Secure Over-the-air Firmware Updating for Automotive Electronic Control Units

Dimitris Mbakoyiannis, Othon Tomoutzoglou and George Kornaros
Informatics Engineering Department, Technological Educational Institute of Crete
Heraklion, Crete, Greece
kornaros@ie.teicrete.gr

ABSTRACT

This work presents secure over-the-air firmware updating that brings a homogenized updating process across OEMs, suppliers and sub-tiers, removing at the same time the costs for individual security precautions and cryptographic countermeasures for each individual component or sub-system. The objective is to overcome all attacks to the servers, to the networks and to the diverse electronic control units (ECUs) in modern vehicles.

The proposed herein secure over-the-air firmware updating, as applied in firmware updating for vehicles, employs separation of roles, e.g., the manager server employs firmware versioning and entitlements for each vehicle and its corresponding ECUs and dependency resolution on behalf of vehicles; In a firmware server, each ECU firmware is associated with metadata that are signed and uploaded by the OEM and/or its suppliers, while a timestamp server on demand records and signs the more recent time for ECUs firmware. An STM32F7xx-based prototype demonstrates a real vehicle case.

CCS CONCEPTS

• **Security and privacy**; • **Computer systems organization → Embedded and cyber-physical systems**;

KEYWORDS

Firmware update over-the-air, Secure VANET communication, Trusted-Authenticated ECU updates

ACM Reference Format:
Dimitris Mbakoyiannis, Othon Tomoutzoglou and George Kornaros. 2019. Secure Over-the-air Firmware Updating for Automotive Electronic Control Units. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3297280.3297299

1 INTRODUCTION

As the self-programming flash memories inside embedded devices and wireless communications increasingly enter the domain of cyber-physical systems, over-the-air updating of firmware, called FOTA, has become possible. We call firmware update the process of replacing the application that executes on top of the Operating System (OS), or Real-Time Operating System (RTOS) of a Cyber-Physical System (CPS) device, or, not only the application, but essentially the full software stack. In most CPS devices, this software commonly includes bootloader, board support package, and the baremetal application, or an RTOS and the application.

FOTA is expected to gain wide acceptance in automotive industry [12] with the increasing number of open cars' ECUs[2], following the great success in mobile phone industry. The current growth of the Internet of Things (IoT), which demands connection between the vehicle and the outside world, makes the challenge even more significant. With the FOTA methodology, the updates can be performed at the customer location and not at the dealership site, thus reducing the fleet management costs. Few vehicles today can receive updates and these updates are mainly designed to update the infotainment or telematics system. Thus, if a fault in the firmware will be discovered during the lifecycle of the vehicle, then the vehicle must be returned to the dealership. On the contrary, in this work we propose to extend the FOTA service to cover the full cyber-physical system. In particular our contributions involve a holistic secure framework by establishing separation of entities (e.g., OEM, Apps provider) and verifying system integrity and authenticity, while employing firmware versioning and entitlements for each vehicle and its corresponding ECUs and dependency resolution on behalf of vehicles.

The rest of the paper is organized as follows. Section 2 gives briefly past works while section 2.1 introduces the principles for secure FOTA. Next, section 3 details the full process. Section 4 discusses security assumptions and analysis and finally section 5 demonstrates the prototype CPS implementation and section 6 concludes the paper.

2 RELATED WORK AND PRELIMINARIES

This section discusses background concepts and assumptions in the scope of remote firmware updating.

The vehicular firmware updates process and the associated security issues are increasingly important in modern cars and researchers have proposed strong solutions [5] [7] and in IoT-era [8]. However, as in Mansor et al. [5], not all entities are authenticated in conjunction with role separation, and hence, indistinguishably entities that may be able to conduct the firmware update process depending on their different capabilities (OEM, dealer, mechanic, etc). In the same way disregarding different roles, in [6] Wolf et al., propose to secure ECU updates over the air by signing the program code using RSA secret key and verifying it by the control units holding the corresponding public key. Blockchain has also been proposed recently, to enable a distributed and tamper-proof

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC '19, April 8–12, 2019, Limassol, Cyprus
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297299

data exchange for automotive software updates among smart vehicles [11]. Hardware-assisted solutions are also employed for secure over-the-air updates in healthcare applications[3].

However, similar solutions as our framework have been proposed to address update delivery of software resilient to key compromise, such as TUF [9], while its descendant Uptane [4] extends and adapts TUF to support secure updates for automobile systems. However, both mechanisms neglect secure in-vehicle networking and ignore actual realization on any real platform.

2.1 Preliminaries

In automotive industry, an original equipment manufacturer (OEM) chooses the ECUs that reside on a vehicle, usually produced by third-party organizations known as suppliers. In principle, Tier-1 suppliers, which directly supply OEMs, may outsource the development of components on ECUs (e.g., the baseband module) to tier-2 suppliers, which in turn may then outsource to tier-3 suppliers. Different OEMs may share the same suppliers. The software for an ECU is maintained by a supplier, and delivered to the OEM to be distributed to vehicles. Thus, the source code for software on ECUs may not be available to OEMs.

