN. This feature randomly selects 128 bits, and at the same time stores their position against a 512-bit unit. The output is our session IV and SNO. Second, NFI encryption is performed using the AES-CBC algorithm using session IV and the secret key. The third step is the distribution of the encrypted NFI and SNO to the target device. The fourth step is an extraction of session IV using the

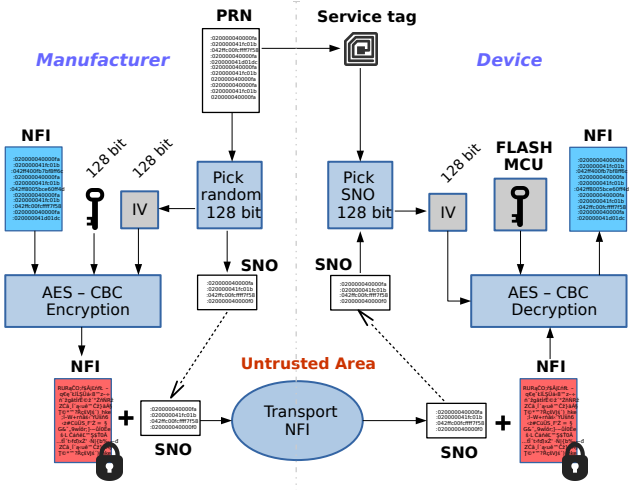

Fig. 1: Proposed FW update method.

PickSNO128 function. This feature has a 512-bit PRN stored in the service tag and received SNO. Based on this information, the correct session IV is reproduced. The last step is to decrypt the NFI and then upload it to the MCU. A detailed description of the FW update process is described in Subsec. 3.3.

3.1. AutoEPCIS UHF RFID Reader

Last year, we redesigned hardware of the AutoEPCIS UHF RFID Reader, mainly because of need for more adapt power supply. Now the reader is not limited by 5 V input, but can be powered from wider range of input voltage (7–13 V).

Summary of changes in the third version of reader, Fig. 2:

- more adapt power supply circuit,
- transmitter leakage suppression,
- packet acknowledgment,
- external 32 Mbit SPI flash memory,
- EMI shielding and
- HW antenna detection.

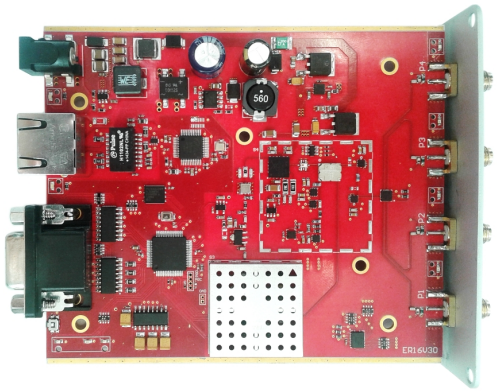

Fig. 2: AutoEPCIS UHF RFID reader version 3 – Top View.

3.2. Previous Bootloader

In the old version of the bootloader, as presented [2], symmetric encryption AES in block cipher mode of operation (CBC) is used. Keys are stored in MCU flash memory space and the space is code protected to prevent unauthorized readout (MCU can be only erased).

To decrypt FW image, it is firstly necessary to store the new FW image to temporary memory area of MCU and then perform decryption process, because of no other storage unit. This created memory space is only a temporary space, where its size depends on the size of uploaded FW Fig. 3.

In case of updating error, report to the terminal is created and device needs to be flashed again.

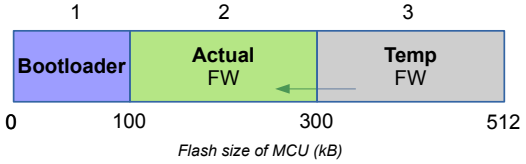

Fig. 3: Partitioning of MCU Flash memory.

