Software Over the Air Update for Modern Software Architecture

Abir Bazzi[1], Dr. Di Ma[2], and Dr. Adnan Shaout[3]

Abstract— **Installing a software or firmware update on an electronic control unit (ECU) involves many tricky operational and security aspects in the process. The most critical parameters for the application of software over the air (SOTA) update in safety-critical automotive domains are safety, security and time. Code signing of software image is the most common security mechanism used in SOTA. While this mechanism introduces a signature appended to the software image to validate its integrity and authenticity, applying it to new automotive architecture where the software running in the ECU is not anymore a single image binary coming from one vendor but a set of images coming from different independent vendors is not free of new challenges. Existing solutions are based on signing each software image individually with a dedicated key pair per image. We propose a new solution to reduce the used keys and provide the necessary security for the ECU while receiving images from different vendors. The proposed approach, working at the ECU level, complement the existing SOTA solutions by implementing a new Merkle Tree-based algorithm to additionally address dependencies and conflicts among the software entities within the ECU using a logical implicit signature verification without adding much overhead to the process and while keeping key management in the vehicle compliant with the current process aiming one key set for SOTA services regardless of how many software images and vendors exist within the ECU.**

Keywords: Software over the air, Safety, Security, Autosar, Hypervisor, Virtualization, Distributed software development.

I. INTRODUCTION

Autonomous driving, connected cars, and shared mobility have dominated the automotive industry in recent years. Such trends significantly impact the architecture model of the vehicle. While these innovations, built on the digitization of in-car systems, offer great advantage as well as create new services and business models for automotive industry, they come with the risk of attacking safety and cybersecurity of modern vehicles. Software and hardware components are and will continue to be among the key innovations in modern vehicles. The software development market is expected to grow steadily over the next few years to reach USD 5.3 billion in 2030[22]. ECU software is becoming more similar to mobile software when different packages are provided separately and need to be integrated together in the car based on certain criteria. Virtualization, Hypervisor, Adaptive and Flexible Classic Platform are example of architectures which splits today's binary image into several software binary images which can be independently developed, integrated, tested, released and programmed on the target ECU. Providing software patches throughout the full vehicle lifecycle is essential for safe vehicle operation. If attackers are able to implant malicious code on a vulnerable embedded software in the vehicle, they can have access to critical functionalities and mount further attacks that can put humans life at risk. The well-known Jeep UConnect attack [8] was caused by a software update vulnerability in one of the ECUs in the vehicle.

Software over-the-air update has gained high interest in the automotive industry. An ECU can only run a new version of received software after empirically verifying that it has successfully received the entire correct image file from the server (directly or through the gateway). The main security mechanism used for SOTA is code signing for each software image build for the vehicle as it is critically important to verify that the contents in the image have not been tampered with, as well as to verify that the received image is from the intended publisher. Code signing is a technology which uses digital signature and the public key infrastructure to sign image files. The integrity of the system relies on securing both the software image as well as the keys against outside access. Lots of methods have been proposed, standardized, and adopted by automotive industry. Such solutions range from full OTA frameworks (Uptane[13], eSync solution[5]) to approaches based on online/offline and symmetric/asymmetric keys([34],[25],[33]), cryptographic hash function ([24]), full and partial verification ([19],[29],[28]), hardware and software implementation([42]), and many other proposals. These solutions have been performed on ECUs with single image binary per ECU. Dependencies and conflicts among software image entities are mainly managed by original equipment manufacturers(OEM) or the Director repository by Uptane specification, and there is no way currently for the ECUs themselves to detect such conflicts once software is installed in the vehicle. Given the new software architecture (which decomposes an ECU image into independent, distributed and somehow connected blocks or software clusters ([45]), Code signing becomes more complicated and demands more for resources (memory and processing overhead) as well as coordination between the publishers and the endpoint ECU in the vehicle. Each software vendor requires a shared key which should be setup and stored in the ECU where the software will be running. ECU has to make sure the software entity belongs to the corresponding module and configuration. Blockchain based approaches [31],[20],[44],and[23] are a promising solution for such distributed system. However, the implementation of Blockchain requires lots of efforts for OEMs as well as high resource consumption at the ECU

[1]Electrical and Computer Engineering Department, The University of Michigan-Dearborn, Dearborn, Michigan (aybazzi@umich.edu)

[2]Computer and Information Science Department, The University of Michigan-Dearborn, Dearborn, Michigan (dmadma@umich.edu)

[3]Electrical and Computer Engineering Department, The University of Michigan-Dearborn, Dearborn, Michigan (shaout@umich.edu)

level. Therefore, our work aim to optimize code signing and key management needed for such new architecture to meet the automotive ECU constraints as well as provide the necessary security to protect the vehicle against both passive and active attacks during software over the air updates. In section 2, we will provide an overview about new software architecture and SOTA in automotive. In section 3, we explain the common process of SOTA and the challenges for new architecture, and then followed with our proposed solution in section 4.

II. Background

A. Software Architecture - New trends

Driven by ADAS and Autonomous Driving, OEMs must bring together software subsystems into a larger system and ensure that the developed functions meet specifications and fulfill their purposes consistently and reliably as well as software shall be security assessed, authenticated and integrity protected. Support required by new automotive systems includes dependable architecture with fail-safe operational systems, cross-domain platforms, high-performance micro-controllers and computing, distributed and remote diagnostics, as well as cloud interaction. To make this possible, OEMs are in need of new software architecture to leverage the capabilities of multi-core processors as well as flexible and hybrid zonal topology. These processors bring additional hardware support for virtualization and necessary processing rates.

