OTA Updates Mechanisms: A Taxonomy and Techniques Catalog

Mónica M. Villegas[1] and Hernán Astudillo[2]

[1] Universidad Técnica Federico Santa María, Valparaíso, Chile
monica.villegas@sansano.usm.cl
[2] Universidad Técnica Federico Santa María, Santiago, Chile
Centro Científico y Tecnológico de Valparaíso (CCTVal)
hernan@inf.utfsm.cl

Abstract. The use of the Internet of Things (IoT) and Cyber-Physical Systems (CPS) in industry and daily life has increased. The embedded software of IoT systems requires updates over time for long-term maintainability, bug fixes, and improvements. Developers and manufacturers design and implement OTA update systems in ad-hoc manners because there are no specific standards and little empirical information about mechanisms. This article describes a systematic literature review to identify proposed OTA update mechanisms, and a taxonomy to organize them for system designers. Academic and professional (grey) literature was gathered from four information sources; 109 studies were found, of which 29 remained after applying inclusion and exclusion criteria; and they were recognized as belonging to six mechanisms (categories). Each technique was associated to a mechanism, yielding an (initial) catalog of OTA update techniques. This taxonomy and catalog can be used to design IoT and CPS applications that must include OTA update functionality.

Keywords: Over-the-Air (OTA) updates, Cyber-Physical Systems (CPS), Internet-of-Things (IoT), Software design

1 Introduction

Software updates are an essential part of almost any computer-based system. There exist different types of strategies for implementing these updates regarding the methods and practices used for delivering software update packages to the final systems. One of these practices is Over-the-Air (OTA) updates, which refers to remotely updating the code on an embedded device[3] and distribute data from a server to clients or nodes in a network channel [12]. OTA updates are widely used in several domains, like the Internet of Things (IoT) and Cyber-Physical Systems (CPS).

For example, Firmware Over-the-Air (FOTA) is an actual technology to perform an update by wirelessly transmitting firmware update data to the target

[3] https://docs.particle.io/tutorials/device-cloud/ota-updates

devices [14]. Through FOTA, applications, operating systems (OS), and in some cases, the full software stack [15], can be replaced.

The use of CPS and IoT in fields of industry and daily life, like automotive, energy, building, and agriculture[4] keep increasing. Depending on the robustness and complexity in the systems designed, OTA updates will be needed eventually for updating a large number of devices connected in a network where manual updates will no be easy to perform (or maybe not feasible at all). CPS and IoT systems' embedded software requires updates over time for long-term maintainability, bug fixes, and security improvement [38], [8] to avoid attacks as specified and studied in [9, 10]. As technologies evolve, these OTA update systems depend heavily on specifics of the system and platform technologies. The existence of several hardware platforms and operating systems for IoT keeps growing, as described in previous work [7]. Thus, many developers and manufacturers design and implement OTA update systems in ad-hoc manners since there are no specific standards, including the complete design of such systems, and little empirical information about mechanisms and generic dependencies is formally reported. Our proposal is based on the analysis of recommendations reported by recognized institutions and technology experts, vs. the techniques published in the literature, in the context of OTA update systems designs for CPS and IoT systems.

In previous work [11], we highlight the importance of taxonomies to organize architectural elements that can be used to support software architects in their work and will proceed likewise in this study. To this end, a *mechanism* is a software or hardware component of an OTA update system; a *recommendation* is a set of steps or mechanisms described by recognized institutions or technology experts, which detail how OTA update systems should be designed; and a *technique* is a specific approach or procedure implemented by researchers, engineers, or practitioners, and formally published as a document.

The research questions conducting our study were:

- **RQ0**: What mechanisms, techniques, or recommendations are reported in OTA update systems for IoT and CPS?
- **RQ1**: How can these mechanisms, techniques, and recommendations be classified into a taxonomy?

The remainder of this article is structured as follows: Section 2 briefly surveys previous work on software updates taxonomies; Section 3 addresses RQ0, describing a systematic review design and execution; Section 4 addresses RQ1, proposing a taxonomy to organize OTA update mechanisms, techniques, and recommendations; and Section 5 summarizes and concludes.

2 Related Work

Some taxonomies have already been proposed to organize OTA update techniques.

[4] https://blog.bosch-si.com/bosch-iot-suite/how-to-manage-software-updates-in-iot

Brown and Sreenan [17] surveyed software update features of wireless sensor networks and presented a taxonomy organized around ideas or design decisions, with five categories: (1) Update Dissemination, (2) Update Activation, (3) Fault Detection, (4) Fault Recovery, and (5) Management.

Halder et al. [18] surveyed remote OTA software updates in the automotive sector, mainly from the security perspective. They proposed a taxonomy of secure OTA software update techniques, with seven categories: (1) Symmetric Key, (2) Hash Function, (3) Blockchain, (4) RSA and Steganography, (5) Symmetric Key and Asymmetric Key, (6) Secure Hardware Module, and (7) Secure Update Framework.

The U.S. NTIA (National Telecommunications and Information Administration[5], through its IoT Security Upgradability Existing Standards, Tools and Initiatives Working Group, drafted a catalog [21] of existing IoT security standards. This catalog includes The Update Framework (TUF) [22][6], which helps developers to secure new or existing software update systems.

These taxonomies have some shortcomings. Halder et al.'s taxonomy [18] focuses on the security of the OTA update systems, leaving aside quality attributes like safety, security, recovery, management, dissemination, propagation, installation, scheduling, elaboration, and packaging. Also, neither taxonomy [18] [17] gives specific techniques or procedures.

To address these shortcomings, we will analyze the whole OTA update process, recognize mechanisms, and report specific techniques and procedures, giving a finer-grained classification of existing OTA update solutions.

3 RQ0: Existing OTA update approaches for IoT and CPS

A systematic review of academic and grey (professional) literature was performed following the well-known guidelines by Kitcheham et al. [13].

3.1 Research Protocol

Search Strategy: The search strategy in this study was divided in two sections:

- **S1:** Search in databases
- **S2:** Search of grey literature using the Google Search engine

Works published in databases were considered for the extraction of relevant information reported by researchers and practitioners, in which techniques were already implemented in working OTA update systems. Data reported in grey literature was considered since recommendations are regarded as sets of steps for designing a complete OTA update system, grouping the relevant mechanisms an OTA update system must-have. The results obtained were analyzed and used to map the recommendations grouped into mechanisms with the reported techniques. The search queries used were the following:

[5] National Telecommunications and Information Administration: www.ntia.gov/

[6] The Update Framework: www.theupdateframework.io

- **Databases:** IEEE Xplore, ACM Digital Library, Science Direct
- **Search string for Databases:** "(OTA OR over-the-air) AND (update) AND ((IoT OR (internet AND things)) OR (CPS OR (cyber AND physical AND systems)))"
- **Search string for grey literature using Google Search engine:** "standard mechanisms techniques recommendations OTA updates systems IoT CPS"

Inclusion/exclusion criteria: To select relevant material, we defined exclusion and inclusion criteria.