3.3. Redesigned Bootloader

We modified our own protocol to send encrypted data to the device, added authentication and integrity check. All data is sent to the device via UDP. The maximum data payload is limited to 1024 bytes, which is a compromise between flash speed and RAM usage of MCU.

Before starting the FW update process, the authentication has to be performed. It is managed in CCM mode (Counter with CBC-MAC). A desktop PC is used as the update terminal. The terminal must first generate session IV from random selected 16 bytes of service RFID tag memory (SNO). The secret key is stored at device flash memory and it is known by the terminal. The following calculation of protected checksum (MIC) needs session IV and secret key as inputs. The host then sends authentication frame to the device, Fig. 4.

Authentication is valid only when device recalculated MIC is equal to MIC sent in authentication frame. The device knows the secret key; the only thing it does not know is session key, but via SNO it can generate it. To do so, the device must "see" the RFID service tag to read from its memory bank. The device responses to authentication request frame with defined error code chart, where the value of 0 means valid access, Fig. 5.

6	1	16	16	2
Header	**Subcmd**	**SNO**	**MIC**	**Tail**
Packet ID, size, cmd	BOOT_AUTH	Selected Number Order	Protected checksum	CRC16

Fig. 4: Authentication request packet.

6	1	1	2
Header	**Subcmd**	**Result**	**Tail**
Packet ID, size, cmd	Bootloader subcmd	Command result	CRC16

Fig. 5: Data response packet.

Image of the firmware is encrypted in CBC mode. The same mechanism for creation of session IV from

SNO as in authentication is used. The secret key here is different then the key used for authentication, i.e. 2 different keys in MCU flash memory are stored. The image is separated to frames and is sent to the device by packets, where size of frame is limited to 1024 bytes. For example, in case of full packet usage and size 216064 bytes of image, 211 packets are sent to the device, Fig. 6.

6	1	0 - 1024	2
Header	**Subcmd**	**Payload**	**Tail**
Packet ID, size, cmd	BOOT_FILL	Encrypted FW image segment	CRC16

Fig. 6: Data frame with encrypted FW segment.

There is no other memory space to store versions of FW as a backup in the older version of bootloader, now external 32 Mbit SPI flash memory is used, managed to 5 blocks Fig. 7. First four blocks have a size of 400 kB, it is sufficient space for encrypted image which has currently 211 kB, except last one. The last block of memory is spared for storing device settings variables, such as network setup and antenna tuning results.

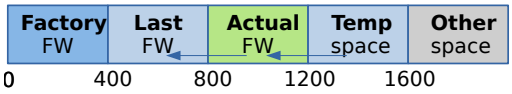

Fig. 7: Partitioning of External Flash memory.

The first block is an area where factory FW image is stored. This is the backup plan of last resort.

The second block is reserved as current running image backup, it is last known updated firmware image. Factory default devices have this area empty.

The third block is an operation area for uploading encrypted image of FW. Ongoing update process fills this area with encrypted segments of FW image.

After successful filling this area, hash function is applied to entire stored image in order to check data integrity (the MD5 algorithm is used). Hash is calculated at both transfer sides, terminal sends the hash result and comparison is done at the device side. If calculated hash is not corresponding, this third block is erased and entire update process must start over with authentication.

All backup images are stored encrypted. Figure 8 shows diagram of FW update process.

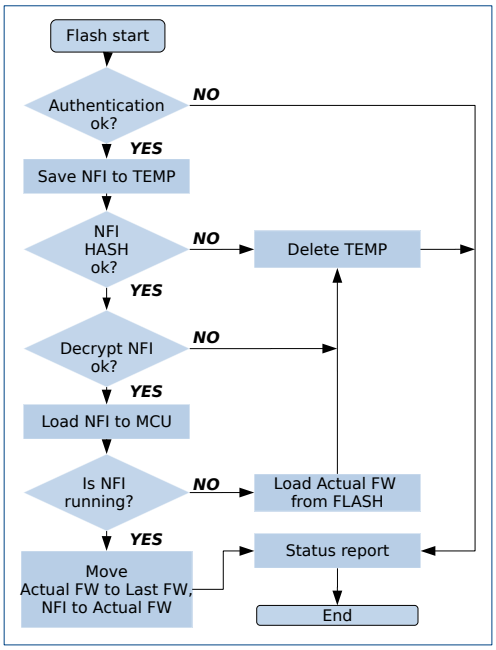

Fig. 8: Firmware update process diagram.

