J Supercomput
DOI 10.1007/s11227-016-1870-0

Blockchain-based secure firmware update for embedded devices in an Internet of Things environment

Boohyung Lee[1] · **Jong-Hyouk Lee**[1]

Abstract Embedded devices are going to be used extremely in Internet of Things (IoT) environments. The small and tiny IoT devices will operate and communicate each other without involvement of users, while their operations must be correct and protected against various attacks. In this paper, we focus on a secure firmware update issue, which is a fundamental security challenge for the embedded devices in an IoT environment. A new firmware update scheme that utilizes a blockchain technology is proposed to securely check a firmware version, validate the correctness of firmware, and download the latest firmware for the embedded devices. In the proposed scheme, an embedded device requests its firmware update to nodes in a blockchain network and gets a response to determine whether its firmware is up-to-date or not. If not latest, the embedded device downloads the latest firmware from a peer-to-peer firmware sharing network of the nodes. Even in the case that the version of the firmware is up-to-date, its integrity, i.e., correctness of firmware, is checked. The proposed scheme guarantees that the embedded device's firmware is up-to-date while not tampered. Attacks targeting known vulnerabilities on firmware of embedded devices are thus mitigated.

Keywords Blockchain · Firmware verification and update · Embedded device

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2014R1A1A1006770).

✉ Jong-Hyouk Lee
jonghyouk@pel.smuc.ac.kr; hurryon@gmail.com; jonghyouk@smu.ac.kr

Boohyung Lee
boohyung@pel.smuc.ac.kr

[1] Protocol Engineering Lab., Sangmyung University, Cheonan, Republic of Korea

Published online: 13 September 2016

1 Introduction

According to the Gartner's report [1], the Internet of Things (IoT) era will change our lives with network connected IoT devices. The number of IoT devices is expected to be 25 billion by 2020 and the number will continuously increase. The IoT devices are tiny and small, while those are mostly embedded devices designed for specific operations such as sensing and controlling.

Recent cyber attacks are targeting firmware, which is a software program on an embedded device [2], rather than services built on well-turned servers [3]. Due to limited resources and capacities of embedded devices, strong security properties have not been applied yet to the embedded devices. Many bugs and vulnerabilities of embedded devices are reported every day and those are being used by attackers to break into the embedded devices.

One of feasible ways to protect the embedded devices is to shorten the attack window time by installing a latest firmware. In other words, updating the embedded devices with the latest firmware helps to minimize effects of attacks targeting known firmware vulnerabilities. A secure firmware update requires a correct firmware version check, validation, and download, while supporting integrity. In addition, due to the increasing number of the embedded devices in the IoT era, excessive network traffic may occur when downloading the latest firmware simultaneously from a dedicated firmware update server. The secure firmware update thus requires a means to avoid such a current client–server model for firmware distribution in an IoT environment.

With this in mind, in this paper, we propose a new firmware update scheme that utilizes a blockchain technology. In the proposed scheme, an embedded device requests its firmware update to blockchain nodes on a peer-to-peer decentralised network. It then receives a response from a blockchain node to determine whether its firmware is up-to-date or not. When the firmware is not latest, the embedded device requests a metadata file to download the latest firmware from a peer-to-peer firmware sharing network consisting of the blockchain nodes. Even in the case that the version of the firmware is up-to-date, the firmware integrity is checked via the blockchain nodes. Accordingly, the proposed scheme guarantees the correctness and recentness of the firmware on the embedded device in the IoT era.

This paper, which is an extended version of the paper published in Proc. of Qshine 2016 [4], is organized as follows. Section 2 provides related works. Section 3 presents the proposed scheme with the overall architecture and operation procedures. In Sect. 4, we discuss the proposed scheme's strengths and weaknesses. Section 5 concludes this paper.

2 Preliminaries

2.1 Remote firmware updates

The ability to securely update firmwares of deployed embedded devices over networks is one of essential features nowadays. Vendors use the remote firmware update to provide new functionalities and also to patch vulnerabilities on the embedded devices.

In general, a vendor maintains its own firmware repository that contains pre-compiled binaries of the firmware for embedded devices [3,5]. When requested, the firmware file is downloaded to the embedded device.

For securing the remote firmware update, asymmetric cryptographic algorithms such as ECC or RSA are normally used. For instance, to provide integrity and authentication of the firmware, the message digest over the original firmware file is calculated with a hash function and then the message digest is digitally signed using a private key. The generated digital signature is attached to the firmware file with the corresponding public key of the private key. After downloading the firmware file from the repository, a security check process is performed before the actual firmware update, i.e., the digital signature is validated only with the attached public key. Once the security check process is successfully done, the actual firmware update is started on the target embedded device.

The current model for firmware distribution from the repository to the embedded device adopts the client–server model that means excessive network traffic may occur when requesting the firmware update files from embedded devices simultaneously. When we consider the IoT environment that tens of millions of the embedded devices are possibly required to be updated simultaneously, this client–server model is inappropriate. In addition, the asymmetric cryptographic algorithm, which is used for message signing, requires a complex key management and protection, e.g., private keys must be well securely used and maintained for each firmware version.

2.2 Blockchain

The blockchain was first introduced in 2009 by an anonymous person, Nakamoto Satoshi [6]. It was first used as a public ledger to provide trust transactions without an involvement of the third party for Bitcoin, which is a digital currency.