- Exclusion Criteria:
  - **EC1.** Texts not written in English or Spanish
  - **EC4.** Not fully available texts
  - **EC2.** Patents
  - **EC5.** Not accessible texts
  - **EC3.** Presentations
- Inclusion Criteria:
  - **IC1.** A primary study must contain software or hardware components used in the development of OTA update systems.
  - **IC2.** A primary study must contain recommendations for the design of OTA update systems.
  - **IC3.** A primary study must have software or hardware techniques or approaches for including on the OTA update process.
  - **IC4.** Grey literature can be included (e.g., tech blogs written by experts or reports by recognized institutions) if they contain software or hardware recommendations for the design and/or development of OTA update systems.

Selection Process: The selection of the works was based in three phases for the search in databases, and in two phases for the search of grey literature, which is explained as follows:

- **Search in databases:**
  - **Phase 1 (PH1.1):** Selection of works in which the title and abstract were related to designing OTA update systems for IoT and CPS
  - **Phase 2 (PH1.2):** Selection of works in which the introduction and proposal were related to mechanisms, techniques or processes in the context of OTA update systems for IoT and CPS
  - **Phase 3 (PH1.3):** Extraction of the techniques used in the OTA update systems reported for IoT and CPS
- **Search of grey literature:**
  - **Phase 1 (PH2.1):** Selection of works in which the title and description were related to recommendations for the design of OTA update systems for IoT and CPS, including sets of steps or mechanisms, and published by relevant institutions and tech blogs
  - **Phase 2 (PH2.2):** Extraction of recommendations used in the design of OTA update systems for IoT and CPS

3.2 Results from academic literature

Results from searching in databases and filtering through phases 1 (PH1.1) and 2 (PH1.2) are shown in Table 1.

Table 1. Results from databases search

Database	Search Results	PH1.1	PH1.2
IEEE Xplore	36	24	17
ACM Digital Library	35	5	3
Science Direct	3	3	1

After filtering through PH1.2, techniques, processes and design principles involved in the OTA update of IoT and CPS devices were identified and extracted through PH1.3 from selected works. Identified techniques in [19] were grouped in T1 group of techniques. From [14], were grouped in T2. From [15], in T3. From [16], in T4. From [25], in T5. From [26], in T6. From [27], in T7. From [28], in T8. From [31], in T9. From [33], T10. From [34], T11. From [35], T12. From [36], T13. From [38], T14. From [39], T15. From [40], T16. From [41], T17. From [29], T18. From [30], T19. From [37], T20. And from [32], T21. Such identified and extracted techniques (T groups) specified as follows, and will be further classified.

Identified and extracted techniques (T groups):

- **T1.1.** Execution of scripts at run-time
- **T1.2.** Injection of intermediate code used by virtual machines and interpreted at run-time
- **T1.3.** Perform full-image replacement of the entire firmware at once
- **T1.4.** Enable partial code updates of protocols and applications
- **T1.5.** Use indirect function calls for not requiring source code modifications
- **T1.6.** Maintain functions pointers required by indirect functions to scale the number of dynamic components
- **T1.7.** Decrease the dependencies with the system level defining system-level boundaries
- **T1.8.** Reduce the overhead by linking components to components rather than linking components to individual functions
- **T2.1** Use a server for transferring update data to a coordinator device
- **T2.2** Use a server for data verification based on the data HASH [42] for when a coordinator device delivers data to each node device in a network
- **T2.3** Use a coordinator device to disseminate update data to node devices in a WSN
- **T2.4** Use a coordinator device for sending updated data sequentially to each node device in a network and perform data dissemination
- **T2.5** Initiate the update process after the dissemination of update data is performed

- **T3.1** Generate a set of keys for authentication between node devices and server
- **T3.2** Register the devices in server to identify each device in a network
- **T3.3** Request and validate timestamp of update images
- **T3.4** Include metadata update information, consisting of the metadata body, and a signature
- **T3.5** Include root metadata update information, consisting of an ASN1[7] data structure containing public keys used for data signature
- **T3.6** Include timestamp metadata update information, containing files names, length and the hashes used for the verification of the integrity of the updates
- **T3.7** Include snapshot metadata update, informing about all targets metadata files released by the repository, indicating available update images
- **T3.8** Include targets metadata update, listing information about the filenames, files length, hashes and identifier of the devices of each available firmware images
- **T3.9** Inspect targets metadata to get valid Firmware Images
- **T3.10** Download a valid firmware image as binary data according to the filename of the targets metadata file
- **T3.11** Deliver firmware image, breaking it down into verifiable transmission blocks using a CRC[8] checksum
- **T3.12** Perform firmware update by using a bootloader to check if the storage of the device contains a firmware. If not, the system starts immediately, else the bootloader fully erases the flash memory and copies the new firmware from to the Flash memory
- **T3.13** Generate and submit a manifest to the Server informing the completion of the firmware upgrade
- **T4.1** Use TLS (Transport Layer Security) to authenticate
- **T4.2** Use DTLS (Datagram Transport Layer Security) to authenticate
- **T4.3** Use smart contracts to verify the firmware update authenticity and handle abnormal incidents during the update process
- **T4.4** Use Web-based blockchain transaction monitor or mobile phone app to check the firmware update status
- **T4.5** Use permissioned blockchain network that supports the verification process through smart contracts
- **T4.6** Use a vendor service to initiate a new blockchain transaction containing the target IoT device information and SHA1[9] hash of the new firmware update
- **T4.7** Use a vendor service for pushing the firmware update binary to the target IoT device
- **T4.8** After an IoT device receives the OTA firmware update binary from the vendor service, it computes the SHA1 hash of the received binary for validation