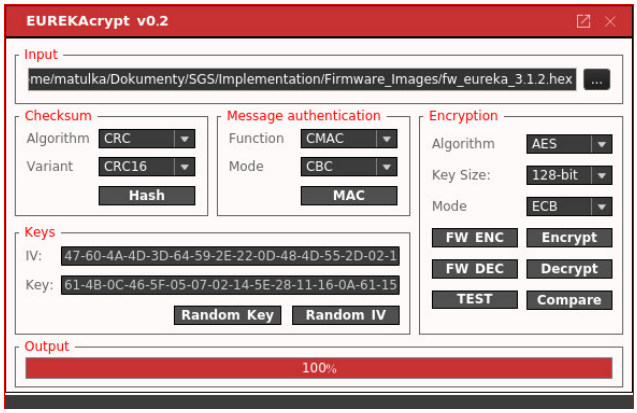

Fig. 9: A PC application to encrypt the firmware image file.

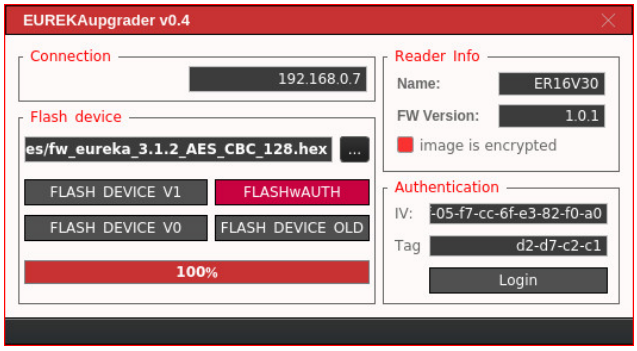

Fig. 10: A PC application to flash the device with a new firmware image file.

4. Measurements and Results

PC applications are modified in order to encrypt and load the FW image file onto device with authentication, Fig. 9 and Fig. 10. Checksum calculation, message authentication and extended encryption are added. All programming is written in C and is portable to a microcontroller.

We compared the RAM and flash memory utilization of the bootloader with authentication against previous

versions, Fig. 11. Now the bootloader occupies 52.7 kB of flash memory from the reserved area of 100 kB. Thanks to the code reduction of previous bootloader version the difference is only 3 kB of flash space.

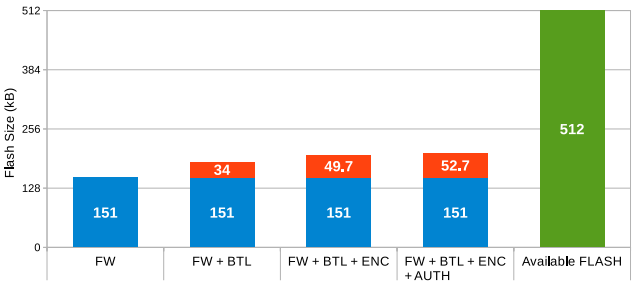

Fig. 11: Flash memory utilisation by bootloader type.

RAM usage is also optimized, Fig. 12, it is now smaller with authentication than before. We managed to reduce 0.66 kB of RAM space while adding authentication algorithm.

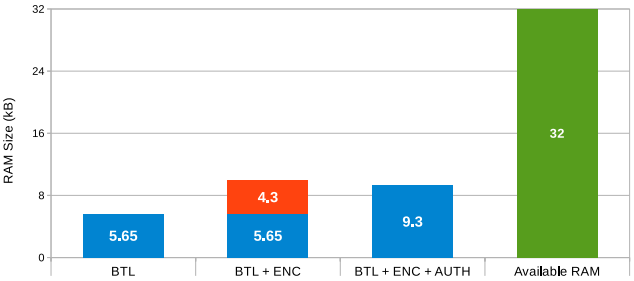

Fig. 12: RAM memory utilization by bootloader type.