AUTOSAR started the development of the Adaptive Platform in 2017 to address current demands of automotive software, with its powerful central computers and Ethernet-based and service-oriented communication to provide an optimal foundation for future vehicle architectures. This new platform architecture requires dynamic deployment and update of adaptive applications as well as firmware. In addition to the Adaptive Autosar, modern vehicle architectures require still microcontroller based on AUTOSAR classic control units due to strong real-time and high safety requirements. The current AUTOSAR classic platform is developed for building one complete executable flashed at once into the microcontroller, and thus does not well support those requirements for more flexibility and independence. A newer AUTOSAR Flex concept[4] is introduced to split today's solid image binary into several binaries representing several software clusters independently developed, integrated, tested, released and programmed on the target ECU.

Virtualization is another innovation technology for hard real-time automotive applications [3]. Hypervisor allows multiple Virtual machines (VM) to execute together over a shared hardware platform but in isolation from each-other. Hypervisor plays an important role as ECUs are getting more and more consolidated in the new automotive architecture[6]. Hypervisor enables an independent software development environment for each Virtual machine (VM) development, and the responsibility of such virtual ECU (vECU) development could be distributed across different suppliers or even different teams within an OEM. Each vECU have eventually its own vECU binary. It is also perceived that AUTOSAR applications are most likely to be a VM application, and thus AUTOSAR application integration will be configured on top of hypervisor.

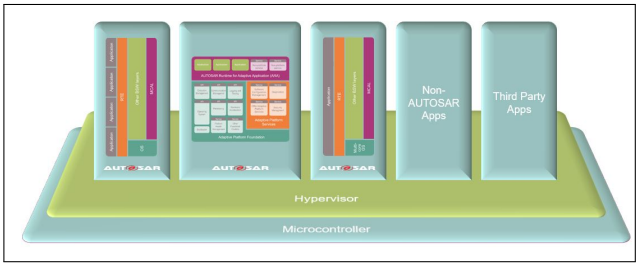

Fig. 1. Software Architecture

It is worth to mention some other factors enabling the split and isolation of software functions or applications in automotive. The number of ECUs in domain or zone architectures is being reduced without compromising on features, and thus more applications are consolidated in one hardware. OEMs target the decomposition of their software into different entities to fulfil different requirements such as combination of safety (including separation of ASIL levels) and QM applications in same ECU, as well as security related applications, flexible activation and deactivation of features, better liability management, diversities in software providers, and agile software development with split software units among teams and organizations. Software split support is mainly required nowadays to enable cost and performance benefits to the customers. As result, suppliers and OEMs should be able to integrate multiple applications into single microcontroller in the next generation ECU as abstracted in Figure 1. Lower response times are critical to satisfy hard real-time constraints of typical automotive ECU system. These applications should then be programmed separately in flash memory and thus enabling granular updates of applications during software over the air updates.

B. Software-over-the-air Overview

There are two types of Software update: (1) Local update and Remote update. The local update refers to the traditional technique where the users bring their cars to the service centers or dealerships and software is updated using dedicated tools through OBD connection. (2) The Remote (Over-the-air) update refers to the technique where the software is sent to the vehicle through wireless communication while the vehicle is running. OEM's desire is to minimize the impact on drivers with no down time when performing software OTA, there should be no risks of leaving vehicle unusable in any ways after the update. OEMs find many benefits for implementing software OTA updates: (i) The ability to update software in a fast and Cost effective–without the need for physical recalls of the cars, (ii) The opportunity to rapidly respond to bugs and security vulnerabilities, and (iii) The ability to generate new revenue by adding new features after an ECU has been deployed in the field.

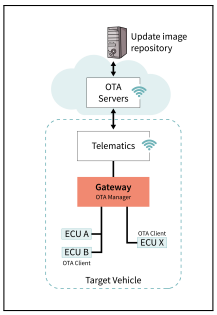

Fig. 2. Software OTA update architecture

Figure 2 shows an example of a typical implementation of software OTA in current vehicle architectures. The gateway acts as the SOTA master within the vehicle, it receives through the Telematics/Gateway module the software image from the OTA server and then distribute it to the target ECU.

C. SOTA Update Threats

The automotive industry is actively working on proactive measures necessary to secure the vehicles and ensure that drivers and passengers remain safe. Recent years have seen an alarming spike in the number of cyber attacks targeting the automotive industry. Based on the Upstream's research reports about the automotive cyber incidents occurred in 2020 & 2021[14], 87.7% threats are related to vehicle data/code, 50.8% potential vulnerabilities that could be exploited if not sufficiently protected or hardened, 24.1% threats regarding back-end servers related to vehicles in the field, 4.3% threats to vehicles regarding their update procedures. SOTA itself comes along with security threats which is also considered as safety threats since an inadequately protected SOTA implementation, which allows potential attackers to manipulate safety-critical vehicle applications, can endanger the overall safety of the vehicle and in the worst case the lives of its occupants[10].

ECU software is becoming more similar to mobile software when different packages are provided separately and need to be integrated together in the car. The main challenges for automotive industry are the real time aspects and resources constraints for how to check in the vehicle as well as remotely that these individual updates are trusted and valid. OEMs consider an ECU more like as a remote test unit where the software packages have to be validated remotely to make sure ECUs have always the latest and greatest software package without any changes.

A system is only as secure as its weakest link. An inadequately protected SOTA implementation, which allows potential attackers to manipulate safety-critical vehicle applications, can endanger the overall safety of the vehicle and in the worst case the lives of its occupants. A connected vehicle is exposed to both malicious software manipulation as we see in the IT world as well as physical attacks at hardware level:

- Attacker can introduce manipulated software to the vehicle
- Attacker can prohibit the update of a package.
- Attacker can downgrade to an older version that contains vulnerabilities.
- Attacker can introduce a personalized software package intended for one specific car to another car.
- Attacker can manipulate key material for authentic communication between backend and vehicle, or for signature verification of the vehicle package.
- Attacker can cause inconsistency between current vehicle configuration and current backend server information.