[7] ASN1: https://tools.ietf.org/html/rfc6025

[8] CRC: https://barrgroup.com/tech-talks/checksums-and-crcs

[9] SHA1: https://www.ssl.com/faqs/what-is-sha-1

- **T4.9** Validate the update through the IoT device which queries a transaction identifier while using smart contracts
- **T4.10** When validation of the update succeeds, the IoT device applies the firmware update and sends a status update to the blockchain. Otherwise, the OTA firmware update process will be aborted, and a failure notice will be recorded
- **T4.11** Vendor service queries the blockchain to collect update statistics to determine further actions
- **T5.1** Upload of the new update file to a gateway using persistent flash storage and Coffee File System
- **T5.2** After transmission of update file to a gateway is finished and verified, Deluge algorithm is started for distributing the file in the network in a broadcast fashion to end devices
- **T5.3** After a file is stored and verified on every device in a network, a gateway will be commanded to update the respective end devices
- **T6.1** Use ARM TrustZone to support secure booting and facilitate secure OTA updates using RSA encryption
- **T6.2** Use X-Cube SBSFU (Secure Boot and Secure Firmware Update) in which a device receives the encrypted binaries and security engine will manage operations and critical data
- **T6.3** Use Crypto-bootloader of MSP430, providing security against unauthorized firmware updates and IP encapsulation
- **T6.4** Use Bootcore which is a serial flash memory device used to load the code from external memory to embedded RAM (Random-Access Memory) or ROM (Read-Only Memory)
- **T6.5** Use Advanced Encryption Standard - Cipher Block Chaining (AES-CBC) and RSA, ciphers to secure firmware updates
- **T6.6** Use JTAG/SWD, a debug infrastructure that can be used as a lock/unlock mechanism
- **T7.1** Use Deluge for code dissemination, in which code image is divided into a set of fixed-size pages and further divided into packets. Packets are then transmitted by using a three-way handshake for reliability
- **T7.2** Use MNP (Multihop Network Reprogramming) for code dissemination. A multihop reprogramming service which avoids collision, and employs a sleep mechanism to save energy
- **T7.3** Use Rateless Deluge which reduces the need for packet retransmission
- **T7.4** Use Freshet [45] for code dissemination, which uses topology information to conserve energy
- **T7.5** Use MDeluge [46] for code dissemination, to support WSNs with mobile sensor nodes
- **T7.6** Use ReXOR [47] for code dissemination, which is a lightweight and density-aware protocol
- **T7.7** Use CoCo+ [48] for code dissemination optimizing the three-way handshake mechanism

- **T7.8** Use EECD (Excellence Estimation Code Dissemination) [43] for code dissemination using selection scheme based on link quality for transmission count in code dissemination
- **T7.9** Use ACDP (Adaptive Code Dissemination Protocol) for code dissemination to reduce communication cost and achieve load balance
- **T7.10** Use Dsare [44] for code dissemination, which addresses control message redundancy and energy balancing problems
- **T7.11** Stablish an optimal payload size in terms of energy efficiency
- **T7.12** Use simple and efficient link quality estimation scheme using LQI (Link Quality Indicator) in code dissemination
- **T7.13** Use code dissemination scheme on top of Deluge
- **T8.1** Use a Web-based dashboard solution for controlling firmware updates
- **T8.2** Establish an Admin type of user role in the update process using a Web-based dashboard to keep track of registered devices and versions of firmware available
- **T8.3** Establish a Firmware uploader/manufacturer type of user role in the update process using a Web-based dashboard, who will be able to add different versions of firmware for each device
- **T8.4** Establish a Firmware user role in the update process using a Web-based dashboard, in which new users must sign-up for being able to authenticate and access to the dashboard and register their IoT device details for firmware updates
- **T8.5** Use Device fingerprinting, a technique used to forensically identify an electronic device on the internet and avoid sending the firmware image to the wrong device
- **T9.1** Use encoded payload using a secure representation/format such as JOSE (JSON Object signing Encryption)
- **T9.2** The end node starts the FOTA image download
- **T9.3** In response of acknowledgment, the server sends a full encrypted firmware object
- **T9.4** Use a Payload containing the encrypted FOTA data
- **T9.5** Perform acknowledgment when required to every object in the network from the client-side to the server
- **T9.6** The server keeps track acknowledgments sent from client devices for resuming when connection breaks
- **T10.1** Use secure download of firmware from the server
- **T10.2** FOTA initializes a trusted application
- **T10.3** Use trusted application to calculate the checksum of file chunks
- **T10.4** Use checksum calculation of firmware File for verification
- **T10.5** FOTA reads signature of firmware file and provides it to a trusted application
- **T10.6** Use checksum and signature trusted application to verify the firmware
- **T10.7** Use Emmc/NAND flash drivers for writing firmware to flash
- **T10.8** Send firmware update status to FOTA application and reboot
- **T11.1** Use creation of the smart contracts

- **T11.2** Use direct distribution of firmware update, directly from the server of the manufacturer
- **T11.3** Use peer-to-peer firmware update, happening between devices of the same type and from the same manufacturer
- **T12.1** Use Flash-Over-CAN which is the basis for wireless software updates
- **T12.2** Use partial software Updates, which download only the changed parts of the update binary
- **T13.1** Perform version check
- **T13.2** Perform update code testing
- **T13.3** Perform Rollbacks when needed
- **T14.1** Download an updated module from a software repository
- **T14.2** Add linker metadata to the resulting binary module through a compilation step
- **T14.3** Check compatibility with the already deployed modules maintained in a binary module repository, through a compatibility analysis step
- **T14.4** Verify the module in a simulation or digital twin network, mirroring the actual network, through a functional verification step
- **T14.5** Encrypt and sign the update binary module
- **T14.6** Transfer the binary module to the devices through a dissemination step
- **T14.7** Install and make the binary module operational, through an activation step
- **T15.1** Use a client-server security architecture that includes strong authentication
- **T15.2** Use an implementation of hardware-assisted secure boot and run, such as OTA Manager and Update Installer
- **T15.3** Use security-hardened pre-installed, mobile and third-party apps to enhance protection against attacks
- **T15.4** Use secure Subscriber Identity Module (SIM) based authentication between cellular and Wi-Fi networks using Access Network Discovery and Selection Function (ANDSF)
- **T15.5** Include ability to update software at component level and provision for strong reversion to basic factory software in case of OTA update failure
- **T15.6** Dynamically compare two versions of the firmware and create a delta file to minimize network bandwidth utilization
- **T15.7** Use OTA management to track all client revisions and determine the lifecycle of the software versions
- **T15.8** Include reliably secure data transfer end-to-end with an inbuilt resilience-layer for handling network interruptions, no coverage areas and minimizing data retransmission
- **T16.1** Use public key pre-installation and management
- **T16.2** Use higher key decryption complexity on constrained node devices
- **T17.1** Perform confirmation of a firmware update when is available over an FTP server
- **T17.2** Download the released update file into a T5320A+G flash memory