The OEM maintains a software repository that hosts software updates produced both by the OEM and suppliers. An OEM may push software updates to all applicable vehicles, or a vehicle may pull the latest known software updates. Some gateway ECUs, or telematics and infotainment ECUs, have a wireless connection to the Internet. Updates are disseminated to vehicles via this wireless connection, or through physical distribution channels, such as dealerships, customer-inserted USB sticks, or OBD-II ports usually reserved for diagnostics.

We assume that all software on the repository is organized by images. Unlike a package, where a single computer can install many packages, we assume that there is exactly one image per ECU. A vehicle would download and install a bundle, or a set of images released by the repository that is meant to be installed by all ECUs on a vehicle at the same time.

The FOTA update adds a layer of signed metadata, such as cryptographic hashes and file sizes, to the software repository. By verifying these metadata, security attacks can be detected and prevented before installation. For example, if the hash of a downloaded software update does not match the signed hash of the same update in the metadata, then this is an indication of a possible security attack. The objective of this secure FOTA update framework is to limit the impact of a compromise.

The first principle of this secure FOTA update framework is separation of duties. Different pieces of metadata are signed by the repository administrators using different roles in order to distribute responsibilities and increase compromise-resilience. The four basic roles that must exist on a secured repository include: the root, timestamp, release, and targets roles. The root role serves as the certificate authority: it distributes and revokes the public keys used to verify metadata produced by each of these four roles (including itself). The timestamp role indicates whether there are any new metadata or images on the repository. The release role indicates which images have been released by the repository at the same time. The targets role provides metadata, such as hashes and file sizes of images, and may delegate the responsibility of signing metadata about images to other, custom-made roles. A delegation binds the public keys used by a delegate to a subset of the images these keys are trusted to sign.

The second design principle is the use of multi-trust signatures. The metadata file for a role can be configured so it must be signed using a minimum number of t out of n keys. This significantly increases the number of keys that must be compromised to launch an attack.

The third principle is an explicit or implicit process to revoke keys. The public keys used to verify a metadata file can be implicitly revoked by including an expiration date in the metadata file (so that the keys would be considered expired after that date), or they can be explicitly revoked by signing new metadata that replaces the old keys with new keys. The fourth principle is using offline keys, or private keys kept physically disconnected from the Internet, to sign the most sensitive roles. This is done so that, even if attackers compromise the repository, they are unable to sign new and malicious versions of sensitive metadata.

By combining these four design principles, the impact of a key compromise can be distributed and minimized such that users are protected from many security attacksâĂŤeven if the repository is compromised.

2.2 Back-end Servers

A repository contains metadata and images. An image is an archive that contains code and/or data necessary for an ECU to function. Metadata describe higher level information, such as the cryptographic hashes and file lengths of images or even other metadata, that can be used to verify that images are up to date and have not been altered since being uploaded to the repository. Different roles produce different pieces of metadata on a repository. The OEM may use repository tools to generate signed metadata, and add it to a repository. An OEM shall include two different types of repositories (see Figure 1).

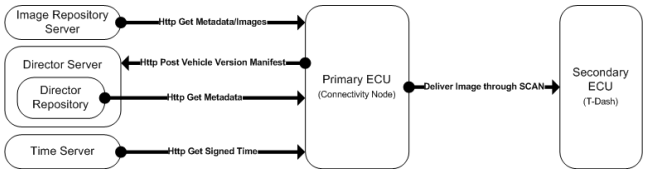

Figure 1: Overview of the FOTA update procedure.

- Image repository: The OEM shall build and run an image repository, which contains unencrypted images as well as associated metadata signed and uploaded by the OEM and/or its suppliers.
- Director repository: The OEM shall also build and run a director repository, which contains metadata about which images should be installed by ECUs. Unlike the image repository, the director repository: (1) produces metadata on demand, (2) does instruct ECUs as to which images should be installed, and (3) may encrypt images per ECU.
- Time server: Finally, the OEM shall build and run a time server, which is not a repository of metadata and images,

but an online server that signs the latest time on demand for ECUs.

The director role allows the OEM to completely control the choice of images downloaded and installed by vehicles. The director role functions much like a traditional package manager, except that it operates on the server side instead of the client. It can be used to instantly blacklist faulty versions of software, or to customize software for different vehicles of the same type when vehicle owners may have paid for extra features. When a vehicle contacts the repository for updates, it must identify itself to the director using its vehicle identification number (VIN) number. Given the VIN, the director would consult the inventory database for which ECUs are installed on this vehicle. Then, the director signs director metadata, or fresh instructions for the vehicle about which images should be downloaded and installed by its ECUs. These instructions bind the unique serial number of every ECU to the filename and hash of the image it should install. Since these instructions are always customized for every vehicle, and freshly signed by the director role, a man-in-the-middle attacker cannot copy these instructions and replay them to other vehicles. The director also performs dependency resolution on behalf of vehicles. This means that if one ECU image depends on another, then the director would include both of these images in its instructions.