The time of new update methods is measured, Fig. 13. All shown times are update times starting from the bootloader. Time needed to switch to the bootloader is approximately 2.5 seconds, but is not included in the graph. Time to update device with encrypted image file and authentication is 13350 ms, compared to previous version of the bootloader with no authentication, the difference is 3572 ms.

This time value includes authentication, save NFI to external flash memory, hash calculation and comparison, NFI decryption, NFI load to MCU, backup and reset, as shown in Fig. 8.

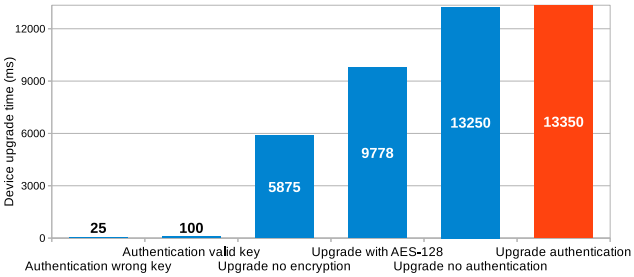

Fig. 13: Update time by encryption access to device.

Time of authentication, with wrong key is 25 ms and with the right key 100 ms, it is negligible values in comparison with update time of 13350 ms.

5. Conclusion

The paper describes a software implementation of a secure firmware update solution with authentication in an IoT context.

We integrated to existing device, i.e. UHF RFID reader, the authentication feature. Together with AES-CBC encryption we added CCM-based authentication. To avoid an asymmetric encryption and certification, we come up with the RFID service tag; it randomly creates initialization vector as input to authentication and encryption. Only the tag owner is able to update device by certified NFI. Secret key storage is left unchanged compare to previous version of the bootloader. Keys are stored in protected area of MCU; it is the limitation of hardware.

The measurements show requirements for MCU: 52.7 kB of flash memory and 9.3 kB of RAM, which is still within our limits. Update speed is 27 % slower with a new version of the bootloader, but it is not so critical given a less frequent rhythm of NFI release. In future, we would like to experiment with CryptoAuthentication devices, because we like the idea of security in one IC chip. Furthermore, we would like to focus on the comparison with other methods, in terms of CPU time, energy consumption, memory requirements and process errors.

Acknowledgment

This work was supported by Eureka grant "U-health: Auto-ID technology and the Internet of Things to enhance the quality of health services", Eureka ID: 11 158, by TA CR grant "The Multichannel Communication Platform for the Internet of Things (IoT)" TH02010568 and CTU internal grant "Security in Internet of Things and Industry 4.0" SGS16/159/OHK3/2T/13.

References

[1] Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. In: *Gartner* [online]. 2017. Available at: `http://www.gartner.com/newsroom/id/3598917`.

[2] KVARDA, L., P. HNYK, L. VOJTECH, Z. LOKAJ, M. NERUDA and T. ZITTA. Software

implementation of secure firmware update in IOT concept. *Advances in Electrical and Electronic Engineering*. 2016, vol. 14, no. 4, pp. 389–396. ISSN 1804-3119. DOI: 10.15598/aeee.v14i4.1858.

[3] PINGALE, P., K. AMRUTKAR and S. KULKARNI. Design aspects for upgrading firmware of a resource constrained device in the field. In: *International Conference on Recent Trends in Electronics, Information and Communication Technology*. Bangalore: IEEE, 2016, pp. 903–907. ISBN 978-1-5090-0774-5. DOI: 10.1109/rteict.2016.7807959.