In the blockchain, a block is used to securely store information. Every block contains a hash value of the previous block header that forms a type of chain [7]. It is then used to authenticate the data and guarantee the block's integrity. The structure of a block, for instance in Bitcoin, consists of the block header and the block body. The block header is composed of the block size, version, previous block header's hash, merkle tree root, etc. The block body is composed of the merkle tree and transaction. A merkle tree is a tree in which the leaf nodes contain the hashes of blocks and the internal nodes contain the hashes of their children so that it enables efficient and secure verification of the blocks [8]. For instance, a blockchain node can quickly verify if the block it receives from other blockchain nodes have not been tampered with or the block is not corrupted during the transmission. In Bitcoin, a transaction is broadcasted to a blockchain network and is collected into blocks.

The blockchain relies on mainly two cryptographic methods: digital signature and cryptographic hash function. A digital signature is a way for demonstrating the authenticity of a digital message. It can be used to provide integrity and authentication of data as well as non-repudiation. A sender signs a message using the sender's private key. After a receiver receives this message, it verifies the message using the sender's public key. This message can be verified by anyone holding the valid public key of the

sender [9]. A cryptographic hash function is a mathematical operation that computes a hash value over a given input. The function is deterministic, i.e., the same input will always produce the same output, with the following properties: pre-image resistance, second pre-image resistance, and collision resistance. In the blockchain for Bitcoin, a SHA-256 hash function is used [10].

3 Proposed secure firmware update

The proposed secure firmware update scheme enables that an embedded device requests its firmware update to blockchain nodes and receives a response from a blockchain node to determine whether the device's firmware is up-to-date or not. If the firmware is not up-to-date, then a metadata file with a peer list to download the latest firmware is provided to the device so that the embedded device can keep the latest firmware by downloading the latest firmware from a peer-to-peer firmware sharing network consists of the blockchain nodes. Note that the peer-to-peer firmware sharing network can be implemented by BitTorrent [11]. If the firmware is up-to-date, then the firmware integrity is checked via the blockchain nodes. Note that the embedded device is a peer involved in the blockchain network and the firmware sharing network as well. Notations used in this paper are shown in Table 1.

3.1 Overview

Figure 1 depicts the overall architecture of the proposed scheme. The following entities are defined:

- *Blockchain node* A blockchain node is a node in a blockchain network of the proposed scheme. A set of blockchain nodes is denoted as $B = \{b_1, b_2, \ldots, b_n\}$ and $b_i \in B$.
- *Normal node* A normal node is an embedded device, which updates its firmware, in a blockchain network. A set of normal nodes is denoted as $N = \{n_1, n_2, \ldots, n_n\}$, $n_i \in N$, and $N \subset B$. It can be a request node or a response node. If a node requests its firmware update, the node becomes a request node. When a request node sends a request message to update its firmware, other normal nodes become response nodes from the perspective of the request node. A normal node's ID is a random unique value, which is generated when it starts its firmware update request based on its public key.
- *Verification node* A verification node is a node, which maintains firmware information such as verifiers, files of firmwares per model in its database. The verification node is operated by a vendor of firmwares so that relevant files for latest firmware updates can be provided to the verification node from the vendor node via a secure channel establish between them. A set of verification nodes is denoted as $V = \{v_1, v_2, \ldots, v_n\}$, $v_i \in V$, and $V \subset B$. A verification node's public key is available to all nodes in the blockchain network.

Table 1 Notations

Notation	Definition
N	A set of normal nodes
V	A set of verification nodes
B	A set of blockchain nodes
D	A model name of n_i
r	Random number
ts	Timestamp
v	Current firmware version of n_i
v_{new}	Latest firmware version of n_i
v'	Current firmware version of n_j
fv	Current firmware file of v
fv'	Current firmware file of v'
fv_{new}	Latest firmware file of v_{new}
$H(fv)$	Verifier generated with fv
$H(fv')$	Verifier generated with fv'
$H(fv_{new})$	Verifier generated with fv_{new}
Lv_{new}	Peer list shared with fv_{new}
Mv_{new}	Metadata file of fv_{new}
IDn_i	Identifier of n_i
IDv_i	Identifier of v_i
E	Elliptic curve
P	Base point of elliptic curve E
PUn_i	Public key of n_i
PRn_i	Private key of n_i
PUv_i	Public key of v_i
PRv_i	Private key of v_i
$Sign_i$	Signature of n_i
$Sigv_i$	Signature of v_i

- *Vendor node* A vendor node is outside of a blockchain network but keeps a secure channel with a verification node to provide the relevant files for latest firmware updates.

All blockchain nodes are either peers or trackers in a firmware sharing network, i.e., normal nodes are peers, whereas verification nodes are trackers. A verification node, which is a BitTorrent tracker, keeps track of where firmware copies reside on peers, which firmware copies are available, and helps coordinate efficient transmission and reassembly of the copied firmware files [12]. When a normal node, more specifically a request node, downloads the latest firmware file, it requires a metadata file and a peer list that are provided from the verification node. Note that the metadata file contains the filename, hash value of the file, piece length, tracker URL, etc., whereas the peer list is composed of a hash value of the file and IP addresses of peers that have a complete firmware file or an incomplete firmware file.