3 SECURE FOTA

ECUs on a vehicle are classified as either a primary or a secondary. For the sake of simplicity, we assume that the Connectivity Node is the only one primary ECU in the vehicle. The primary downloads, verifies, and distributes metadata, as well as images, to all secondary ECUs. This is done so to protect them from security attacks. A secondary depends upon the primary to be given its metadata and image, but verifies them before installing the image.

Depending on whether an ECU is critical to the operational security of a vehicle, an ECU should perform either partial or full verification of metadata. Partial verification means that the director repository is the only root of trust about the image to be installed on an ECU, whereas full verification means that both the image and director repositories must agree about the image to be installed. Primary and safety-critical secondary ECUs that are updated over-the-air must use full verification. Non-safety-critical secondary ECUs that are updated over-the-air should use at least partial verification and if resources permit they should perform full verification. If they are incapable of performing even partial verification, then they should not be updated over-the-air.

3.1 Secure FOTA Process

Figure 2 shows the overall architecture that consists of multiple vehicle ECUs, where the main units are a "Trusted Dashboard" (T-Dash), the Connectivity Node to perform gateway functionality and a secure hub (e.g., ODB-II) to trigger the FOTA process.

This section describes the step-by-step procedure until a new firmware image is securely delivered to the T-Dash. The Connectivity Node, which is considered as the primary ECU, communicates with the back-end servers over a network interface (LTE/Wi-Fi) in order to download and verify metadata and images on behalf of the T-Dash. The T-Dash is considered as a secondary ECU that communicates with the primary ECU to receive a fresh firmware image through SCAN. The Connectivity Node is responsible to perform any actions required on behalf of the T-Dash. The connection between the ECUs is done via a CAN-bus where frames are encrypted for security reasons; however, in the current context we do not examine the CAN-bus encryption method.

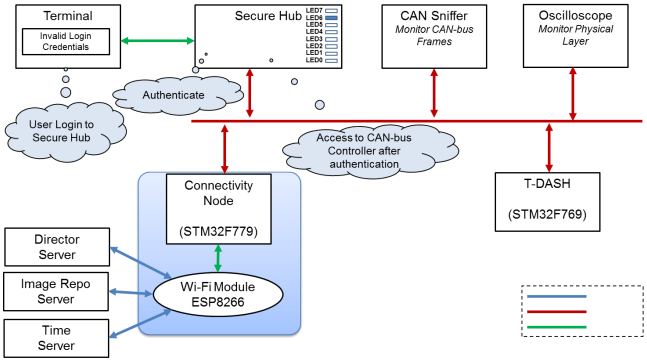

Figure 2: Overview of the prototype platform performing secure FOTA.

3.1.1 Key Generation. The primary and any secondary ECU must first acquire a set of keys that are used to sign any data that must be delivered to the Director server. This step is performed by the primary ECU which executes a C-implementation of the ED25519 digital signature scheme. First, a 32 byte sequence is produced using a random number generator (RNG) that represents the private key. Next, the ED25519 generates a 32 byte public key based on the private key. Combining the private and public keys the ED25519, also, generates a signature which, hereafter, will be known as a Key ID. This Key ID may be used to authenticate the public key. Each ECU has a unique set of keys (Figure 3).

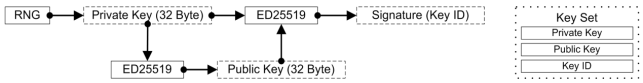

Figure 3: Key Generation Process for each ECU (Primary or Secondary).

3.1.2 Registration of Vehicle Nodes to the Directory Server. Manifests that may be transmitted to the Director server are signed by using the keysets of the Connectivity Node as well as the keysets of the T-Dash. The Director server must be, in advance, aware of these keys in order to authenticate any received manifests. In this step, the Connectivity Node sends over a HTTP Post a XML/RPC structure along with its keys to register itself with the Director Server. Afterwards, it registers the T-Dash by sending a second XML/RPC structure along with the keys of the T-Dash.

3.1.3 Request and Validate Timestamp. The Connectivity Node must request a timestamp from the Timeserver that will be used to check for expired metadata. The request for a timestamp is a XML/RPC structure carried over a HTTP Post. The timestamp is

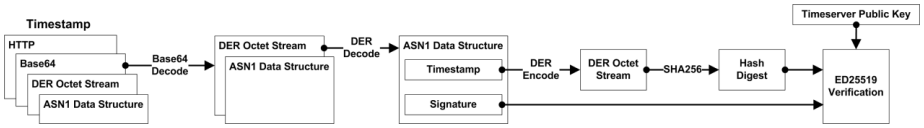

Figure 4: Timestamp verification process.

received in Base64 format. As depicted in Figure 4 the primary ECU validates the timestamp through the following process:

- Decode the Base64 data which reveals a DER octet stream.
- Decode the DER octet stream to reconstruct the ASN1 timestamp module which, actually, consists of the timestamp and its signature.
- Encode the timestamp with the DER encoding rules to produce a new octet stream and calculate the SHA256 hash digest of the new octet stream.
- Use the ED25519 to verify the signature of the ASN1 timestamp module according to the hash digest and the Timeserver's public key.