To prevent such attacks, a "defense in depth" approach is needed. With split software architecture, the OEMs are really hesitant to let any third party vendors install any piece of software into their ECUs. When considering evolving cybersecurity approaches, relevant standards and national legislation are perhaps the most important market drivers for cybersecurity approaches (ISO/SAE 21434, UNECE WP.29, ISO/AWI 24089, Cybersecurity Best Practices by NHTSA, Uptane IEEE-ISTO 6100.1.0.0 and many others). Several papers have already presented concepts for a secure software update process as well as the possible threats that such systems face. We will present in the next section the current SOTA strategy used in automotive, followed by a summary survey on the current proposed SOTA solution.

III. SOTA Process in Automotive

The SOTA process is usually carried out in several successive steps as shown in Figure 3: it starts with the software developer, usually a supplier or the OEM itself, with releasing a new software image, and then sign it using Public Key Cryptography (PKC). There is a key pair consisting of a private key (Prv-k) and a public key (Pub-k) that are associated with each software image update. Only the signer (OEM/Supplier) has access to Prv-k (protected in a secured environment) whereas Pub-k can be publicly distributed and deployed in the ECUs. The image code (x) is first hashed to a short fixed length h(x) (e.g. 32 bytes calculated using SHA256). Then, a digital signature is computed over the hash value using the private key Prv-k.

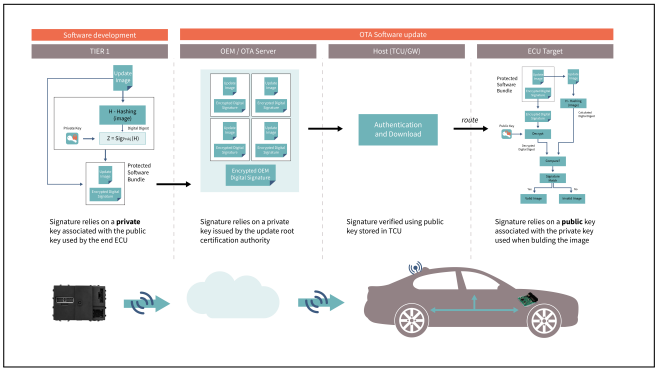

Fig. 3. Software OTA update implementation in automotive

Once the software package is received by the ECU end target, a hash function (e.g. SHA-256) is applied to the code image to calculate the hash or digest. Next, the digital

signature is decrypted using the stored public key, mostly computed in a security controller (HSM or TPM) which provides tamper-resistant features in accordance with the Common Criteria (CC) security standard. The calculated digital digest and the decrypted digital digest are then compared for an exact match. If the signature verification is successful, the downloaded image file is accepted and activated. It is worth to mention that it is optional to encrypt the image before sign it. It is important to track installation of updates to ensure that ECU have been updated successfully, and in case of failures, the ECU should also be resilient and be able to recover from failures. ECU manufacturers may wish to monitor remotely which versions of software and confirm that latest unaltered software is executed in an ECU. To enable remote attestation, ECU needs to communicate with the server performing similar verification as the one performed during the SOTA updates. As a result, ECUs with old software should be triggered and upgraded to the newest versions. ECU with malicious or unknown software should be isolated for forensic inspections if it is not possible to locally roll back to a previous working software.

A. Existing Software Update Solutions

In this section, we provide an overview of existing work related to software update and distribution in automotive industry as well as similar industry. Consumer electronics, commercial aviation and medical devices are adjacent industries that have many hardware devices containing loadable firmware components, and have the task to track and verify the installation of the software updates across their fleets. In the NHTSA report [18], the authors present a literature review of the state-of-the-art of software updates in the industries related to automotive. While there is no single and perfect reference model for securing software updates due to different requirements and user experiences of these systems, there are mainly 2 common existing defense mechanisms: trusted content distribution network, and digitally signed software update.

In Avionics, software is transferred into a Loadable Software Aircraft Parts (LSAP), within the Electronic Distribution of Software (EDS) Crate [1], loaded using different types of onboard loaders. Once manufacturer provides and loads LSAP updates to their repositories, they notify airlines which have to retrieve and apply such updates. A professionally trained technician will then trigger the transfer of the LSAP to the aircraft via a local secure wireless connection. The ARINC 835 (Guidance for Security of Loadable Software Parts Using Digital Signatures) report [2] describe the standard to use digital signatures for software distribution to the aircraft. There is a difference on how airlines used digital signatures (either one or multiple signatures).

The internet of things (IoT) devices have similar resource constraints (e.g. energy, computation, and storage capacity) to the automotive ECUs. Several standards groups and consortia have issued documents on secure software and firmware update. The Trusted Computing Group (TCG) has released a reference report [16] describing how secure software and firmware update for embedded systems can be done using Trusted Computing technologies. Each updated software should be provisioned with its manufacturer's public key, or a hash of the manufacturer's public key. The Internet Engineering Task Force (IETF) Software Updates for Internet of Things (SUIT) working group is actively working on specifying a software update architecture for IoT devices. The current informational RFC9019 [9] describes the architecture and security requirements for firmware updates of IoT devices and standardize the manifest files used for the update process.The pre-authorisation step involves verifying whether the entity signing the manifest is indeed authorized to perform an update. The authors of [31], [20] , [44], and [23] suggests a blockchain-based firmware update for IoT devices where different nodes are defined with different roles including the verification node managed by the device manufacturer and include the main information about the firmware. The main drawback is the IoT device (as a Blockchain node) needs to store a copy of Blockchain's ledger in the devices which can be challenging due to limited resources of IoT devices, as well as not suitable for heterogeneous IoT ecosystem.

The automotive industry has already started to follow the user and access Level and interaction attributes used in consumer electronics. With Tesla [11], the user can check for new software updates by using their touchscreen to install the update immediately or schedule for later. As shown in Table I, we can categorize SOTA solutions in automotive based on the security mechanisms used by the researchers. They all follow the process described in the SOTA Process in Section III.