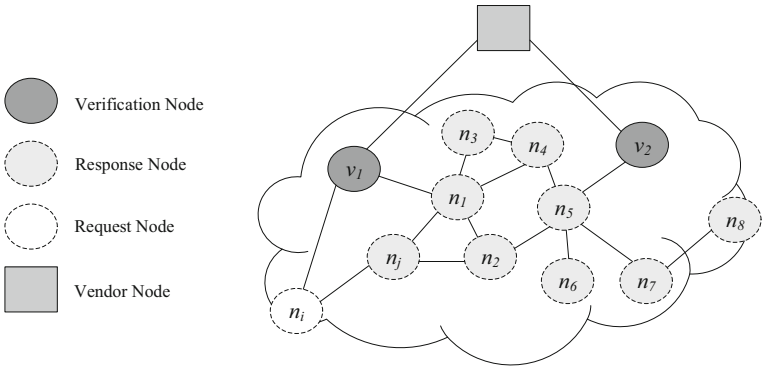


Fig. 1 Overall architecture

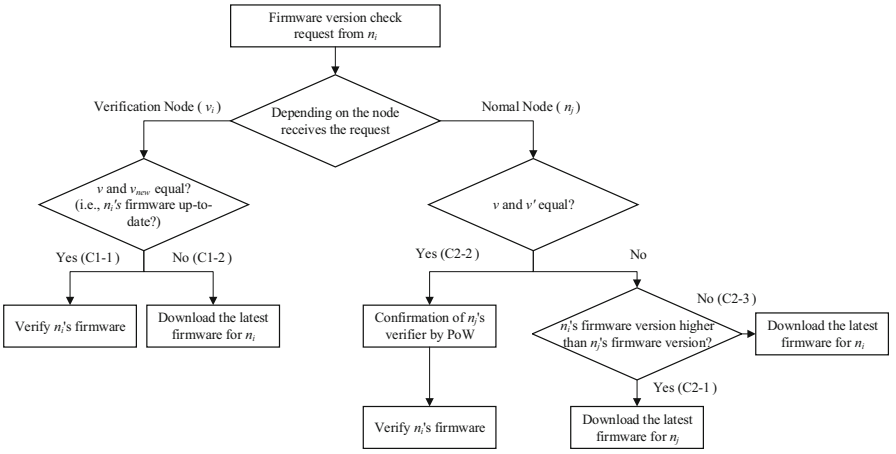


Fig. 2 Overall procedure

Figure 2 shows the overall procedure of the proposed scheme. When an embedded device, which is a normal node in a blockchain network, starts its firmware update request by broadcasting a version-check request message, it becomes a request node in the proposed scheme. Once the version-check request message is broadcasted in the blockchain network, other normal nodes, i.e., response nodes, and verification nodes respond to the version-check request message, respectively. Depending on the node type, the response procedure is different as defined in the following cases:

- C1: Response from a verification node v_i to a request node n_i
- C2: Response from a response node n_j to a request node n_i

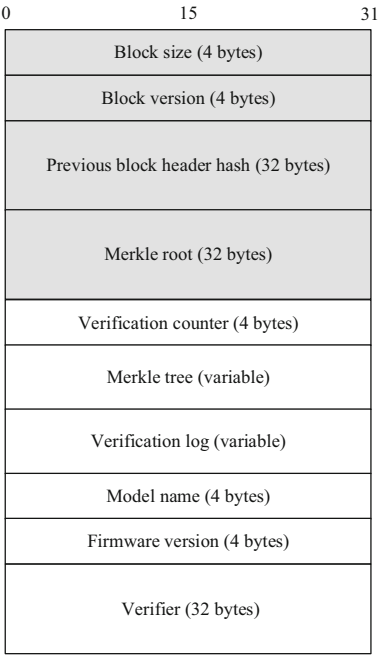
In C1, a verification node checks whether a request node has the latest version of the firmware or not. If the request node has the latest version, the verification node further checks the integrity of the request node's firmware. Otherwise, the request node's firmware is needed to be updated via the firmware sharing network.

In C2, a response node compares its firmware version with the firmware version of the request node. If the firmware version of the response node is same with that of the request node, the response node demands other nodes in the blockchain network to verify its firmware file's hash value, which is called a verifier that will be explained later. If the verifier of the response node is confirmed by other nodes that will be done via a six confirmation, i.e., proof-of-work (PoW), of the blockchain, the response node believes that its firmware is correct. Then, the response node can check whether the request node's firmware has been altered by comparing each other's verifiers. If the firmware version of the response node is different with that of the request node, the response node further checks whether the firmware version of the request node is higher than its firmware version. If the firmware version of the request node is higher, the response node's firmware is needed to be updated via the firmware sharing network. Otherwise, the request node's firmware is needed to be updated via the firmware sharing network.

3.2 Block structure

The proposed scheme uses a different block structure compared with the blockchain of Bitcoin. The block in the proposed scheme is made up of the block header and verification field as shown in Fig. 3. The block header is composed of the block size, version, previous block header hash, and merkle root. The verification field consists of the verification counter, merkle tree, verification log, model name, firmware version, and verifier.

Fig. 3 Block structure



Details of the verification field are given below.