3.1.4 Metadata Update. At this step the Connectivity Node must make HTTP Get requests to download all required metadata files from the Director and Image Repository servers. Each repository has a set of root, timestamp, snapshot and targets metadata files available which results in a total of eight files from both repositories. There is dependence between the metadata files, thus, they must be fetched with the following sequence: root, timestamp, snapshot, targets. Each metadata file consists of two individual parts, the metadata body and a signature. The metadata body carries the concerned information. The signature is produced by signing the metadata body. The primary ECU must perform the generic metadata update procedure for downloading and verifying metadata files which is described in the steps below:

- Download the metadata file which is transmitted as a DER octet stream.
- Decode the received octet stream using the DER decoding rules to reconstruct the metadata in the form of ASN1 data structure.
- Compare the expiration date contained in the metadata body with the latest downloaded timestamp from the Timeserver to verify that the metadata is not expired.
- Encode individually the metadata body as a DER octet stream.
- Calculate the SHA256 hash digest of the DER octet stream.
- Validate the authenticity of the metadata file. To do so, it uses the generated hash digest and the public key of the referenced role as inputs for the ED25519 in order to verify the signature of the metadata body.

3.1.5 Root Metadata Update. The root metadata file is a ASN1 data structure that contains four public keys that are used to sign the root metadata itself as well as the timestamp, snapshot and targets metadata respectively. The servers encode the root metadata file to a DER octet stream before being transferred. To update the root metadata the primary ECU initially checks its SD card for local root metadata that may be already available from a previous update process. If a local root metadata file is not available or is expired the primary ECU must perform the generic metadata update procedure for downloading and verifying a fresh root metadata file. Otherwise, it must perform the generic metadata update procedure bypassing its first step since downloading is not required. Finally, the verified root metadata file must be stored as a DER octet stream in a specified offset of the SD card for future usage. If the verification of the root metadata file is successful the primary ECU, also stores the four public keys in a keys table to later verify the timestamp, snapshot and targets metadata files.

3.1.6 Timestamp Metadata Update. The timestamp metadata provides information about the snapshot metadata file such as its filename, file length and the hashes that must be used to verify its integrity. The primary ECU must always perform a fresh download to make sure that the timestamp metadata file is the latest possible due to the fact that the referenced snapshot metadata file may change unexpectedly. To download and verify the timestamp metadata only the generic metadata update procedure is required. The usage of the SD card is excluded since data storing is not required in this case.

3.1.7 Snapshot Metadata Update. The snapshot metadata file informs about all targets metadata files released by the repository at the given time which indirectly indicates which images are currently available by the Image Repository. To update the snapshot metadata the primary ECU primarily checks its SD card for local snapshot metadata that may, also, be available from a previous update process. The first step of the generic metadata update procedure for downloading will be bypassed if there are available up-to-date snapshot metadata locally. Before proceeding with the rest steps of the generic metadata update procedure the primary ECU authenticates the integrity of the snapshot metadata according to the file length and hash information provided from the latest downloaded timestamp metadata file. Once the generic metadata update procedure is completed the verified snapshot metadata file will be stored as a DER octet stream in a specified offset of the SD card for future usage.

3.1.8 Targets Metadata Update. The targets metadata file lists information about the filename, file length, SHA256 and SHA512 hashes and the ID of the corresponding ECU of each available firmware image. In this implementation, the targets metadata file includes information specifically for the firmware image that should be installed by the T-Dash (see Figure 5). The primary ECU requests the targets metadata file from the server according to the targets metadata filename provided in the snapshot metadata file. To update the targets metadata the primary ECU primarily checks its SD card for local targets metadata that may reside from a previous update process. Afterwards, if a local targets metadata file is not available or is expired the primary ECU must perform the generic

metadata update procedure for downloading and verifying a fresh targets metadata file. Otherwise, it must perform the generic metadata update procedure bypassing its first step since downloading is not required. Once the generic metadata update procedure is completed the verified targets metadata file will be stored as a DER octet stream in a specified offset of the SD card for future usage.

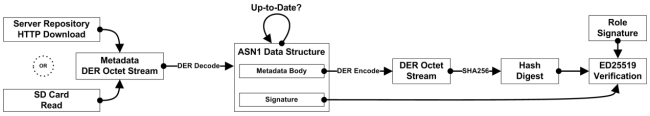

Figure 5: The generic metadata update procedure.