- **T17.3** Store and verify the firmware update
- **T17.4** Identify the targeted end device
- **T17.5** Reprogram the targeted end device
- **T17.6** Verify software update succeeded
- **T18.1** Update the control software when the device receives a firmware file through an HTTP POST request
- **T18.2** Check periodically for new updates available through a management software
- **T18.3** Store the firmware file in the flash drive when it arrives
- **T18.4** Before upgrading, the integrity, security, and robustness of the firmware file must be validated to check if the downloaded file matches the one on the server
- **T19.1** Use MQTT[10] for critical security patches or Software firmware updates which cannot wait and need reliable, guaranteed and fast delivery
- **T19.2** Use MQTT + CoAP[11] (CoAP encapsulated inside MQTT), which utilizes MQTT protocol as base and encapsulates CoAP protocol inside MQTT as URL
- **T19.3** Use CoAP Only for communicating IoT devices and Gateway. Once the Gateway has received software package, it forwards it to CoAP clients
- **T20.1** Include user network analysis, including host IP address, file names of firmware, the current version of firmware, device brands and types of routers
- **T20.2** Use single-user or batch updating settings depending on different situations
- **T20.3** Use update schedule to arrange multi-period updating tasks and suspend or resume tasks of update modules
- **T20.4** Use single-user and batch update processes to execute firmware update. A Batch update process solves large numbers of routers firmware update. A single user update process solves on-demand and urgent firmware update
- **T21.1** Use a server that uses representational state transfer (REST) architecture to perform the request and receive a response via HTTP protocol
- **T21.2** Assign a specific identifier to a newly uploaded file, which can be fetched by a RESTful API 'GET' request
- **T21.3** Periodic checks by the gateway for latest updated versions of firmware
- **T21.4** Serial communication between gateway and end device after firmware update permanent storage
- **T21.5** Communicate the firmware update file size along with the number of bits that will be transmitted
- **T21.6** Transmit update file through serial cable from the gateway to the end nodes
- **T21.7** In an end device, all received binary data will be stored into its temporary storage to reduce the memory consumption

[10] http://mqtt.org/

[11] https://coap.technology/

- **T22.1** Use End-to-End Security, in which a device must verify that OEM originates a firmware update it receives, and OEM must specify device-specific constraints on the update
- **T22.2** Use update authorization from a controller, in which the controller must control which firmware updates must be installed on Device
- **T22.3** Include attestation of the update installation, by obtaining a verifiable proof of successful update installation
- **T22.4** Include protection of Code and Secret Keys on Device for code integrity
- **T22.5** Impose minimal computational and storage burden on Device

3.3 Results from grey literature

Step PH2.1 (searching grey literature with Google Search) yields eight relevant information sources (see (Table 2), of which six are recognized organizations and institutions, and two are recognized tech blogs in the software and IoT industry.

Table 2. Results from search in Google Search engine

	Search Results	**PH2.1**
Google Search engine	49500 but only 35 were shown since Google omitted not relevant entries (4 pages of search)	8 selected: 1. Texas Instruments 2. DZone 3. Cloud Security Alliance (CSA) 4. Internet Engineering Task Force (IETF) 5. European Union Agency for Network and Information Security (ENISA) 6. Particle 7. INRIA 8. EUROSMART

From Texas Instruments[12] [4], extracted recommendations were grouped in R1 groups of recommendations. From DZone[13] [1], in R2. From Cloud Security Alliance (CSA)[14] [2], in R3. From Internet Engineering Task Force (IETF)[15] [3], in R4. From European Union Agency for Network and Information Security (ENISA)[16] [20], in R5. From Particle[17] [5], in R6. From INRIA[18] [23], in R7. And from EUROSMART[19] [24], in R8.

[12] http://www.ti.com/
[13] https://dzone.com/
[14] https://cloudsecurityalliance.org/
[15] https://www.ietf.org/
[16] https://www.enisa.europa.eu/
[17] https://docs.particle.io/
[18] https://hal.inria.fr/
[19] https://www.eurosmart.com/

Identified and extracted recommendations (R groups):

- **R1.1** OTA deployment operator security
- **R1.2** Incremental roll-out of OTA updates
- **R1.3** Securely downloading the update
- **R1.4** Security from physical attacks
- **R1.5** Authenticating the OTA update image
- **R1.6** Minimizing intrusion
- **R1.7** Reversion if the OTA image fails to boot successfully
- **R2.1** Automatic recovery from corrupted or interrupted updates is a must
- **R2.2** Code provenance and integrity checks are essential
- **R2.3** Code compatibility verification is advisable
- **R2.4** Use secure communication channels by default
- **R2.5** Partial updates should be possible
- **R3.1** Backup the current working configuration of IoT device before applying an update
- **R3.2** Rollbacks should be supported; however, older images should not be reloaded without vendor authorization
- **R3.3** System design should allow administrators to schedule updates to their devices to avoid network saturation and limit unintended downtime
- **R3.4** Vendors should support configuration options by system administrators in support of automatic updates
- **R3.5** One component must manage updates of multiple microcontrollers that compose IoT devices
- **R3.6** Update strategy, differential or complete image should be adapted to the bandwidth constraint
- **R3.7** Updates should be authenticated and integrity protected from end-to-end
- **R3.8** Verification signing keys storage must be secured
- **R3.9** Provide a recovery procedure to cover update failure
- **R3.10** Ensure a long-term support contract by vendors
- **R4.1** The update mechanism must deliver by broadcast, allowing updates to reach multiple users at once
- **R4.2** End-to-end security must be used to verify and validate firmware images
- **R4.3** Rollback attacks must be prevented
- **R4.4** All information necessary for a device to make a decision about the installation of an update must fit into the available RAM of a constrained IoT device
- **R4.5** A power failure at any time during the update process must not cause a failure of the device
- **R4.6** The firmware update mechanism must not require changes to existing firmware file formats
- **R4.7** The new firmware update mechanism must be able to operate with a small bootloader, specific to most IoT devices
- **R4.8** The update mechanism must account for multiple permissions

- **R4.9** The new IoT firmware update architecture must support manifest files containing specific details about the update
- **R5.1** Ensure that the device software/firmware, its configuration, and its applications can update Over-The-Air (OTA)
- **R5.2** Ensure the update server is secure
- **R5.3** Ensure that the update file is transmitted via a secure connection
- **R5.4** Ensure that the update file does not contain sensitive data
- **R5.5** Ensure that the update file is signed by an authorized trust entity and encrypted using accepted encryption methods
- **R5.6** Ensure that the update package has its digital signature, and signing certificate chain, verified by the device before the update process begins
- **R5.7** Offer an automatic firmware update mechanism
- **R5.8** Backward compatibility of firmware updates
- **R5.9** Automatic firmware updates should not modify user-configured preferences, security, and/or privacy settings without user notification
- **R6.1** Atomic updates
- **R6.2** Automatic rollbacks
- **R6.3** Minimal disruption
- **R6.4** Application and Device OS version management Support for updates for sleeping devices
- **R6.5** Encrypted communications
- **R6.6** Sender verification
- **R6.7** Firmware releases
- **R6.8** Release by device groups
- **R6.9** Intelligent firmware releases
- **R7.1** Produce firmware updates that are integrity-protected and authenticated
- **R7.2** Trigger the device to fetch (via push or pull) and verify the integrity and authenticity of a firmware image and then reboot
- **R7.3** Delegate authorization to another maintainer, in case of new ownership or change of contracts (we use the same technique to switch trust anchor when it expires or has to be revoked)
- **R7.4** Reconfigure the device so that cryptographic algorithms can be upgraded if needed
- **R8.1** Control the installation and update of the software in operating systems
- **R8.2** The device should include a secure boot process, verifying that the device bootloader and kernel have not been modified
- **R8.3** Roll-back to a secure state after a security breach occurs or if an upgrade is not successful
- **R8.4** Transmit Update file via a secure connection
- **R8.5** Update file shall not contain sensitive data
- **R8.6** Prevent reverting to old software versions
- **R8.7** Encrypt update file using accepted encryption methods