B. Challenges of existing solutions

Lots of methods have been considered, standardized, adopted by automotive industry. From one side, Uptane and other proprietary solutions have been introduced to manage software update process. On the other side, AUTOSAR and new automotive trends have defined the architecture to manage software update needs within an ECU. Uptane is a comprehensive and robust solution to securing SOTA on vehicles. Starting with Uptane specification, dependencies and conflicts between ECU software image entities are mainly managed by the Director repository. Uptane is only a framework for software over the air. It assumes all should be done properly and there should not be any backdoor where the system can be compromised. If the Director repository is not careful in updating software images, the vehicle may end up installing conflicting images that will cause ECUs to fail to interoperate [15]. There is no way currently for the ECUs to detect such conflicts once software is installed in the vehicle. Same concept apply to dependencies and conflicts between the software entities within an ECU software image. If the repository is also not careful in installing any unintended software entities, the ECU may end up installing malfunctioned software that will cause an ECU to malfunction, for example an attacker takes control over an application by buffer overflows causing malfunctioned

execution in the module. If the ECU is not able to detect such error, then the attacker is able to influence all the other applications, the attacker can exhaust resources (e.g. CPU memory), can call interrupts, or can misuse interfaces exposed by the other applications.

As mentioned earlier, software updates need to be applied across a distributed system of automotive devices, which are designed and serviced by different suppliers. OEMs has to shift from the traditional vehicle architectures toward the more flexible and scalable approach required for the next generation architecture, thus, a single ECU integrates software and hardware dynamically from multiple vendors. The new software architecture leads to changes in the software release as shown in Figure 4. Uptane allows software image to be available on a private repositories - not just limited to OEM ones. OEMs already announced the implementation for application-based software, not just for infotainment and media but also for traditional functionality such as tuning services (e.g. [7]). Thus, software images will be saved on directories that might not be managed by OEMs or standard suppliers.

In order to support the individual software update to each application binary, each image needs to have its own digital signature created with a dedicated key pair (private and public keys). if any of the keys is compromised for such images, the repository (director in case of Uptane framework) might not able to detect such issue and a malfunction software can be installed in the ECU, thus, vehicles becomes as vulnerable to attack by malicious parties as any other smart devices.

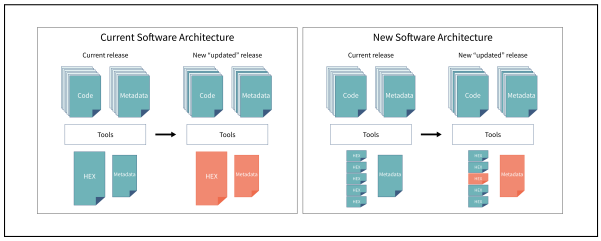

Fig. 4. Software release process: current versus future

Regardless of the used SOTA approach, certain features are common to the process. Most approaches aim to ensure software safety and security using digital signatures, hash, cryptography, etc... There is a need that the the microcontroller establishes a root-of-trust, and includes a handler which verifies integrity, authenticates and validate the image at reception through the signature. As much as it is important to sign image and related metadata, the security of ECUs depends on precisely how it is signed (e.g. online keys vs. offline keys, symmetric vs. asymmetric keys) and how protected are the signing keys. This choice is effectively a performance vs. security trade-off. Before any SOTA service can take place, the OEM should provide the production signing verification keys for each ECU. Each ECU/supplier is assigned only one public key to verify the signature of any software packages programmed on that ECU. Multiple methods of public key deployment and storage exist based on OEM processes and ECU/microcontroller capabilities. The keys are commonly generated using public key infrastructure (PKI) and then public or shared key is injected in ECUs at the assembly line, or compiled with the ECU primary bootloader software of the ECU, and can be updated/changed via bootloader update. Diffie-Hellman (DH) or EC Diffie-Hellman (ECKA-DH) are commonly used as key agreement procedures for symmetric keys.

Key management has many functions, such as key generation, key distribution, key update, key storage, and key destruction. Key management and security is extremely important through the whole lifecycle of the vehicle. Confidentiality, authenticity and integrity of software update are ensured if shared keys are properly secured. Extraction and cloning of keys should be prevented. Any compromising for the keys means no security for the vehicle. If an adversary is able to read out or replace a public key, he might be able to manipulate code or prevent software updates. The keys must be kept confidential over the vehicle entire life cycle. Keys must not be transmitted unprotected over internal ECU hardware devices or over networks external to the ECU. Thus, each ECU should have its keys stored locally. Protecting the keys is crucial, and protection can be achieved only by using secure memory and applying hardware-assisted approaches employing a security anchor (e.g. HSM, TPM)[39].

The use of keys ranges over the entire vehicle life cycle, from initial keys used during development, until production keys used to update a new software or activate a new feature, including the establishment of these keys in the ECU. Each ECU should deal with the whole cycle of a key. Since there was no standardization of key management previously, but solutions were needed, automotive OEMs followed mainly individual approaches. An initial step towards standardization was the KeyM module[12] introduced with AUTOSAR 4.4. It does not standardize a specific key management strategy, rather it offers generic interfaces for implementing various strategies. The automotive community should focus more intensively on harmonizing strategies for key management. Whatever the used SOTA approach, each software supplier will require a unique set of keys, setup these keys have to be done and implemented into the ECU before any software download can be done. In some cases, a key set should be used for a limited time period, and should be determined how often the keys should be changed to achieve a sufficient level of privacy. If software application is transitioned from one ECU to another ECU, keys have to be destroyed at old ECUs and new setup has to be done in the new ECU.