3.1.9 Check Targets Metadata for Valid Firmware Images. The Connectivity Node now has a targets metadata file from the Director server and another targets metadata file from the Image Repository server (see Figure 6). Both targets metadata files must have matching information for a firmware image so that the latter can be considered valid for download. As previously mentioned a targets metadata file lists information about the filename, file length, SHA256 and SHA512 hashes for each available firmware image. The targets metadata file of the Director server additionally references a firmware image with its corresponding ECU ID. The connectivity Node initially searches both targets metadata files for a matching filename of a firmware image. The next step is to check that the file length that refers to the firmware image, also, matches between the targets metadata files. Finally, in the same manner the SHA256 and SHA5212 hashes must be found equal. A full match in all checks indicates that there is a valid firmware image for download.

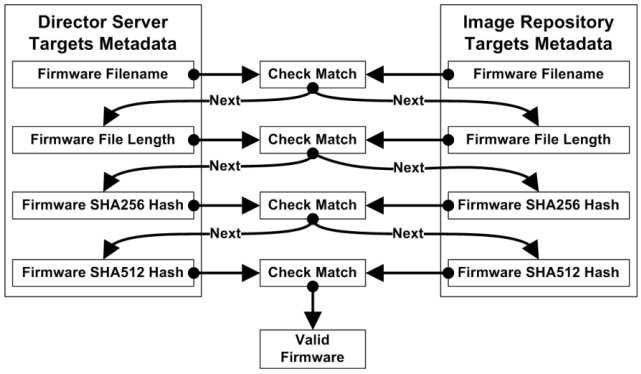

Figure 6: Check targets metadata for valid firmware images to download.

3.1.10 Download a Valid Firmware Image for the T-Dash. The Connectivity Node sends a HTTP Get request to the Image Repository server to download the firmware image as binary data according to the filename of the targets metadata file (see Figure 7). The firmware image must be stored directly to the SD card due to the restricted memory of the Connectivity Node. Initially, the firmware data are received through the network interface in small chunks and temporally stored in a local buffer. The Connectivity Node keeps copying chunks of 512 bytes from the buffer to sequential sectors of the SD card until the total firmware image is stored. In order to authenticate the received firmware image the Connectivity Node, primarily, compares the file length and the SHA256 and SHA512 hashes referenced in the targets metadata file with the actual file length and the calculated hashes of the received data.

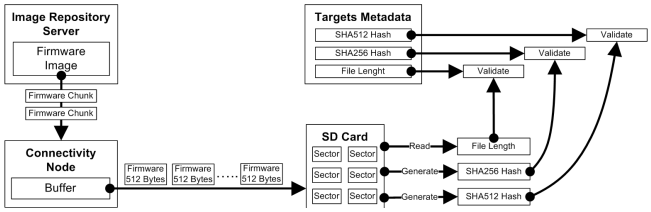

Figure 7: Firmware image download and validation procedure.

3.1.11 Deliver the Fresh Firmware Image to the T-Dash. The transferring sub-process involves a multi-frame procedure which breaks down the firmware into verifiable transmission blocks using a CRC checksum. In the end of this sub-process, the client device (T-Dash) will soft-reset itself to trigger the upgrading firmware step.

The multi-frame procedure is inspired by the KWP2000 protocol, already implemented in many modern vehicles [10]. The format of the frames sent by the "source" to the "destination" is indicated below.

B0	B1	B2	B3	B4	B5	B6	B7
TargetID	Length	ServiceID	SessionType	D0	D1	D2	D3

Each frame contains a common preamble which includes a unique Target ID, a data length, a Service ID, which identify the meaning of the frame and a Session type, equal to 0x10 for FOTA. Below is a list of the available Service IDs used by the FOTA procedure:

0x11 Start Upgrade Session
0x12 Transfer Request
0x13 Transfer Data
0x14 CRC Verification
0x15 Stop Upgrade Session
0x16 End of Upgrade Firmware Procedure

For every incoming frame, the destination must reply with an acknowledge or a not-acknowledge frame. Both frames contain a unique Target ID, a data length, a byte that indicates if the reply is positive (Service ID+0x10) or negative (0xFF), the session type and, in case of not-acknowledge, a NACK code which indicates the reason for a negative response. If C-Node receives a not-acknowledge frame from T-Dash, FOTA procedure is aborted but it can be executed again at the next power on of the motorbike. Below is the reply frame format.