- **R8.8** Verify signature and certificate in the device before the update process begins
- **R8.9** Perform firmware updates Automatically
- **R8.10** Non-disruptive updates
- **R8.11** Ensure an update is signed cryptographically
- **R8.12** Implement run-time protection and secure execution
- **R8.13** Implement resistance to perturbation, ensuring that the device is resistant to perturbation attacks that could lead to the malfunctioning of the device
- **R8.14** Encrypt data during processing ensuring that the data is not exposed during the update process or accessed by an unauthorized party
- **R8.15** Implement Code obfuscation to protect the confidentiality of source codes/binaries from reverse engineering and IP theft
- **R8.16** Implement Generic Error messages to ensures that the error messages do not reveal any sensitive information to an attacker
- **R8.17** Robust password recovery and reset mechanism In the event of an authentication failure for users safely and securely reset or recover lost and forgotten passwords
- **R8.18** Prevent Version Downgrade by verifying update files versions, to ensure that the device is not being updated with an older and vulnerable version of the firmware
- **R8.19** Ensures that the trust is maintained on devices by checking for authenticity and integrity of the update file using its signature before the file is executed
- **R8.20** Reduces human intervention in update processes
- **R8.21** Ensure updates do not cause disruptions by altering user settings

4 RQ1: A Taxonomy for OTA Updates Mechanisms

This section organizes the gathered information on OTA update solutions, yielding (1) a taxonomy of OTA update mechanisms, and (2) a catalog of techniques for each mechanism. The OTA update taxonomy is based on grouping and homologating the recommendations extracted (R groups), by relating recommendations regarding their aimed quality attribute (QA) inside the OTA update process and deriving into six mechanisms. The techniques catalog elaboration was based on mapping the extracted techniques (T groups) with the mechanisms (M items).

Tables 3 and 4 homologate recommendations presented in the R1 to R8 groups into six mechanisms. The mechanisms are represented by the M items:

- **M1**. Mechanism for secure and safe updates
- **M2**. Mechanism for updates management
- **M3**. Mechanism for code dissemination, propagation and installation
- **M4**. Mechanism for system recovery
- **M5**. Mechanism for updates scheduling
- **M6**. Mechanism for elaboration and packaging

Table 3. Mechanisms classification (part 1)

	R1.1	R1.2	R1.3	R1.4	R1.5	R1.6	R1.7	R2.1	R2.2	R2.3	R2.4	R2.5	R3.1	R3.2	R3.3	R3.4	R3.5	R3.6	R3.7	R3.8	R3.9	R3.10	R4.1	R4.2	R4.3	R4.4	R4.5	R4.6	R4.7	R4.8	R4.9	R5.1	R5.2	R5.3	R5.4	R5.5	R5.6	R5.7	R5.8	R5.9
M1	X		X	X	X	X			X	X	X								X	X				X	X		X			X			X	X	X	X	X			X
M2		X														X						X																		
M3																	X	X					X																	
M4							X	X					X	X							X																		X	
M5															X																							X		
M6												X														X		X	X		X	X								

Table 4. Mechanisms classification (part 2)

	R6.1	R6.2	R6.3	R6.4	R6.5	R6.6	R6.7	R6.8	R6.9	R7.1	R7.2	R7.3	R7.4	R8.1	R8.2	R8.3	R8.4	R8.5	R8.6	R8.7	R8.8	R8.9	R8.10	R8.11	R8.12	R8.13	R8.14	R8.15	R8.16	R8.17	R8.18	R8.19	R8.20	R8.21
M1			X		X	X				X	X				X		X	X		X	X		X	X	X	X	X	X	X	X	X	X		X
M2				X			X	X	X			X		X																				
M3																																		
M4		X														X			X															
M5																						X											X	
M6	X												X																					

Tables 5-7 show the classification of the techniques presented in the T1 to T22 groups. The classification was based on mapping the mechanisms of the OTA update taxonomy with the techniques, deriving into a techniques catalog for OTA updates in the IoT and CPS context. The taxonomy and catalog can be used by engineers and practitioners while designing OTA update systems for CPS or IoT systems.

Table 5. Techniques classification into mechanisms (part 1)

	T1.1	T1.2	T1.3	T1.4	T1.5	T1.6	T1.7	T1.8	T2.1	T2.2	T2.3	T2.4	T2.5	T3.1	T3.2	T3.3	T3.4	T3.5	T3.6	T3.7	T3.8	T3.9	T3.10	T3.11	T3.12	T3.13	T4.1	T4.2	T4.3	T4.4	T4.5	T4.6	T4.7	T4.8	T4.9	T4.10	T4.11	T5.1	T5.2	T5.3	T6.1	T6.2	T6.3	T6.4	T6.5	T6.6
M1										X				X	X	X											X	X	X		X			X	X						X	X	X		X	X
M2																						X				X				X						X	X									
M3									X		X	X	X												X							X	X					X	X	X				X		
M4																																														
M5																																														
M6	X	X	X	X	X	X	X	X									X	X	X	X	X		X	X																						

The proposed taxonomy is shown in Figure 1. The techniques catalog is shown in Figure 2. Each mechanism in the proposed taxonomy is described as follows:

1. **Mechanism for secure and safe updates (M1)**: Performs the download of the update files through secure channels without exposing the system to possible attacks through software or hardware.
2. **Mechanism for updates management (M2)**: Allows vendors or users to create strategies for controlling and performing updates physically or remotely.
3. **Mechanism for code dissemination, propagation, and installation (M3)**: Defines the update dissemination and installation strategy and exposes the update details and information to orchestrate the installation in IoT or CPS networks.