The hardware security module(HSM) and trusted platform module(TPM) enable secure storage of keys and acceleration of cryptographic computations. A protected non-volatile memory is required to store the keys in each ECU. Each key usually requires some additional protection bytes used to protect the memory slot against failures as well as replay attacks during update. In general, a limited number of write cycles is supported and guaranteed per memory slot due to the physical memory write endurance. How-

ever, the ECUs are already facing challenges on the key handling and management for all existing keys needed for used security applications within the vehicle (Debug access, SecOC, Flash protection...). In addition, newer security mechanisms have been adopted by OEMs and recommended by standards such as intrusion detection, remote debugging, IpSec, Plug&Charge (ISO15118)...These applications come also with the need of using crypto keys. Thus, one of the main challenges for SOTA for new software architecture lies in the key management needed for each application software binary code.

The process to setup, distribute and update each shared or public key adds lots of overhead for the vehicle in addition to the memory resources. Thus, we need to come up with a method that allows each update to be secure as well as independent from other application software clusters while using the same hardware devices and resources.

IV. Proposed Solution

Our proposed solution is to extend current solutions by a new approach to address the new software architecture including dependencies and conflicts among the software entities as well as create a logical implicit verification without adding much overhead to the process and ECUs while keeping key management compliant with the current system aiming one key set for SOTA services regardless of how many software applications and vendors exist for the ECU software. The proposal will ease the implementation and reduce complexity in the ECU, and at the same time, shift more control from the vehicle to the server repositories.

Our proposed solution is based on using the Merkle Tree approach to calculate the signature to update the software within an ECU. Each ECU is allocated a key pair for software update task - same as the traditional process where software image application is a single file per ECU. This key pair is used to sign the Merkle tree Root created with the hashes or message digests of each software entity block residing on the ECU. With this approach, a single key is still used to update multiple independent software image binaries, making this a fairly efficient digital signature scheme for over the air update for new software architecture. Merkle (hash) tree based authentication[35] has been widely used owing to its lightweight computation and storage overheads with wide range of applicability[27], including blockchain technologies[32] as well Merkle trees have been used as basis of most hardware memory authentication and protection schemes[37][46]. The reliability of the Merkle tree is based on the collision resistance characteristic of hash function used to construct it. It is assumed that it is infeasible to find an image software of a given hash value within a computationally reasonable time[30], only the repository with the same software images can get the same Merkle tree. Thus, the security of this Merkle tree-based solution depends on the security of the hash function in use.

There are two parts to consider when proposing such solution. Along each image software, the OTA server needs to only send the signed Merkle Root with the software image. Once the ECU receives the software image, it constructs or retrieves the Merkle tree from the hashes of the software images, then it determines whether the result is the same as the one received from the server (after decrypting with the followed by constituting with the public key of the ECU). Once verified, the ECU stores the value of root node in the Merkle tree in a secure location. The software supplier, OTA server, and ECU have the information on the hash function to be used and the size of each image software prior to the verification. The Merkle tree based scheme is designed to work properly regardless of the number of verification or number of software components. This implies that the implementation of the Merkle tree schemes has to be implemented efficiently in a periodic manner, especially when the ECU has tremendous contents to validate. Based on ECU capabilities, the complete Merkle Tree can be saved in the ECU, so the verification can be done quickly. Upon a change of any software image, the correspondent hash (leaf node in the merkle tree) is changed. Thus, all the nodes on the path from that leaf to the root should be changed accordingly. For smaller processors (secondary ECU), it is possible that only the Merkle tree root can be saved in secure memory. Such ECU only stores the value of the root node of the tree and removes the rest of the data once the tree is constructed. Upon a change of any software image, the ECU has to re-create the complete tree to obtain the MT root. The decision to save complete Merkle Tree nodes versus just the root is a memory/cost versus signature verification time trade-off.

Uptane uses layered defense mechanisms so the security of automotive software updates does not degrade all at once, but is supported by a hierarchy in which different levels of access to vehicle's or the automaker's infrastructure must be gained. Software decomposition into independent components and outsourcing software to non-OEM (third-party) storage means that control of the software is delegated to the authority controlling the remote repository. Inadvertent software manipulations are possible as third-party repositories are expected to be less vigilant than the OEM repositories. By extending Uptane framework with our Merkle-tree-based digital signature, additional level is built into the security system to efficiently verify the integrity of the software image stored in a remote repository. Even if attackers compromise servers, gain access to third party software vendors, these incursions will be limited in how much damage they can cause to the ECU as our solution enables the ECU itself to detect suspicious updates software as well as the OEM to detect an ECU with wrong software or old versioned software. This Merkle tree-based approach enables verifying that the server and the ECU own the same software images and related ECU information. The server needs to generate the Merkle tree for the software components residing in the ECU, while the ECU has to calculate a few number of hash values to generate the new Merkle tree root and compare it with the received decrypted signature value.

The Merkle tree has two other features: the first is quick location modification. If a data node is modified, this affects

only the nodes from this node to the root. The Merkle tree's second feature is Zero-knowledge proof, which means that the prover can convince the verifier that a particular assertion is correct without providing any useful information. When a software image is updated, the ECU calculate its hash and update the correspondent leaf nodes, and with the remaining software information, the ECU calculates new root. Therefore, if the result is the same as the received root(decrypted signature), this can prove that this software image is valid, and there is no need to publish the information of other software components.

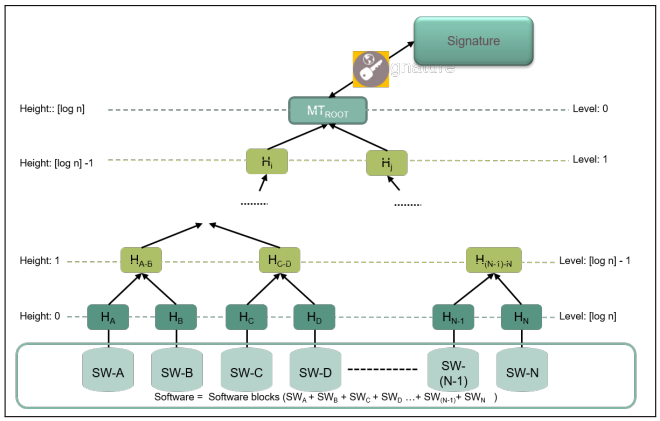

Fig. 5. Merkle Tree approach for software image digital signature

- SW-X: Software image component file downloaded from server and to be programmed in the ECU. There are a total of N components.
- H_x: Hash value of software component X. This represents the leaf nodes of the Merkle tree.
- $H_{y,z}$: Hash value of child nodes = Hashing (H_y, H_z).
- MT_{ROOT}: The hash value stored in the root node of the Merkle tree. This value is signed with the private key to create the signature.
- l(i): level of node i.