TargetID	Length	Positive/Negative reply	SessionType	NACK

To reduce the time required by the firmware transfer, the Transfer data frame is structured slightly different from the other frames and it does not require a reply from the destination until the end of the block. Below is the structure of the Transfer Data Frame:

B0	B1	B2	B3	B4	B5	B6	B7
TargetID	ServiceID	Counter	D0	D1	D2	D3	D4

Where:

- Counter is a frame identifier which helps keeping the right order of the transmitted data and ranges between 0x00 and 0xFF.
- Data field for Start upgrade session frame contains firmware dimension in Bytes; this is useful to let the destination unit verifies if it has enough free space in memory and to be able to detect missing blocks at the end of the transmission. The destination board will initially store the firmware in an SD Card and at the end it will move it into the flash memory.
- Data field for Transfer request frame contains the block dimension in Bytes and it will be sent before each block. CRC Verification frame is sent by C-Node to T-Dash after transmission of each block and it contains the CRC-16 calculated over the block's data content. Once received the whole block, T-Dash will calculate the same CRC and it will compare it with the received one. In case of corrupted or missing data, the two values will differ and T-Dash will send a not-acknowledge frame to the C-Node, aborting the FOTA procedure.
- Stop upgrade session frame indicates the end of transmission while End of upgrade firmware procedure frame identifies the end of the procedure. Once received the last FOTA frame, C-Node can update Server with the new current firmware for the vehicle.

Using the above integrity protocol (and associated CAN frames format), the downloaded firmware is broken down into several transmission blocks of 4000 bytes. But before transmitting each block, secure FOTA protocol implements a command to inform the receiver ECU about the amount of bytes of the data block that will receive.

After the block reception of T-Dash, it will also receive the corresponding CRC to validate the current block by calculating the a new CRC of the block and comparing it with the one received by the Connectivity Node. If these are equal then the receiver will answer positively to the transmitter, that will continue the transmission; otherwise the firmware update will fail. The CRC frame identify also the end of each single block transmission. If the T-Dash respond with a positive acknowledgement then, the Connectivity Node will start immediately the transmission of the next blocks, if available; otherwise it will conclude the FOTA procedure. Once the full firmware is transmitted, the protocol implements a command to notify the receiver that the firmware transmission is completed. The receiver should answer positively to let the transmitter validate the firmware transfer. After the firmware transmission, the T-Dash will reset itself to perform the firmware upgrade.

3.1.12 T-Dash Firmware Update. Firmware upgrading is based on a simple yet effective technique. When T-Dash (re)boots, a special bootloader is executed before the actual firmware. The bootloader checks if the SD Card of T-Dash contains a firmware. In case there is not any firmware available, then the system starts immediately, else the bootloader fully erases the flash memory of the board and copies the new firmware from the SD card in the Flash memory. After completion, the bootloader executes (jumps to) the new firmware.

3.1.13 Generate and Submit a Vehicle Version Manifest to the Director Server. After completion of the firmware upgrade the Connectivity Node must inform the Director Server about the currently installed image on the T-Dash. To do so, it must generate and submit a Vehicle Version Manifest.

As shown in Fig. 8 the Connectivity Node starts by generating an ECU Version Manifest which consists of the ECU Version Manifest body and its signature. The ECU Version Manifest body contains the filename, file length, and the SHA256 and SHA512 hashes of the currently installed firmware image on the T-Dash as well as the ECU ID of the T-Dash. The signature of the ECU Version Manifest is generated according to the keys of the T-Dash. The Vehicle Version Manifest consists of the Vehicle Version Manifest body and its signature. The Vehicle Version Manifest body contains the ID of the vehicle, the ECU ID of the Connectivity Node and the previously generated ECU Version Manifest. The signature of the Vehicle Version Manifest is generated based on the keys of the Connectivity Node. The Connectivity Node encodes the Vehicle Version Manifest to a DER octet stream and then encodes it to Base64 format. The Base64 Vehicle Version Manifest data are encapsulated in a XML/RPC structure and transmitted over a HTTP Post to the Director server.

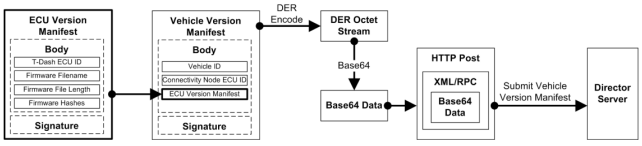

Figure 8: Vehicle Version Manifest generation and submission to the Director server.

4 SECURITY ANALYSIS

Security attacks fall into four categories according to the attacker's objectives:

(1) Read updates in order to reverse-engineer the firmware's contents. One way to achieve this is through eavesdrop attacks where an attacker may read unencrypted firmware updates transmitted from a server repository to vehicles.
(2) Deny updates where attackers prevent vehicles from fixing software issues. There are four types of attacks for this category.
   a Drop-request attack: blocks network traffic outside or inside the vehicle to prevent an ECU from receiving any updates. In other words, attackers refuse to serve request for software updates.
   b Slow retrieval attack: attackers do respond to requests, but so slowly that they never finish or error out. As a consequence, a known security vulnerability can be exploited by the attacker in the meantime.
   c Freeze attack: An ECU receives indefinitely the last known update, even if there may be newer updates on the repository.
   d Partial bundle installation attack: cause different ECUs to not install the latest updates. Attackers can do this by dropping traffic to some ECUs.