Table 6. Techniques classification into mechanisms (part 2)

	T7.1	T7.2	T7.3	T7.4	T7.5	T7.6	T7.7	T7.8	T7.9	T7.10	T7.11	T7.12	T7.13	T8.1	T8.2	T8.3	T8.4	T8.5	T9.1	T9.2	T9.3	T9.4	T9.5	T9.6	T10.1	T10.2	T10.3	T10.4	T10.5	T10.6	T10.7	T10.8	T11.1	T11.2	T11.3	T12.1	T12.2	T13.1	T13.2	T13.3	T14.1	T14.2	T14.3	T14.4	T14.5	T14.6	T14.7
M1																		X	X			X			X	X	X	X	X	X													X	X	X		
M2														X	X								X									X						X			X						
M3	X	X	X	X	X	X	X	X	X	X		X	X							X	X			X							X			X	X	X										X	X
M4																X																							X	X							
M5																	X																														
M6											X																						X				X					X					

Table 7. Techniques classification into mechanisms (part 3)

	T15.1	T15.2	T15.3	T15.4	T15.5	T15.6	T15.7	T15.8	T16.1	T16.2	T17.1	T17.2	T17.3	T17.4	T17.5	T17.6	T18.1	T18.2	T18.3	T18.4	T19.1	T19.2	T19.3	T20.1	T20.2	T20.3	T20.4	T21.1	T21.2	T21.3	T21.4	T21.5	T21.6	T21.7	T22.1	T22.2	T22.3	T22.4	T22.5
M1	X	X	X	X				X	X	X	X			X			X			X															X		X	X	
M2						X	X											X						X	X			X	X			X				X			
M3												X	X		X				X		X	X	X				X				X		X	X					
M4																X																							
M5																										X				X									
M6					X																																		X

4. **Mechanism for system recovery (M4)**: Allows uninstalling an updated image in case of malfunctioning after the update and exposes a way for recovering to an initial and stable state on the end devices.
5. **Mechanism for updates scheduling (M5)**: Allows the scheduling of the updates by vendors or users remotely or through a local application.
6. **Mechanism for elaboration and packaging (M6)**: Structures and organizes the update image in full or partial updates, and specifies the strategy for the compression/decompression of update images/packages in both the server and the end device.

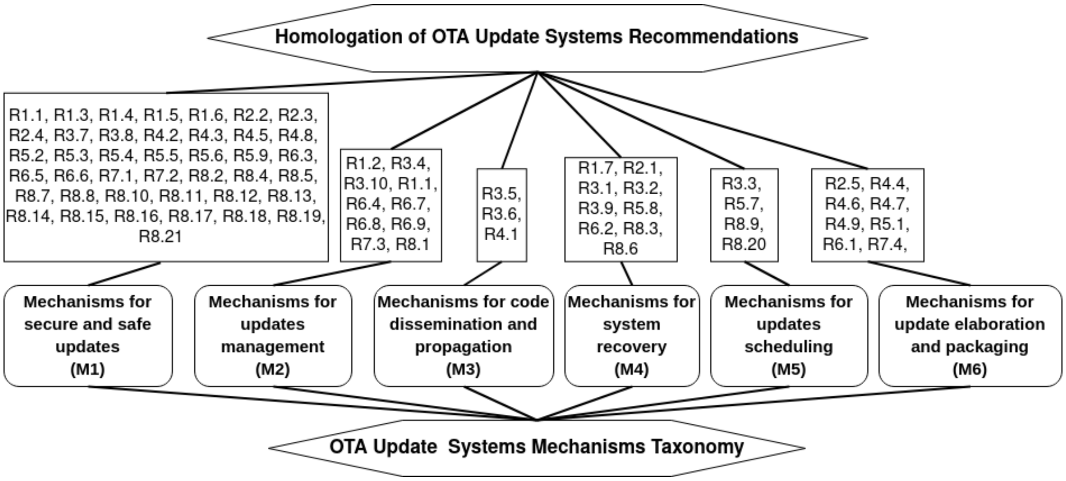
Homologation of OTA Update Systems Recommendations
R1.1, R1.3, R1.4, R1.5, R1.6, R2.2, R2.3, R2.4, R3.7, R3.8, R4.2, R4.3, R4.5, R4.8, R5.2, R5.3, R5.4, R5.5, R5.6, R5.9, R6.3, R6.5, R6.6, R7.1, R7.2, R8.2, R8.4, R8.5, R8.7, R8.8, R8.10, R8.11, R8.12, R8.13, R8.14, R8.15, R8.16, R8.17, R8.18, R8.19, R8.21
R1.2, R3.4, R3.10, R1.1, R6.4, R6.7, R6.8, R6.9, R7.3, R8.1
R3.5, R3.6, R4.1
R1.7, R2.1, R3.1, R3.2, R3.9, R5.8, R6.2, R8.3, R8.6
R3.3, R5.7, R8.9, R8.20
R2.5, R4.4, R4.6, R4.7, R4.9, R5.1, R6.1, R7.4,
Mechanisms for secure and safe updates (M1)
Mechanisms for updates management (M2)
Mechanisms for code dissemination and propagation (M3)
Mechanisms for system recovery (M4)
Mechanisms for updates scheduling (M5)
Mechanisms for update elaboration and packaging (M6)
OTA Update Systems Mechanisms Taxonomy

Fig. 1. OTA Update Mechanisms Taxonomy

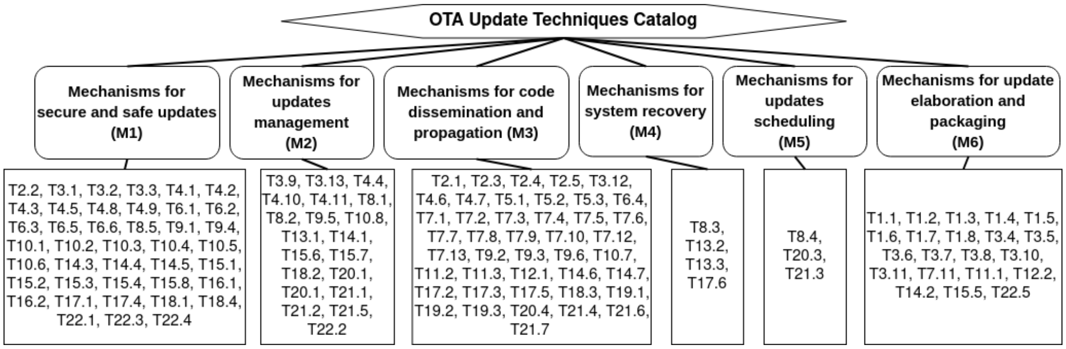
OTA Update Techniques Catalog
Mechanisms for secure and safe updates (M1)
Mechanisms for updates management (M2)
Mechanisms for code dissemination and propagation (M3)
Mechanisms for system recovery (M4)
Mechanisms for updates scheduling (M5)
Mechanisms for update elaboration and packaging (M6)
T2.2, T3.1, T3.2, T3.3, T4.1, T4.2, T4.3, T4.5, T4.8, T4.9, T6.1, T6.2, T6.3, T6.5, T6.6, T8.5, T9.1, T9.4, T10.1, T10.2, T10.3, T10.4, T10.5, T10.6, T14.3, T14.4, T14.5, T15.1, T15.2, T15.3, T15.4, T15.8, T16.1, T16.2, T17.1, T17.4, T18.1, T18.4, T22.1, T22.3, T22.4
T3.9, T3.13, T4.4, T4.10, T4.11, T8.1, T8.2, T9.5, T10.8, T13.1, T14.1, T15.6, T15.7, T18.2, T20.1, T20.1, T21.1, T21.2, T21.5, T22.2
T2.1, T2.3, T2.4, T2.5, T3.12, T4.6, T4.7, T5.1, T5.2, T5.3, T6.4, T7.1, T7.2, T7.3, T7.4, T7.5, T7.6, T7.7, T7.8, T7.9, T7.10, T7.12, T7.13, T9.2, T9.3, T9.6, T10.7, T11.2, T11.3, T12.1, T14.6, T14.7, T17.2, T17.3, T17.5, T18.3, T19.1, T19.2, T19.3, T20.4, T21.4, T21.6, T21.7
T8.3, T13.2, T13.3, T17.6
T8.4, T20.3, T21.3
T1.1, T1.2, T1.3, T1.4, T1.5, T1.6, T1.7, T1.8, T3.4, T3.5, T3.6, T3.7, T3.8, T3.10, T3.11, T7.11, T11.1, T12.2, T14.2, T15.5, T22.5