As show in Figure 5, each leaf node of this full balanced Merkle tree represents the hash of a software component(e.g. N is the number of leaf nodes or number of software components/blocks/modules). There is a single root MT per ECU software. This root MT will be compared to the decrypted received signature of the software image update to validate the software image update. To validate the integrity of a software component, all the intermediate nodes on the path from the correspondent leaf node to the root node need to be consulted. Thus, the validation depends on the number of levels of the tree. A smaller MT with fewer leaf nodes and fewer MT levels takes less time to verify a corresponding MT branch during a leaf node update (software block update). The more leaf nodes are, the more MT levels are used, and thus more MT branches to change and verify any updates to a software block. The number of tree levels is equal to $[\log_k N]$ where k is the arity of the tree.

The software components are expected to have different sizes, but assumed to be in sizes aligned with the micro-controllers flash sector sizes. As some software components are updated more often than others, it is efficient to consider using Multi-root Merkle tree (MMT) concept. With MMT, software components are arranged into groups. Each group has its own MT and correspondent root. These roots then represent the leaf nodes of another Merkle tree to calculate the Root used to validate the signature. The advantage of using MTT over classical single root MT scale logarithmically with the number of groups, Theoretically, a k-ary MT with leaf nodes arranged in n groups reduce MT level by $[\log_k n]$. For example, for a binary Merkle tree, the maximum depth from leaf to root is at most $[\log_2 n]$ for n software components blocks. Thus, for any software component update, it is faster to span and update the individual MT than the single-root MT for all software components, substantially reducing the MT reads/updates, and decreasing time to update and verify the MT root. Different designs for the Merkle trees can be used depending on the software architecture and intended performance. In addition to the full balanced tree and multi-Root Merkle tree, left/right/middle skewed trees[21] can be used. The main goal with skewed trees is to have shorter paths for software components updated more often, and eventually fast signature verification. There are different possible skewness levels e.g. skewed-by-two, etc.. With each level, the shorter paths gets shorter, while the longest path gets longer. The are 1/2 as many nodes on the short path as there are on the long path, so skewing the tree too much can actually hurt the performance if it is not designed carefully. In Figure 6, right skew-by-one Merkle tree and middle skewed-by-one Merkle tree are shown, N/4 nodes have path to the root shorter by 1 hop, N/4 have same path as for full tree and N/2 have a path that is 1 hop longer.

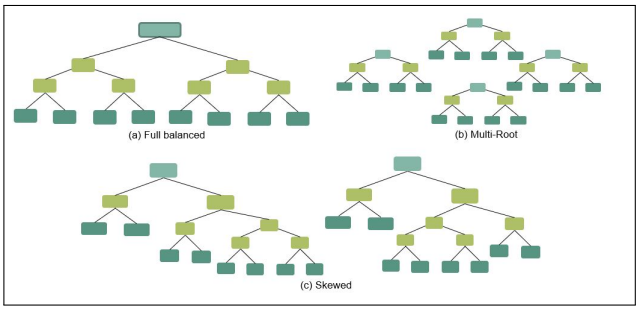

Fig. 6. Merkle tree design Types

Our proposed solution is also suitable for future quantum age. Quantum computers may break cryptographic algorithms in the future or decrypt data from the past (assumption is in 10 to 15 years). While the standardization of new post-quantum cryptographic schemes to replace RSA and ECC is still ongoing until 2024, hash-based signatures such as LMS and XMSS[26] are particularly suited for the protection of firmware updates due to their high security level using hash functions (e.g., SHA-2). Devices with over 10 years lifestyle must be prepared for the quantum computing range. Thus, post-quantum cryptographic is relevant for the automotive industry due to long lifetime of the vehicle and ECUs. Hashing-hardware may help (in general) and XMSS can greatly benefit from a SHA-256 accelerator. A merkle-based solution for SOTA allows OEMs to built in scalable solution across various platforms for different types of ECUs. It is believed that the usage of Merkle signature scheme is

resistant against quantum computer algorithms when used with sufficiently large security parameters.

V. CONCLUSION

Bruce Schneier, called a “security guru”, says that “complexity is the enemy of security. As systems get more complex, they get less secure.” This directly applies to our use-case where we are moving from securing one application image binary update per ECU to many independent images produced by diverse software parties and updated in the field. With our proposed approach, we tend to to reduce the key materials needed at the ECU level to update the individual independent software applications, smoothly add a new software image without setting up specific keys in the ECU, provide the necessary security for the ECU while receiving images from different repositories, as well as confirm compatibility of the new image with the target ECU avoiding any malfunctioning. advantages can be summarized into main three folds: the sensitivity to alternations, the tremendous amount of information that can be stored, and the ability to organize information recorded by packing them altogether. Our ongoing and future work focuses on implementation of such merkle tree based SOTA solution and study of performance improvements since not all software components are updated equally and frequently.