- *Verification counter* This is the number of successful verifications. It is a value which is only considered in a normal node's block. It is similar with the block height in the blockchain of Bitcoin [13]. In a verification node, this value is fixed at 0.
- *Merkle tree* It is a tree information for the calculation of the merkle root. It is used for the verification of block data. It is also used as a feature for managing memory of node efficiently [6].
- *Verification log* It is a verification log composed of the verification time (e.g., timestamp), request node's ID, and response node's ID. If a verification node responses for the request message, the request node's ID and verification node's ID are stored to this field. To provide integrity and authentication of the verification log, the digital signature over the verification log is also included at the end of the log.
- *Model name* It is a normal node's model name.
- *Firmware version* It is a normal node's current firmware version.
- *Verifier* It is a hash value of a firmware file. This value is used to verify the firmware integrity without comparing genuine files. If the firmware file is composed of more than one file, then those should be concatenated before generating the verifier.

3.3 Procedure

As mentioned, the proposed scheme has the two different operation cases: C1 and C2. The case C1 is then divided into two sub-cases (C1-1 and C1-2), while the case C2 is also similarly separated into C2-1, C2-2, and C2-3. The cases depends on the type of response node, which receives the version-check request message sent from the request node.

3.3.1 C1: response from a verification node v_i to a request node n_i

When n_i transmits a version-check request message to a blockchain network and a verification node responds to the request, C1 starts. C1 has two different sub-cases.

- *C1-1* It starts when n_i has the latest firmware.
- *C1-2* It starts when n_i does not have the latest firmware.

The procedure of C1-1 shown in Fig. 4 is as follows.

1. A request node n_i broadcasts a *req_verchk* message including PUn_i, IDn_i, D, v, and r to a blockchain network. Note that r is used for preventing a replay attack.
2. When a verification node v_i receives the message, it obtains the latest firmware version of n_i from its database based on IDn_i, D, and v.
3. v_i responds by sending a *res_verchk* message including v_{new}, $r+1$, and IDv_i that has been encrypted with PUn_i.
4. When n_i receives the message, it decrypts the message with PRn_i and checks $r+1$. If v and v_{new} are equal, i.e., n_i has the latest firmware, n_i generates its verifier $H(fv)$. Otherwise, the process is terminated with an error.

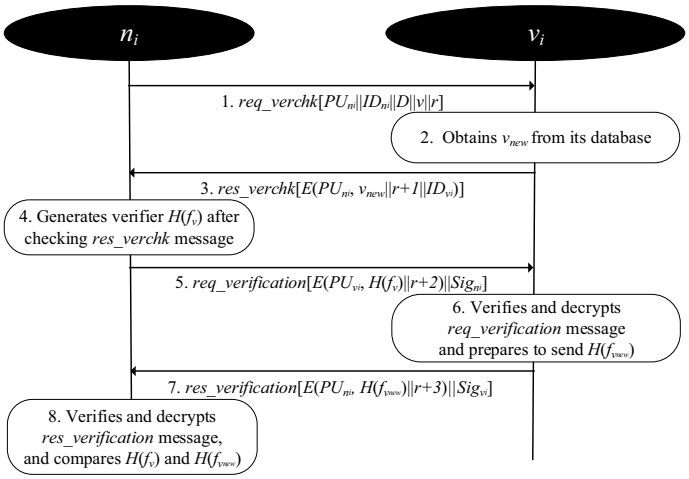


Fig. 4 Procedure of C1-1: n_i has the latest firmware

5. n_i first encrypts $H(fv)$ and $r+2$ with PUv_i and generates a digital signature $Sign_i$ over a *req_verification* message containing the encrypted $H(fv)$ and $r+2$. Then, $Sign_i$ is attached in the *req_verification* message, when n_i sends the message to v_i.
6. When v_i receives the message, it first verifies the message's integrity and originator with $Sign_i$ and PUn_i obtained from the *req_verchk* message. Then, it decrypts the message with PRv_i and checks $r+2$.
7. v_i first encrypts $H(fv_{new})$ and $r+3$ with PUn_i and generates a digital signature $Sigv_i$ over a *res_verification* message containing the encrypted $H(fv_{new})$ and $r+3$. Then, $Sigv_i$ is attached in the *res_verification* message, when v_i sends the message to n_i.
8. When n_i receives the message, it first verifies the message's integrity and originator with $Sigv_i$ and PUv_i previously shared. Then, it decrypts the message with PRn_i and checks $r+3$. Now, n_i compares $H(fv)$ and $H(fv_{new})$ to verify n_i's firmware integrity. If the verifiers are equal, the process is successfully finished. Otherwise, n_i needs to download the latest firmware via a peer-to-peer firmware sharing network similar to C1-2.

The procedure of C1-2 shown in Fig. 5 is as follows.

1. A request node n_i broadcasts a *req_verchk* message including PUn_i, IDn_i, D, v, and r to a blockchain network.
2. When a verification node v_i receives the message, it obtains the latest firmware version of n_i from its database based on IDn_i, D, and v.
3. v_i responds by sending a *res_verchk* message including v_{new}, Mv_{new}, $r+1$, and IDv_i that has been encrypted with PUn_i.
4. When n_i receives the message, it decrypts the message with PRn_i and checks $r+1$. n_i then obtains $H(fv_{new})$ from Mv_{new}.