Fig. 2. OTA Update Techniques Catalog

5 Conclusions

OTA (Over-the-air) Update is an essential functionality of IoT and Cyber-Physical Systems. This article presented the design and results of a systematic literature review of academic and grey (professional) literature, which addresses two research questions about (1) which approaches have been proposed and (2) how they can be classified. Two hundred six (206) approaches were identified, 132 in academic literature, and 74 in grey sources. A careful analysis led to classify then into six common mechanisms, allowing to create an (initial) catalog of techniques for OTA update systems.

Although OTA update systems are not a new topic, most of the grey literature reports describe only empirical information or learned experiences, rather than reflective approaches or standardization; indeed, we found evidence for only one draft standardization proposal. This undoubtedly leads developers and OEMs to develop and implement non-standard and ad-hoc, empirically-based OTA software update systems.

We also detected that OTA update systems are essential to all IoT or CPS system design since improvements and maintenance are continuously required after deployment. OTA update systems must be designed to exhibit several quality attributes: scalability (since any amount of devices might be in a network); availability (since images must be ready to be used for performing updates and devices can not stop working after updates); and security (since downloads, authentications, and installation must always be in a secure way).

Since there are several technologies and techniques that can be included in a design, having a taxonomy of mechanisms and a catalog of techniques will allow practitioners to quickly identify design options. Further work will explore how to help designers to compare alternatives and make actual design decisions with the available information.

Acknowledgments

This work has been partially supported by ANID PCHA/Doctorado Nacional under grant 2017-21171351 and ANID PIA/APOYO AFB180002 (CCTVal).

References

1. Califano, J. 2018. “How to Approach OTA Updates for IoT”, DZone (2018-06-19). https://dzone.com/articles/how-to-approach-ota-updates-for-iot.
2. Cloud Security Alliance. 2018. “Recommendations for IoT Firmware Update Processes”. https://downloads.cloudsecurityalliance.org/assets/research/internet-of-things/recommendations-for-iot-firmware-update-processes.pdf.
3. Moran, B., Meriac, M. and Tschofenig, H. 2017. “A Firmware Update Architecture for Internet of Things Devices”. https://tools.ietf.org/html/draft-moran-suit-architecture-00.
4. Lethaby, N. 2018. “A more secure and reliable OTA update architecture for IoT devices”. http://www.ti.com/lit/wp/sway021/sway021.pdf.
5. Particle. 2018. “OTA Firmware Updates”. https://docs.particle.io/tutorials/device-cloud/ota-updates.
6. Lee, E. A. 2010. “CPS foundations”. Design Automation Conference, pp. 737-742.
7. Villegas, M. M., Orellana, C. and Astudillo, H. 2019. “A study of over-the-air (OTA) update systems for CPS and IoT operating systems”. Proceedings of the 13th European Conference on Software Architecture-Volume 2, pp. 269-272.
8. Orellana, C., Villegas, M. M. and Astudillo, H. 2019. “Mitigating Security Threats through the use of Security Tactics to design Secure Cyber-physical Systems (CPS)”. Proceedings of the 13th European Conference on Software Architecture-Volume 2. pp. 109-115.
9. Loukas, G. 2015. “Cyber-Physical Attacks: A Growing Invisible Threat”. 1st edition, Butterworth-Heinemann.
10. Seifert, D. and Reza, H. 2016. “A Security Analysis of Cyber-Physical Systems Architecture for Healthcare”. Computers. 5. 24.
11. Orellana, C., Villegas, M. M. and Astudillo, H. 2019. “Architectural Tactics for Scalability”. XXII Ibero-American Conference on Software Engineering, pp. 128-140.
12. Kavya, M., Kishore, C. and Rajath Kumar, K. S. 2018. “Verification and validation of data on Wi-Fi Over The Air(OTA)”. 3rd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology, pp. 429-433.
13. Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J. and Linkman, S. 2009. “Systematic literature reviews in software engineering – A systematic literature review”. Inf. Softw. Technol. 51, 1, 7–15.
14. Park, H., Kim, H., Kim, S., Mah, P. and Lim, C. 2019. “Two-phase dissemination scheme for CoAP-based firmware-over-the-air update of wireless sensor networks: demo abstract”. In Proceedings of the 17th Conference on Embedded Networked Sensor Systems. Association for Computing Machinery, pp. 404–405.
15. Mbakoyiannis, D., Tomoutzoglou, O. and Kornaros, G. 2019. “Secure over-the-air firmware updating for automotive electronic control units”. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. Association for Computing Machinery, pp. 174–181.
16. He, X., Alqahtani, S., Gamble, R. and Papa, M. 2019. “Securing Over-The-Air IoT Firmware Updates using Blockchain”. In Proceedings of the International Conference on Omni-Layer Intelligent Systems. Association for Computing Machinery, pp. 164–171.