(3) Deny functionality of the vehicle with one of the following ways:
   a Rollback attack: Attackers cause an ECU to install obsolete software with known vulnerabilities. An example would be to send version 1 of the firmware for an ECU that has version 10 installed even though version 11 exists on the repository.
   b Endless data attack: Attackers induce failure by executing an endless data attack, where they cause an ECU to crash by sending an indefinite amount of data, and thus making it run out of storage.
   c Mix-and-match attack: Here, attackers also cause failure of ECUs to interoperate by creating new bundles of images, and cause ECUs to install an arbitrary combination of new versions of images.

(4) Control an ECU by installing software of the attacker's choice.
   - Arbitrary software attack: Attackers overwrite the software on an ECU with malicious software.
   - Remote exploit: Attackers may compromise an ECU using a programming error such as buffer or heap overflow.

The capabilities of an attacker that should be taken into consideration are the following:

(1) Intercept and change network communications such as man-in-the-middle attacks. These actions can be accomplished with the following ways:
   - Outside the vehicle: an example could be an attacker that controls a cellular network used to distribute updates.
   - Inside the vehicle: for example, the attacker could control communications over a gateway ECU, normal ECU, OBD - II port or USB.

(2) Compromise ECUs in a vehicle.
(3) Compromise keys that are used to sign software updates or the servers that store those keys.

Secure FOTA incorporates several features to solve security weaknesses that may compromise a vehicle during one of the previously mentioned attacks. These features are:

(1) Using additional storage to recover from endless data attacks. Man-in-the-middle attackers can execute endless data attacks on ECUs that have only enough space to keep only one image. Thus, if these attackers send an ECU random data instead of an actual image, then it is unable to boot to a working image even though the ECU can verify that that the random data does not match the latest downloaded metadata. In order to solve this problem, Secure FOTA uses enough storage to maintain not only a previous image, but also the latest downloaded. This allows an ECU to be updated without having a previous known good image overwritten. Thus, if attackers have tampered with the image during transmission, and have not remotely exploited the ECU, then it may still boot to a functioning image.
(2) Using a vehicle version manifest to detect partial bundle installation attacks. Even if there are no attacks on ECUs, sometimes partial bundle installation attacks may accidentally happen because only some ECUs successfully installed the last set of updates sent by the director. Regardless of how these events happened, an OEM can detect these attacks using a vehicle version manifest which is signed by the Connectivity Node about what the T-Dash has currently installed. After verifying and installing an update the primary would use the signature of the T-Dash to build the vehicle version manifest, and send it to the director whenever it requests updates. If the director detects a mismatch between the last updates sent to the vehicle compared to what it has actually installed, then the OEM may be alerted about this for further follow-up.
(3) Using a time server to limit freeze attacks. Attackers can execute freeze attacks on ECUs because unlike desktop and server machines, ECUs do not typically have reliable real-time clocks. Thus, the primary may not be able to tell whether a metadata file has expired. In order to solve this problem, the primary sends a token to the time server on behalf of the T-Dash. Then, the time server signs the current time with the token, and sends the response to the primary. After receiving the current time, the primary verifies the signature of the time server using its public key and stores the updated time. Whenever the primary verifies the latest downloaded metadata, it would check whether the metadata has expired using this latest downloaded time. Note that if the primary includes its latest known time in building the vehicle version manifest, then the director can detect whether the primary is indefinitely replaying the same vehicle version manifest.

5 PROTOTYPE DEMONSTRATION PLATFORM

As described in the overall organization (see Fig.2) of a prototype CPS platform for vehicles, we developed the proposed secure FOTA process targetting the STM32F779NI MCU as the gateway – connectivity node – that is extended with an ESP8266 WIFI unit and STM32F769 as the T-DASH. Fig. 9 shows a snapshot of the process; the connectivity node (bottom left) has completed the authorization steps with the servers and transmits the fresh firmware to the T-DASH ECU (bottom right) according to the protocol presented in paragraph 3.1.11.

The memory footprint of the secure FOTA inside the Connectivity Node is almost 230 KB.

By using the Data Watchpoint and Trace (DWT) counter in the STM32F779NI-based gateway, Table 1 summarizes the computation delay of keys and signatures generation. Additionally, considering as a reference the Root Metadata in table 2 we list the mean latencies for encode and decode functionality.