Approach	Description	Pros	Cons	Security Pillars
Message-digest / Hash based(e.g. [36])	Software package is hashed (e.g. SHA-2-256), and result is compared with a trusted known-good value. Different ways of hash chain have been proposed to increase security. Commonly used for Data Validation/Platform integrity/SecureBoot	Simple to implement. One-way only (computationally infeasible to compute the inverse). No used Keys. Can be software or hardware implemented (implementations are efficient and fast in Software). Memory efficient (typically 256bits)	Inefficient when there are many collisions. Known-good value must be Read & Write protected. Slower than CMAC if AES HW acceleration is used.	Integrity: yes Authentication : no Confidentiality: no
Symmetric-based[33]	Software package is encrypted with a common key,and then decrypted at the receiver side. Another approach is to generate a message authentication code using the common key (Hash-based message authentication e.g. SHA-256-HMAC or using Cipher-based message authentication e.g. AES-128-CMAC) and verify the same MAC code at reception. Commonly used for Stored data encryption / Encrypted communication.	Fast cryptography. Easily implemented (AES accelerators are widely available in microcontrollers.) Small key lengths. Example of used Symmetric Keys: AES: 128/192/256 bits. CHACHA20: 128+ bits. Modes of sharing: Diffie-Hellman and Physically.	Key security (key must be Read & Write-protected) Managing keys in this system is a challenge. Complexity consistent regardless of number of users or frequency of use	Integrity: yes Authentication : yes Confidentiality: yes
Asymmetric-based[34]	Software package is encrypted with a private key, and then decrypted at the receiver side with the public key. Another approach is to generate a code signature by hashing the software package and then encrypted using a private key (e.g. RSA 2048, Ed25519)	Most secure for SW updates. Key Flexibility. Relatively long key lengths. Example of used Asymmetric Keys: RSA-2048: 256 bytes RSA-7096: 512 bytes ECDSA (secp521r1):public key :1042-bitsignature : 132 bytes EdDSA:keys 32/57bytesSignature: 64/114 bytes Modes of sharing: PKI Same verification as a Secure Boot strategy.	Key security (key must be Read & Write-protected). Large keys (2kb), smaller keys for ECC (256b) Slower cryptography than Symmetric scheme Complex to implement for some algorithms (e.g. ECDSA and EdDSA) Complexity grows with number of users and frequency of use	Integrity: yes Authentication : yes Confidentiality: yes
Blockchain-based[40][17]	Distributed peer-to-peer database, Maintained by the network members. Data is saved on each node, Transactions are saved into blocks. Each block is chained to the previous block. Cryptography is based on Hashing and Digital signatures (using asymmetric cryptography e.g. ECDSA)	Immutable - (permanent and tamper-proof). Decentralized controlled. Redundant decentral copy on every node of the network. Consensus-based, creates trust and integrity in an untrusted environment	New to Automotive High memory consumption in the ECU	Integrity: yes Authentication : no Confidentiality: yes
Secure Software Repository Framework[13][15]	Framework introduced to allow repositories to build different security models that provide varying degrees of security and usability. Most recent proposed framework is Uptane : Multiple servers known as repositories are used. Image repository holds the images and the corresponding metadata. Director repository instructs vehicles as to what updates should be installed next.	Separation of trust. Threshold signatures. Explicit and implicit revocation of keys. Keeping the most vulnerable keys offline. Flexible Implementation: Full or partial verification Asymmetric or symmetric ECU key, Encrypted or unencrypted update image.	OEM has to setup the Image and Director repositories, and Time server (if used)	Integrity: yes Authentication : yes Confidentiality: yes
Software Package Update Strategies[28][38][41][43]	Full binary: The new software is sent in its entirety. Diff file: the new software is compared to the previous version and creates a “diff” file which contains a list of differences between them. Only the diff file will be transmitted to the vehicle. Compression: Code compressed based on repeatedly replacing patterns with single tokens using available difference technologies. “A/B” approach: required double the flash amount on each ECU so that it can contain the current software in “primary” flash and has space for the full new version in the “secondary” flash. “In place” approach: only one version of the software exists on the device and individual blocks are erased and programmed as part of the update.	Full binary: No reliance on the previous software so that the update can take place even if the previous version has become corrupted. Diff binary: Quicker to transmit as smaller file sizes than original software. A/B approach: Best fit for the users as the ECU remains in normal operation using the primary storage	Full binary: Requires time to transmit the binary and space to store it at the receiving ECU. Traditional ECUs are typically communicating on CAN busses at 500kbit/s Diff binary: Reliant on the previous firmware being a specific version. Increases complexity in the ECU. A/B approach: Increased cost due to duplicating the execution flash on the MCU. “In place” approach : The vehicle is inoperable during the update. No “roll-back” to previous image	Based on Hash-based, Symmetric and asymmetric algorithms.

TABLE I

SOFTWARE OTA UPDATE APPROACHES USED IN AUTOMOTIVE

References

[1] Arinc 827-1: Electronic distribution of software by crate (eds crate). Report. Aeronautical Radio Inc.

[2] Arinc 835-1: Guidance for security of loadable software parts using digital signatures. Report. Aeronautical Radio Inc.

[3] Aurix tc4x sw application architecture supports next level of automotive applications. "https://www.infineon.com/cms/en/product/promopages/AURIX-TC4xx-SW-Application-Architecture/". Accessed: 2022-01-22.

[4] Concept: Classic platform flexibility. AUTOSAR Release R20-11.

[5] Esync alliance program: A multi-company initiative for ota updates and diagnostics. "https://esyncalliance.org/". Accessed: 2022-07-01.

[6] Global automotive hypervisor market. "https://www.globenewswire.com/news-release/2021/08/05/2275805/0/en/Automotive-Hypervisor-Market-to-Garner-2-03-Billion-by-2030-Allied-Market-Research.html". Accessed: 2022-01-22.

[7] How gm's ultifi software will change the buying and ownership experience. "https://www.autoblog.com/2021/09/30/ultifi-general-motors-software/". Accessed: 2021-10-12.

[8] Remote exploitation of an unaltered passenger vehicle. Black Hat USA (2015).

[9] Rfc 9019: A firmware update architecture for internet of things. RFC - Informational. April 2021.

[10] Securing safety critical automotive systems. "https://deepblue.lib.umich.edu/handle/2027.42/152321". Accessed: 2022-01-22.