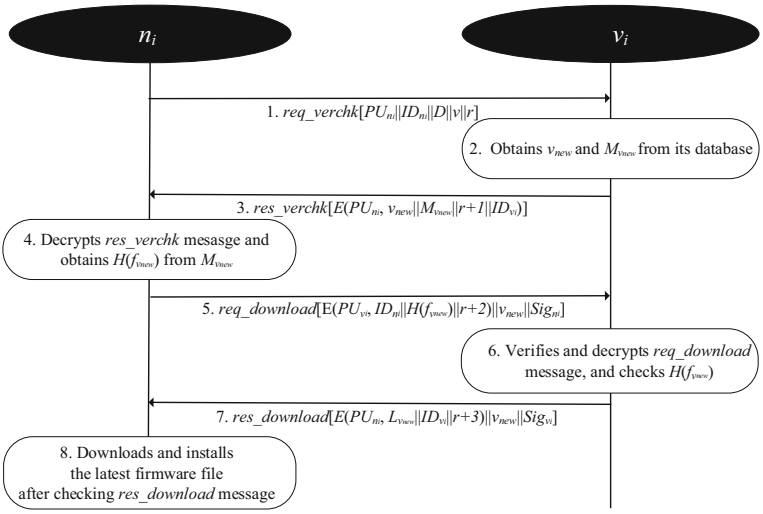


Fig. 5 Procedure of C1-2: n_i does not have the latest firmware

5. n_i first encrypts IDn_i, $H(fv_{new})$ and $r+2$ with PUv_i and generates a digital signature $Sign_i$ over a *req_download* message containing the encrypted data such as IDn_i, $H(fv_{new})$, and $r+2$, and v_{new}. Then, $Sign_i$ is attached in the *req_download* message, when n_i sends the message to v_i.
6. When v_i receives the message, it first verifies the message's integrity and originator with $Sign_i$ and PUn_i obtained from the *req_verchk* message. Then, it decrypts the message with PRv_i and checks $r+2$. Now, v_i prepares Lv_{new} based on $H(fv_{new})$. Note that Lv_{new} is used by n_i to download the latest firmware file from a peer-to-peer firmware sharing network.
7. v_i first encrypts Lv_{new}, IDv_i, and $r+3$ with PUn_i and generates a digital signature $Sigv_i$ over a *res_download* message containing the encrypted data such as Lv_{new}, IDv_i, and $r+3$, and v_{new}. Then, $Sigv_i$ is attached in the *res_download* message, when v_i sends the message to n_i.
8. When n_i receives the message, it first verifies the message's integrity and originator with $Sigv_i$ and PUv_i previously shared. Then, it decrypts the message with PRn_i and checks $r+3$. Now, n_i is able to download the latest firmware file fv_{new} from the peer-to-peer firmware sharing network and then installs the file.

3.3.2 C2: response from a response node n_j to a request node n_i

In C2, a response for n_i's *req_verchk* message is performed by other normal node n_j, and as a result of comparison of n_i's firmware version and n_j's firmware version, C2 is divided into three different sub-cases.

Contrary to C1, a response node is a normal node so that the legitimacy of a verifier must be proofed during the firmware update procedure. Note that in C1, the legitimacy of a verifier is assumed because a verification node maintains the verifier provided from

the vender node. In C2-2, the legitimacy of a verifier is confirmed via the PoW as like in the blockchain of Bitcoin [6], whereas the legitimacy of a verifier does not need to be considered in C2-1 and C2-3 because there is no requirement of checking the verifier.

- *C2-1* It starts when n_i's firmware version is higher than n_j's firmware version.
- *C2-2* It starts when n_i's firmware version and n_j's firmware version are equal.
- *C2-3* It starts when n_i's firmware version is lower than n_j's firmware version.

The procedure of C2-1 shown in Fig. 6 is as follows.

1. A request node n_i broadcasts a *req_verchk* message including PUn_i, IDn_i, D, v, and r to a blockchain network.
2. When receiving the message, n_j compares v obtained from message and v' and confirms that n_i's firmware version is higher.
3. n_j responds by sending a *res_verchk* message containing the encrypted data such as v', $r+1$, and IDn_j. Note that PUn_i obtained from the *req_verchk* message is used for the encryption.
4. After decrypting the message with PRn_i, n_i compares v and v'. Then, it also confirms that its firmware version is higher than that of n_j.
5. n_i generates a digital signature $Sign_i$ over a *notice_download* message containing IDn_i, v', and $r+2$. Then, $Sign_i$ is attached in the *notice_download* message, when n_i sends the message to n_j.