Table 1: Keys and signatures generation mean latency in STM32F779NI MCU vehicle gateway

Function	Size (bytes)	Latency (μsec)
RNG Generate Private Key	32	191
ED25519 Generate Public Key	32	15489
ED25519 Generate Signature	64	15979
ED25519 Verify Signature	64	48152

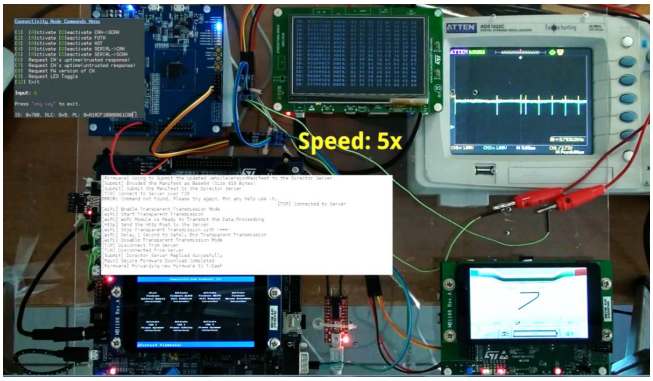

Figure 9: Prototype platform performing secure FOTA; Servers' output is overlaid in the snapshot window over the connectivity node; T-DASH screen displays the percentage (7%) of received firmware.

Table 2: Encoding and decoding mean latency with regard to Root Metadata

Function	Size (bytes)	Latency (μsec)
ASN1 Decode	651	1123
ASN1 Encode	651	3970
SHA256	651	591
SHA512	651	1959
Base64 Encode	651	152
Base64 Decode	868	477

Although security solutions for FOTA have been proposed [1], [4] to the best of our knowledge our prototype platform is the first actual complete end-to-end secure realization.

6 CONCLUSION

Today vehicle continues its transition from a hardware-driven machine to a software-driven electronics device with ever larger innovations, from efficiency to connectivity to autonomous driving to electrification and new mobility solutions. In this scope we presented a secure FOTA framework to provide multi-layered protection to enhance vehicle security. The developed solution allows to dynamically add new functionality by installing or updating apps, while providing complete security control over the apps offered and deployed to any particular in-vehicle device. This assists the system vendor in ensuring that apps even of the highest criticality are securely deployed to any of their systems.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union (EU) Horizon 2020 project TAPPS (Trusted Apps for open CPSs) under RIA grant No 645119. This work has also been partially funded by National Matching Funds 2017-2018 of the Greek Government, and more specifically by the General Secretariat for Research and Technology (GSRT), related to EU project TAPPS (No 80519).

REFERENCES

[1] K. Daimi, M. Saed, S. Bone and M. Rizwan, "Securing Vehicle ECUs Update Over The Air," In *12th Twelfth Advanced International Conference on Telecommunications*, 2016, pp. 45-50.

[2] G. Kornaros, E. Wozniak, O. Horst, N. Koch, C. Prehofer, A. Rigo, M. Coppola, "Secure and Trusted Open CPS Platforms", in book *Handbook of Research on Solutions for Cyber-Physical Systems Ubiquity*, Ed.: N. Druml, A. Genser, A. Krieg, M. Menghin and A. Hoeller, IGI Global book series Advances in Systems Analysis, Software Engineering, and High Performance Computing (ASASEHPC), 2017.

[3] G. Kornaros and S. Leivadaros, "Securing Dynamic Firmware Updates of Mixed-Critical Applications", In *3rd IEEE International Conference on Cybernetics (CYBCONF)*, 2017, pp. 1-7.

[4] K. Kuppusamy, Trishank, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch, and J. Cappos, "Uptane: Securing software updates for automobiles," In *14th Embedded Security in Cars (escar)*, 2016.

[5] H. Mansor, K. Markantonakis, R. N. Akram and K. Mayes, "Don't Brick Your Car: Firmware Confidentiality and Rollback for Vehicles," In 2015 10th International Conference on Availability, Reliability and Security, 2015, pp. 139-148.

[6] M. Wolf, A. Weimerskirch, T. Wollinger, "State of the art: embedding security in vehicles," Eurasip Journal on Embedded Systems, 2007.

[7] R. Petri et al., "Evaluation of Lightweight TPMs for Automotive Software Updates Over the Air," In 6th escar USA, 2017.

[8] A. J. Poulter, S. J. Johnston and S. J. Cox, "SRUP: The secure remote update protocol," 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), 2016, pp. 42-47.

[9] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable key compromise in software update systems," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10, 2010, pp. 61–72. [Online] Available: http://doi.acm.org/10.1145/1866307.1866315

[10] C. Smith, "The Car Hacker's Handbook: A Guide for the Penetration Tester", 1st ed., No Starch Press, 2016.

[11] M. Steger, A. Dorri, S. S. Kanhere, K. Romer, R. Jur- ÂÍ dak, and M. Karner, "Secure wireless automotive software updates using blockchains: A proof of concept", in *Advanced Microsystems for Automotive Applications* 2017, pp. 137-149, Springer, 2018.

[12] Windriver Systems Inc., "Implementing Over-the-Air Software Updates for Automotive Applications", Whitepaper, [Online] Available: :http://events.windriver.com/wrcd01/wrcm/2017/11/Over-the-Air-Updates-for-Automotive-White-Paper.pdf, 2017.