[11] Software updates. "https://www.tesla.com/support/software-updates". Accessed: 2020-01-22.

[12] Specification of key manager. AUTOSAR Release 4.4.2.

[13] The update framework: A framework for securing software update systems. "https://theupdateframework.io/". Accessed: 2022-07-01.

[14] Upstream security global automotive cybersecurity report 2021. Upstream Security Ltd.

[15] Uptane framework: Securing software updates for automobiles. "https://uptane.github.io/". Accessed: 2020-11-22.

[16] Tcg guidance for secure update of software and firmware on embedded systems. *TCG*, 2020.

[17] M. Baza, M. Nabil, N. Lasla, K. Fidan, M. Mahmoud, and M. Abdallah. Blockchain-based firmware update scheme tailored for autonomous vehicles. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–7, 2019.

[18] R. Bielawski, R. Gaynier, D. Ma, S. Lauzon, and A. Weimerskirch. Cybersecurity of firmware updates. Technical report, 2020.

[19] Daniel Bogdan, Razvan Bogdan, and Mircea Popa. Delta flashing of an ecu in the automotive industry. In *11th IEEE International Symposium on Applied Computational Intelligence and Informatics*. IEEE, 2016.

[20] A. Boudguiga, N. Bouzerna, L. Granboulan, A. Olivereau, F. Quesnel, A. Roger, and R. Sirdey. Towards better availability and accountability for iot updates by means of a blockchain. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 50–58, 2017.

[21] Gerth Stølting Brodal and Gabriel Moruz. Skewed binary search trees. In Yossi Azar and Thomas Erlebach, editors, *Algorithms – ESA 2006*, pages 708–719, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[22] O. Burkacky, J. Deichmann, B. Klein, K. Pototzky, and G. Scherf. Cybersecurity in automotive: Mastering the challenge. *McKinsey & Company, Inc.*, 2020.

[23] S. Choi and J. Lee. Blockchain-based distributed firmware update architecture for iot devices. *IEEE Access*, 8:37518–37525, 2020.

[24] A. Ghosal, S. Halder, and M. Conti. Stride: Scalable and secure over-the-air software update scheme for autonomous vehicles. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2020.

[25] I. Hossain, S. M. Mahmud, and S. Shanker. Analysis of a secure software upload technique in advanced vehicles using wireless links. In *IEEE Intelligent Transportation Systems Conference*, page 1010–1015. IEEE, 2007.

[26] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, 2018.

[27] Shimin Jing, Xin Zheng, and Zhengwen Chen. Review and investigation of merkle tree's technical principles and related application fields. In *2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA)*, pages 86–90, 2021.

[28] S. Kang, I. Chun, and W. Kim. Dynamic software updating for cyber-physical systems. In *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, pages 1–3, 2014.

[29] R. Kiyohara, S. Mii, M. Matsumoto, M. Numao, and S. Kurihara. A new method of fast compression of program code for ota updates in consumer devices. *IEEE Transactions on Consumer Electronics*, 55(2):812–817, 2009.

[30] Leslie Lamport. Constructing digital signatures from a one way function. 2016.

[31] B. Lee and J.-H Lee. Blockchain-based secure firmware update for embedded devices in an internet of things environment. pages 1152–1167. J. Supercomput, 2016.

[32] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 468–477, 2017.

[33] K. Mansour, W. Farag, and M. ElHelw. Airodiag: a sophisticated tool that diagnoses and updates vehicles software over air. In *IEEE International Electric Vehicle Conference*, pages 1–7. IEEE, 2012.

[34] K. Mayilsamy, N. Ramachandran, and V. S. Raj. An integrated approach for data security in vehicle diagnostics over internet protocol and software update over the air. *Computers & Electrical Engineering*, page 578–593, 2018.

[35] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987.

[36] D. K. Nilsson and U. E. Larson. Secure firmware updates over the air in intelligent vehicles. In *IEEE International Conference on Communications Workshops*, page 380–384. IEEE, 2008.

[37] Joydeep Rakshit and Kartik Mohanram. Assure: Authentication scheme for secure energy efficient non-volatile memories. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017.

[38] K. Ryozo, M. Satoshi, M. Mitsuhiro, N. Masayuki, and K. Satoshi. New method of fast compression of program code for ota updates in consumer devices. In *IEEE Transactions on Consumer Electronics*.

[39] Christian Schleiffer, Marko Wolf, André Weimerskirch, and Lars Wolleschensky. Secure key management - a key feature for modern vehicle electronics. In *SAE 2013 World Congress Exhibition*. SAE International, apr 2013.

[40] M. Steger, A. Dorri, S. S. Kanhere, K. Romer, R. Jurdak, and M. Karner. Secure wireless automotive software updates using blockchains: A proof of concept. In *22nd International Forum on Advanced Microsystems for Automotive Applications*, page 137–149, 2018.

[41] Bjoern Steurich, Klaus Scheibert, Axel Freiwald, and Martin Klimke. Feasibility study for a secure and seamless integration of over the air software update capability in an advanced board net architecture. In *SAE Technical Paper*. SAE International, 2016.

[42] S. Stević, V. Lazić, M. Z. Bjelica, and N. Lukić. Iot-based software update proposal for next generation automotive middleware stacks. In *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, 2018.

[43] N. Suzuki, T. Hayashi, and R. Kiyohara. Data compression for software updating of ecus. In *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, pages 304–307, 2019.

[44] A. Yohan and N. Lo. An over-the-blockchain firmware update framework for iot devices. In *2018 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8, 2018.

[45] Alexander Zeeb. Autosar classic platform flexibility managing the complexity of distributed embedded software development : Invited talk. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pages 167–167, 2021.

[46] Yu Zou and Mingjie Lin. Fast: A frequency-aware skewed merkle tree for fpga-secured embedded systems. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 326–331, 2019.