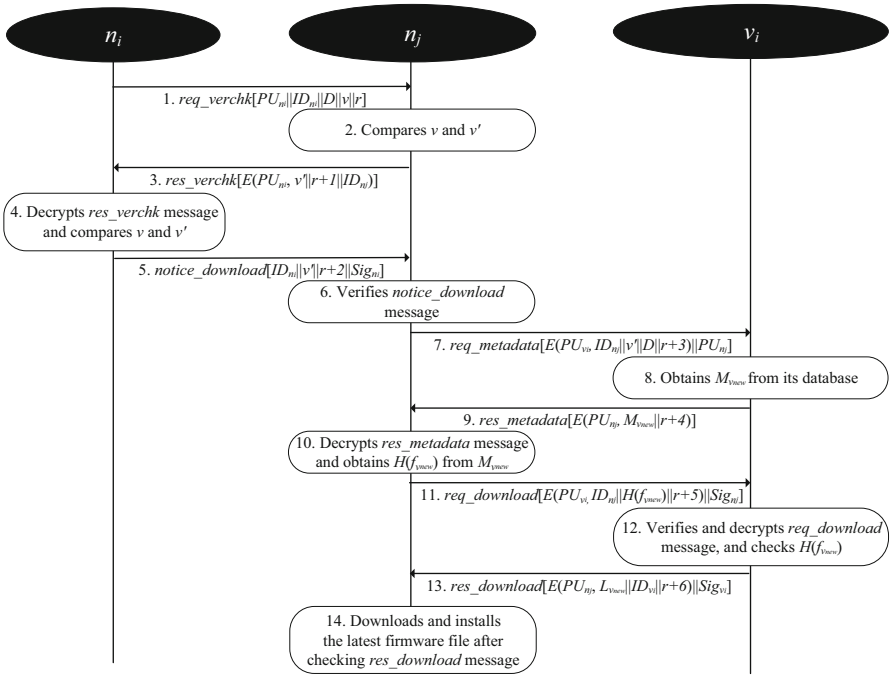


Fig. 6 Procedure of C2-1: n_i's firmware version is higher than n_j's one

6. When n_j receives the *notice_download* message, it verifies the message's integrity and originator with $Sign_i$ and PUn_i obtained from the *req_verchk* message.
7. Now n_j requests v_i a metadata by sending a *req_metadata* message. The message contains the encrypted data such as IDn_j, v', D, and $r+3$, and PUn_j.
8. When v_i receives the *req_metadata* message, it decrypts the message. Based on the obtained data such as IDn_j, v', and D, v_i finds Mv_{new} for n_j.
9. v_i now provides Mv_{new} by sending a *res_metadata* message to n_j. The message contains the encrypted Mv_{new} and $r+4$ with PUn_j.
10. n_j acquires $H(fv_{new})$ from Mv_{new} after decrypting the *res_metadata* message with PRn_j.
11. n_j sends a *req_download* message to request a peer list by sending a *req_download* message. For sending the message, it first encrypts IDn_j, $H(fv_{new})$, and $r+5$ with PUv_i and generates a digital signature $Sign_j$ over the *req_download* message containing the encrypted data. $Sign_j$ is attached in the *req_download* message, when n_j sends the message to v_i.
12. When v_i receives the message, it first verifies the message's integrity and originator with $Sign_j$ and PUn_j obtained from the *req_metadata* message. Then, it decrypts the message with PRv_i and checks $r+5$. Now, v_i prepares Lv_{new} based on $H(fv_{new})$. Note that Lv_{new} is used by n_j to download the latest firmware file from a peer-to-peer firmware sharing network.
13. v_i first encrypts Lv_{new}, IDv_i, and $r+6$ with PUn_j and generates a digital signature $Sigv_i$ over a *res_download* message containing the encrypted data. Then, $Sigv_i$ is attached in the *res_download* message, when v_i sends the message to n_j.
14. When n_j receives the message, it first verifies the message's integrity and originator with $Sigv_i$ and PUv_i previously shared. Then, it decrypts the message with PRn_j and checks $r+6$. Now, n_j is able to download the latest firmware file fv_{new} from the peer-to-peer firmware sharing network and then installs the file.

The procedure of C2-2 shown in Fig. 7 is as follows.

1. A request node n_i broadcasts a *req_verchk* message including PUn_i, IDn_i, D, v, and r to a blockchain network.
2. When a response node n_j receives the message, it compares v and v'. If v and v' are equal, n_j generates its verifier $H(fv')$.
3. n_j broadcasts a *join_verification* message including IDn_i, IDn_j, D, and $H(fv')$ to a blockchain network for asking other nodes to join the verification process, and other nodes perform the PoW stage. After completing the PoW, they add IDn_i as request node's ID, IDn_j as response node's ID, ts into the verification log and then broadcast the verification log into the blockchain network.
4. If n_j receives the verification log from more than six other nodes, n_j responds by sending a *res_verchk* message including v', $r+1$, IDn_j, and PUn_j that has been encrypted with PUn_i.