[4] Internet of Things (IoT) Security and Privacy Best Practices. In: *IEEE Internet Initiative* [online]. 2017. Available at: `http://internetinitiative.ieee.org/images/files/resources/white_papers/internet_of_things_feb2017.pdf`.

[5] KOLIAS, C., G. KAMBOURAKIS, A. STAVROU and J. VOAS. DDoS in the IoT: Mirai and Other Botnets. *Computer*. 2017, vol. 50, iss. 7, pp. 80–84. ISSN 0018-9162. DOI: 10.1109/MC.2017.201.

[6] STANCIK, P. At least 15 percent of home routers are unsecured. In: *welivesecurity* [online]. 2016. Available at: `https://www.welivesecurity.com/2016/10/19/least-15-home-routers-unsecure/`.

[7] CHEN, C.-L., T.-T. YANG, C.-L. FAN and K.-H. WANG. A secure and with a low cost updated information system in VANET. In: *International Conference on Applied System Innovation*. Okinawa: IEEE, 2016, pp. 1–5. ISBN 978-1-4673-9888-6. DOI: 10.1109/ICASI.2016.7539884.

[8] MANSOR, H., K. MARKANTONAKIS, R. N. AKRAM and K. MAYES. Don't Brick Your Car: Firmware Confidentiality and Rollback for Vehicles. In: *0th International Conference on Availability, Reliability and Security*. Toulouse: IEEE, 2015, pp. 139–148. ISBN 978-1-4673-6590-1. DOI: 10.1109/ARES.2015.58.

[9] HUTH, C., P. DUPLYS and T. GUNEYSU. Secure software update and IP protection for untrusted devices in the Internet of Things via physically unclonable functions. In: *International Conference on Pervasive Computing and Communication Workshops*. Sydney: IEEE, 2016, pp. 1–6. ISBN 978-1-5090-1941-0. DOI: 10.1109/PERCOMW.2016.7457156.

[10] POULTER, A. J., S. J. JOHNSTON and S. J. COX. SRUP: The secure remote update protocol. In: *3rd World Forum on Internet of Things*. Reston: IEEE, 2016, pp. 42–47. ISBN 978-1-5090-4130-5. DOI: 10.1109/WF-IoT.2016.7845397.

[11] RISALAT, N. A., T. HASAN, S. HOSSAIN and M. RAHMAN. Advanced real time RFID mutual authentication protocol using dynamically updated secret value through encryption and decryption process. In: *International Conference on Electrical, Computer and Communication Engineering*. Cox's Bazar: IEEE, 2017, pp. 788–793. ISBN 978-1-5090-5627-9. DOI: 10.1109/ECACE.2017.7913010.

[12] GUILLEN, O. M., D. SCHMIDT and G. SIGL. Practical evaluation of code injection in encrypted firmware updates. In: *Design, Automation & Test in Europe Conference and Exhibition*. Dresden: IEEE, 2016, pp. 325–330. ISBN 978-3-9815-3707-9.

About Authors

Lukas KVARDA is a Ph.D. student of a study program at Telecommunication Engineering at the Czech Technical University in Prague, Czech Republic. His research interests include HW and SW design, RFID technology and cryptography.

Pavel HNYK is a Ph.D. student of a study program at Telecommunication Engineering at the Czech Technical University in Prague, Czech Republic. His research interests include HW and SW design, RFID technology and cryptography.

Lukas VOJTECH received M.Sc. and Ph.D. at Telecommunication Engineering at the Czech Technical University in Prague, Czech Republic in 2003 and 2010. He has been actively involved in several national and international projects. He is a leader of RFID laboratory at the Czech Technical University in Prague since 2010. His research interests are hardware prototyping and measurement especially in the of RFID technology, textile antenna design and localization.

Marek NERUDA received the M.Sc. and Ph.D. degree in electrical engineering from the Czech Technical University in Prague, Faculty of Electrical Engineering, Czech Republic in 2007 and in 2014, respectively. His research interests include RFID technology and electrically conductive textile materials.