17. Brown, S., Sreenan, C.J. 2013. “Software Updating in Wireless Sensor Networks: A Survey and Lacunae”. Journal of Sensor and Actuator Networks. 2. pp. 717-760.
18. Halder, S., Ghosal, A. and Conti, M. 2019. “Secure OTA Software Updates in Connected Vehicles: A survey”. ArXiv, abs/1904.00685.
19. Ruckebusch, P., De Poorter, E., Fortuna, C. and Moerman, I. 2015. “GITAR: Generic extension for Internet-of-Things ARchitectures enabling dynamic updates of network and application modules”. Ad Hoc Networks. 36.
20. European Union Agency for Network and Information Security (ENISA). 2017. “Baseline Security Recommendations for IoT in the context of Critical Information Infrastructures”. https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot/at_download/fullReport
21. National Telecommunications and Information Administration (NTIA) - Existing Standards, Tools and Initiatives Working Group (WG1). 2017. ”Catalog of Existing IoT Security Standards Version 0.01” (2017-09-12). https://www.ntia.doc.gov/files/ntia/publications/iotsecuritystandardscatalog_draft_09.12.17.pdf
22. Asokan, N., Nyman, T., Rattanavipanon, N., Sadeghi, A. and Tsudik, G. 2018. “ASSURED: Architecture for Secure Software Update of Realistic Embedded Devices”. ArXiv, abs/1807.05002.
23. Zandberg, K., Schleiser, K., Acosta, F., Tschofenig, H. and Baccelli, E. 2019. “Secure Firmware Updates for Constrained IoT Devices Using Open Standards: A Reality Check“. IEEEAccess, IEEE, 7, pp.71907-71920.
24. Eurosmart. 2019. “Technical Report TR-e-IoT-SCS-Part-2”. https://www.eurosmart.com/wp-content/uploads/2019/11/Part-2-Eurosmart_IoTsCS-GPP_v1.2_RELEASE.pdf
25. Kauer, F., Meyer, F. and Turau, V. 2017. “A holistic solution for reliable over-the-air software updates in large industrial plants”. 13th Workshop on Intelligent Solutions in Embedded Systems, pp. 29-34.
26. Kumar, S. K., Sahoo, S., Kiran, K., Swain, A. K. and Mahapatra, K. K. 2018. “A Novel Holistic Security Framework for In-Field Firmware Updates”. http://dspace.nitrkl.ac.in/dspace/bitstream/2080/3140/1/2018_ISES_SKumarK_NovelHolistic.pdf
27. Kim D., Nam H. and Kim D. 2017. “Adaptive Code Dissemination Based on Link Quality in Wireless Sensor Networks”. IEEE Internet of Things Journal. 4. 3. pp. 685-695.
28. Thakur, P., Bodade, V., Achary, A., Addagatla, M., Kumar, N. and Pingle, Y. 2019. “Universal Firmware Upgrade Over-The-Air for IoT Devices with Security”. 6th International Conference on Computing for Sustainable Global Development, pp. 27-30.
29. Nikolov, N. 2018. “Research Firmware Update Over the Air from the Cloud”. IEEE XXVII International Scientific Conference Electronics - ET, pp. 1-4.
30. Thantharate, A., Beard, C. and Kankariya, P. 2019. “CoAP and MQTT Based Models to Deliver Software and Security Updates to IoT Devices over the Air”. International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data, pp. 1065-1070.
31. Doddapaneni, K., Lakkundi, R., Rao, S., Kulkarni, S. G. and Bhat B. 2017. “Secure FoTA Object for IoT”. IEEE 42nd Conference on Local Computer Networks Workshops, pp. 154-159.
32. Chandra, H., Anggadjaja, E., Wijaya, P. S. and Gunawan, E. 2016. “Internet of Things: Over-the-Air (OTA) firmware update in Lightweight mesh network protocol for smart urban development”. 22nd Asia-Pacific Conference on Communications, pp. 115-118.

33. Dhobi, R., Gajjar, S., Parmar, D. and Vaghela, T. 2019. “Secure Firmware Update over the Air using TrustZone”. Innovations in Power and Advanced Computing Technologies, pp. 1-4.
34. Witanto, E. N., Oktian, Y. E., Kumi, S. and Lee, S. 2019. “Blockchain-based OCF Firmware Update”. International Conference on Information and Communication Technology Convergence, pp. 1248-1253.
35. Steger, M., Boano, C. A., Niedermayr, T., Karner, M., Hillebrand, J., Roemer, K., Rom, W. 2018. “An Efficient and Secure Automotive Wireless Software Update Framework”. In IEEE Transactions on Industrial Informatics, vol. 14, no. 5, pp. 2181-2193.
36. Varadharajan, V. S., Onge, D. S., Guß, C. and Beltrame, G. 2018. “Over-the-Air Updates for Robotic Swarms”. In IEEE Software, vol. 35, no. 2, pp. 44-50.
37. Teng, C., Gong, J., Wang, Y., Chuang, C. and Chen, M. 2017. “Firmware over the air for home cybersecurity in the Internet of Things”. 19th Asia-Pacific Network Operations and Management Symposium, pp. 123-128.
38. Bauwens, J., Ruckebusch, P., Giannoulis, S., Moerman, I. and Poorter, E. D. 2020. “Over-the-Air Software Updates in the Internet of Things: An Overview of Key Principles”. In IEEE Communications Magazine, vol. 58, no. 2, pp. 35-41.
39. Khurram, M., Kumar, H., Chandak, A., Sarwade, V., Arora, N. and Quach, T. 2016. “Enhancing connected car adoption: Security and over the air update framework”. IEEE 3rd World Forum on Internet of Things, pp. 194-198.
40. Kim, J. Y., Hu, W., Shafagh, H. and Jha, S. 2018. “SEDA: Secure Over-the-Air Code Dissemination Protocol for the Internet of Things”. In IEEE Transactions on Dependable and Secure Computing, vol. 15, no. 6, pp. 1041-1054.
41. Mirfakhraie, T., Vitor, G. and Grogan, K. 2018. “Applicable Protocol for Updating Firmware of Automotive HVAC Electronic Control Units (ECUs) Over the Air”. IEEE International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data, pp. 21-26.
42. Edgar, T. W. and Manz, D. O. 2017. ”Research Methods for Cyber Security, Chapter 2 - Science and Cyber Security”. Syngress.
43. Liu, Q., Shen, J., Xiao, B., Wang, B. W. and Linge, N. 2015. “EECD: A Code Dissemination protocol in a WSN-based Home Energy Management System”. IEEE International Conference on Consumer Electronics, pp. 279-280.
44. Do, Dinh-Sy and Young, K. 2015. “Lightweight Reprogramming and Energy Balancing in Wireless Sensor Networks.” International Journal of Distributed Sensor Networks. 2015. 1-8.
45. Krasniewski, M. D., Panta, R. K., Bagchi, S., Yang, C.-L., and Chappell, W. J. 2008. “Energy-efficient on-demand reprogramming of large-scale sensor networks”. ACM Trans. Sen. Netw. 4, 1, Article 2, 38 pages.
46. Zheng, X. and Sarikaya, B. 2009. “Task dissemination with multicast deluge in sensor networks”. In IEEE Transactions on Wireless Communications, vol. 8, no. 5, pp. 2726-2734.
47. Dong, W., Chen, C., Liu, X., Bu, J. and Gao, Y. 2011. “A lightweight and density-aware reprogramming protocol for wireless sensor networks”. In IEEE Transactions on Mobile Computing, vol. 10, no. 10, pp. 1403-1415.
48. Zhao, Z., Bu, J., Dong, W., Gu, T. and Xu, X. 2015. “CoCo + : Exploiting correlated core for energy efficient dissemination in wireless sensor networks”. Ad Hoc Networks. 37.