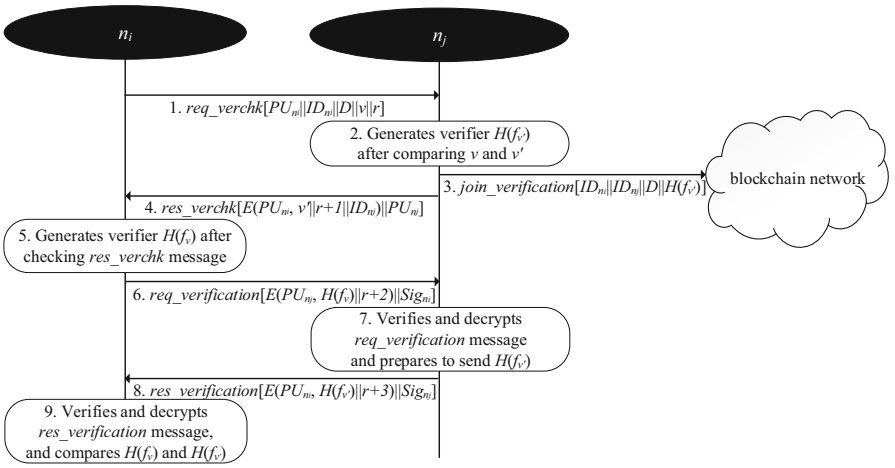


Fig. 7 Procedure of C2-2: n_i's firmware version is equal with n_j's one

5. When n_i receives the message, it decrypts the message with PRn_i and check $r+1$. If v and v' are equal, n_i generates verifier $H(fv)$. Otherwise, the process is terminated with an error.
6. n_i first encrypts $H(fv)$ and $r+2$ with PUn_j and generates a digital signature $Sign_i$ over a *req_verification* message containing the encrypted $H(fv)$ and $r+2$. Then, $Sign_i$ is attached in the *req_verification* message, when n_i sends the message to n_j.
7. When n_j receives the message, it first verifies the message's integrity and originator with $Sign_i$ and PUn_j obtained from the *res_verchk* message. Then, it decrypts the message with PRn_j and checks $r+2$. As a result of checking the value, n_j prepares to send $H(fv')$. Otherwise, the process is terminated with an error.
8. n_j first encrypts $H(fv')$ and $r+3$ with PUn_i and generates a digital signature $Sign_j$ over a *res_verification* message containing the encrypted $H(fv')$ and $r+3$. Then, $Sign_j$ is attached in the *res_verification* message, when n_j sends the message to n_i.
9. When n_i receives the message, it first verifies the message's integrity and originator with $Sign_j$ and PUn_j previously shared. Then, it decrypts the message with PRn_j and checks $r+3$. Now, n_i compares $H(fv)$ and $H(fv')$ to verify n_i's firmware integrity. If the verifiers are equal, the process is successfully finished. Otherwise, n_i needs to download the latest firmware via a peer-to-peer firmware sharing network similar to C2-1 or C2-3.

The procedure of C2-3 shown in Fig. 8 is as follows.

1. A request node n_i broadcasts a *req_verchk* message including PUn_i, IDn_i, D, v, and r to a blockchain network.
2. When receiving the message, n_j compares v obtained from message and v' and confirms that n_j's firmware version is higher.

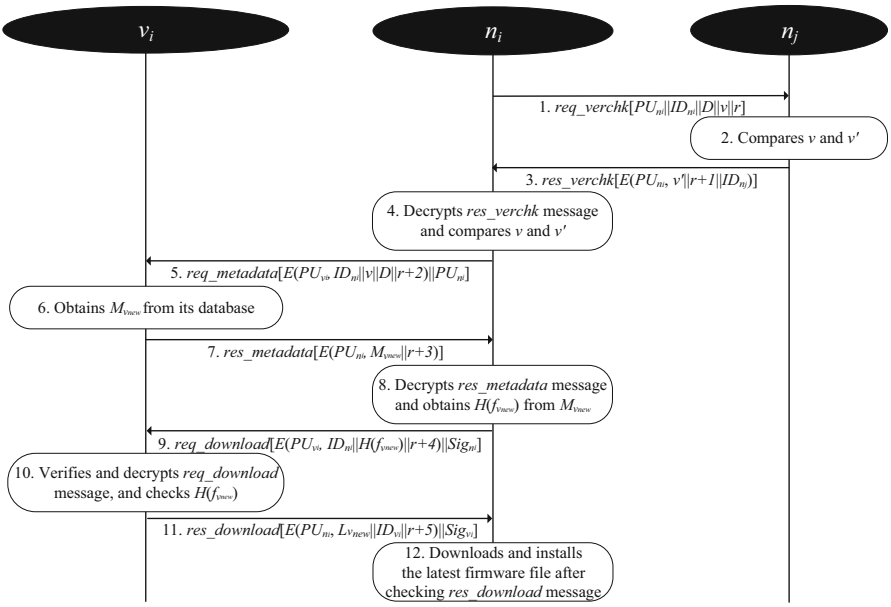


Fig. 8 Procedure of C2-3: n_i's firmware version is lower than n_j's one

3. n_j responds by sending a *res_verchk* message containing the encrypted data such as v', $r + 1$, and IDn_j. Note that PUn_i obtained from the *req_verchk* message is used for the encryption.
4. After decrypting the message with PRn_i, n_i compares v and v'. Then, it also confirms that its firmware version is lower than that of n_j.
5. Now n_i requests v_i a metadata by sending a *req_metadata* message. The message contains the encrypted data such as IDn_i, v, D, and $r + 2$, and PUn_i.
6. When v_i receives the *req_metadata* message, it decrypts the message. Based on the obtained data such as IDn_i, v, and D, v_i finds Mv_{new} for n_i.
7. v_i now provides Mv_{new} by sending a *res_metadata* message to n_i. The message contains the encrypted Mv_{new} and $r + 3$ with PUn_i.
8. n_i acquires $H(fv_{new})$ from Mv_{new} after decrypting the *res_metadata* message with PRn_i.
9. n_i sends a *req_download* message to request a peer list by sending a *req_download* message. For sending the message, it first encrypts IDn_i, $H(fv_{new})$, and $r + 4$ with PUv_i and generates a digital signature $Sign_i$ over the *req_download* message containing the encrypted data. $Sign_i$ is attached in the *req_download* message, when n_i sends the message to v_i.
10. When v_i receives the message, it first verifies the message's integrity and originator with $Sign_i$ and PUn_i obtained from the *req_metadata* message. Then, it decrypts the message with PRv_i and checks $r+4$. Now, v_i prepares Lv_{new} based on $H(fv_{new})$. Note that Lv_{new} is used by n_i to download the latest firmware file from a peer-to-peer firmware sharing network.

11. v_i first encrypts Lv_{new}, IDv_i, and $r + 5$ with PUn_i and generates a digital signature $Sigv_i$ over a *res_download* message containing the encrypted data. Then, $Sigv_i$ is attached in the *res_download* message, when v_i sends the message to n_i.
12. When n_i receives the message, it first verifies the message's integrity and originator with $Sigv_i$ and PUv_i previously shared. Then, it decrypts the message with PRn_i and checks $r+5$. Now, n_i is able to download the latest firmware file fv_{new} from the peer-to-peer firmware sharing network and then installs the file.

After the end of firmware verification, the request node makes its verification log including request node's ID, response node's ID, timestamp, etc. The request node then broadcasts it to the blockchain network. Consequentially, it enables to validate the firmware integrity and decide whether the update firmware on the device requires or not without a user's control.

4 Discussion

The proposed scheme focuses on minimizing the attack window time by enabling an embedded device can quickly check its firmware version and download the latest firmware if needed. It thus helps to shorten effects of attacks targeting known firmware vulnerabilities on the embedded device. As this scheme has been developed for the IoT environment, all the procedures such as firmware validation and download to update are based on distributed models. In other words, a blockchain technology has been adopted for secure firmware version check and validation, while BitTorrent has been used for a peer-to-peer firmware sharing network for downloading the latest firmware. However, a verification node acts as a tracker in the peer-to-peer firmware sharing network. The verification node is thus considered as a server assisting in the communication between peers for firmware downloading. The proposed scheme can be improved by adopting distributed hash tables, i.e., implementing a trackerless BitTorrent system.

In the proposed scheme, a verification node's public key is assumed to be pre-shared to all normal nodes. As the number of the verification nodes increases, the key distribution and management cost increases. However, in practice, the number of the verification nodes would not be many.

The proposed scheme enables that an embedded device keeps its firmware up-to-date while providing its integrity, i.e., correctness of firmware. However, it does not mean that the provided firmware is not buggy nor vulnerable, e.g., stack overflow [14]. In addition, the proposed scheme does not guarantee that involved nodes work correctly. We have assumed that all nodes work as defined in the proposed scheme. However, in practice, some nodes can be compromised by an attack's physical access and firmware modification attack.

In the proposed scheme, a request node starts its firmware update request by sending the version-check request message, which is a broadcast message. Then, depending on the node type receives the broadcast message, the proposed scheme has 2 different operation cases such as C1 and C2. However, due to the nature of the broadcast message, the proposed scheme may cause unnecessary network traffic and nodes'

operations. For instance, even if the firmware verification is done by a verification node (i.e., via C1-1), a response node may perform the confirmation of its verifier through the PoW (i.e., via C2-2) that consumes network traffic and other normal nodes' computational power.

The proposed scheme's security relies on the Blockchain technology and some basic properties of asymmetric cryptography like message signature and encryption. However, the proposed scheme's security has been not formally verified in this paper that can be done for instance through the BAN logic and Scyther tool [15].

5 Conclusion

In this paper, we presented a new firmware update scheme that securely checks a firmware version, validates the correctness of firmware, and enables downloading of the latest firmware for embedded devices in an IoT environment. The proposed scheme relies on a blockchain technology for firmware checking and validation, while BitTorrent can be used to implement a peer-to-peer firmware sharing network for firmware downloading.

The architecture and detailed operation procedures of the proposed scheme have been illustrated. In addition, we have discussed the proposed scheme's strengths and weaknesses. As mentioned, there are some considerations and limitations of the current scheme so that we will improve the proposed scheme to eliminate the found limitations and improve security and scalability.

References

1. Gartner (2014) Gartner says 4.9 billion connected things will be in use in 2015, Gartner Newsroom
2. Firmware—Wikipedia. https://en.wikipedia.org/wiki/Firmware
3. Choi B-C, Lee S-H, Na J-C, Lee J-H (2016) Secure firmware validation and update for consumer devices in home networking. IEEE Trans Consum Electron 62(1):39–44
4. Lee B, Malik S, Wi S, Lee J-H (2016) Firmware verification of embedded devices based on a blockchain. In: Proceedings of the Qshine 2016
5. Jurkovi G, Sruk V (May 2014) Remote firmware update for constrained embedded systems. In: Proceedings of the MIPRO 2014
6. Nakamoto S (2009) Bitcoin: a peer-to-peer electronic cash system
7. Blockchain Bitcoin Wiki. https://en.bitcoin.it/wiki/Block_chain
8. Hu Y, Perrig A, Johnson DB (2003) Efficient security mechanisms for routing protocols. In: Proc. NDSS03
9. Badev A, Chen M (2013) Bitcoin: technical background and data analysis. Federal Reserve Board
10. Bider D, Baushke M (2012) SHA-2 data integrity for the secure shell (SSH) transport layer protocol. IETF RFC 6668
11. Cohen B (2003) Incentives build robustness in bitorrent. In: Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems
12. Wiki Theory—Bittorrent Protocol Specification v1.0. https://wiki.theory.org/BitTorrentSpecification
13. Antonopoulos, Andreas M (2014) Mastering Bitcoin: unlocking digital crypto-currencies. OReilly Media
14. Alouneh S, Bsoul H, Kharbutli M (2016) A software tool to protect executable files from buffer overflow attacks. Int J Internet Technol Secur Trans 6(2):133–166
15. Ahamad S, Al-Shourbaji I, Al-Janabi S (2016) A secure NFC mobile payment protocol based on biometrics with formal verification. Int J Internet Technol Secur Trans 6(2):103–132