# Secure Software Updates for Intelligent Connected Vehicles

Yunshui Zhou, Xinkai Wu, Pengcheng Wang*
School of Transportation Science and Engineering, Beihang University, Beijing, China,100191
Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing, 100191
pcwang@buaa.edu.cn
*Corresponding author

***Abstract*—With the emergence and application of intelligent and connected technologies, the Intelligent Connected vehicle (ICV) becomes the most promising manner to improve transportation. Meanwhile, the demand to update the software is also increasing to repair the bug in the software or introduce new functions for the automobile. This paper develops a new distributed remote software update system for automobiles which includes the OEM (Original Equipment Manufacturer), supplier and supervisory authority. In this system architecture, the OEM is able to delegate supplier to update software, so the OEM and supplier can coordinate with each other to perform the security software update. In this process, the government as a regulator needs to supervise the software update information. In the end, a prototype system is developed to verify the feasibility of our update system.**

***Keywords—connected vehicle; remote software updates; security; distributed***

## I. INTRODUCTION

The number of electronic control units (ECUs) on each vehicle has reached hundreds, and the number of software codes exceeds 100 million lines, which has dramatically increased the complexity of the software system of the entire vehicle. After the car leaves the factory, you need to upgrade the original software and firmware for introducing new features, fixing software bugs or security vulnerabilities. The traditional way of updating software requires recalling the car to the 4S store and updating the software via a wired interface such as OBD or USB. This method cannot timely update software for large-scale vehicles, and the cost is very high.

The Over-the-Air (OTA) update [4] has become a trend in automotive software updates. The OTA upgrade also means exposing the car's internal software to the public network, thereby increasing the attack surface of the car, and the attacker can launch the attack remotely to the car [7]. The existing secure OTA system for automobile leverage cryptographic methods [1] [2][3] and blockchain-based methods [8] [9] to ensure the integrity, authenticity and confidentiality of software update package. But these systems, which do not consider the supply chain model of the automotive industry or involve the supplier of vehicle components. In addition, the regulator of the automotive industry cannot participate in the procedure of software update. Regulators need to consider the software security and compliance of automotive software updates for safety reasons. For example, they want to scan the software to detect if there are backdoors or known vulnerabilities.

In this study, we proposed a novel automotive security software update system, *i.e.*, distributed automotive software update, which requires the OEM (Original Equipment Manufacturer), the software provider, and the regulator to cooperate with each other. This system can increase the security of software updates for the automobile. A proof-of-concept system is implemented to evaluate the applicability of the proposed software update architecture.

## II. BACKGROUND

### A. Secure Automotive Software Updates

With the increase of the intelligent level of vehicle parts, more and more software and hardware are embedded inside the vehicle. There is no doubt that the increase of vehicle complexity will bring difficulties in maintenance. Additionally, the need for software and firmware upgrades is growing. To this end, the OTA update, a simple and convenient method is encouraged and investigated by many scholars and engineers.

Although the OTA update makes the upgrade of software more convenient, it also introduces some extra attack surfaces [6]. The attacker may even install malicious software on vehicles by exploiting the OTA process. The most important security requirements of software updates are authenticity and integrity [10]. The authenticity means that all the software/firmware to be updated are released by the trusted organization but not the malicious organization. The integrity requires the update packages have not been tampered in the transmission process. Hash algorithm and digital signature are the common methods [3] to ensure authenticity and integrity. In the following text, we will introduce these two methods.

### B. Hash Algorithm

Because of its strong anti-collision property, the hash function[11] has become the most widely used technology in verifying data integrity. The working principle of the hash function is to accept variable-length messages, output fixed-length hash values, and it is irreversible. The common hash algorithms are Message Digest algorithm and Secure Hash Algorithm, we use the SHA256 because of the collision-resistant characteristic.

### C. Cryptographic Digital Signature

Digital signature [5] plays an important role in the realization of identity authentication, data integrity, non-repudiation, and other functions. We use the digital signature to protect the authenticity of the software update package.

Digital signature provides an authentication method, which makes use of data encryption technology and data transformation technology to enable the receiver to identify the identity claimed by the sender. The sender can not deny the fact that the data has been sent in the future. Digital signature is based on public key cryptography. In public key cryptosystem, the signer has his own private key and the corresponding public key is public. Then the signer encrypts the data with his own private key. Others can use the public key of the signer to decrypt the data. Firstly, the sender transforms the information mathematically, and the information obtained corresponds uniquely to the original information. Then the sender inversely transforms the original information to get the original information. As long as the mathematical transformation method is designed reasonably, the transformed information has strong security in transmission, and it is difficult to be decoded and tampered.

We use RSA-based signature scheme in the software system. RSA public key cryptosystem was proposed in 1978 by Rivest, Shamir and Adelman, three young scientists of MIT, and named after three inventors. It is the first mature and theoretically most successful asymmetric key cryptosystem. The security of RSA is based on the difficulty of factorization of large integers. The problem of large integer factorization is a well-known problem in mathematics. So far, there is no effective way to solve it, so the security of RSA algorithm can be ensured.

## III. DISTRIBUTED SOFTWARE UPDATE SYSTEM

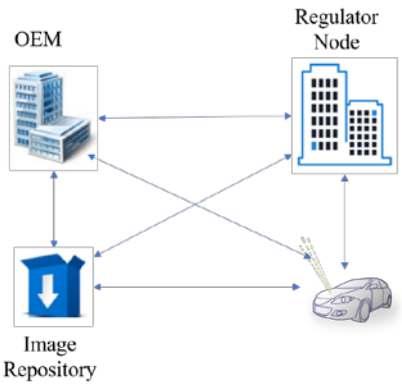


Fig. 1. The architecture of secure automotive software update

In this section, we propose our secure automotive software update systems. First, we present the architecture of our software update system, and illustrate the different roles in this system and their duty for software. Second, we explain how software updates are sent to vehicles. Third, we describe the procedure of downloading and verifying software updates on a vehicle. Finally, we analyze the security of this software updates system.

### *A. Automotive Software Update System Architecture*

To make up the shortcomings of traditional automotive software update system, we proposed the distributed automotive software update system which involves OEM, Software Provider, and Regulator. The OEM can customize the software for different vehicles, but it does not need to manage and maintain all software packages. The OEM plays the role of director, it is able to delegate software provider to supply software products. Also, it can monitor the software update behavior conducted by the delegated third parts. In addition, the OEM can also allow the regulator to detect the software update package, which introduces additional security needs for software updates. The architecture is showed in *Fig. 1*.

**Master Repository:** The OEM sets up the master repository, and it offers the metadata of software updates for different vehicles. The metadata contains the version of the software, the hash of the software, the URL of the image repository, and the public key of the image repository which managed and maintained by the software provider. Besides, the metadata is signed using the private key by the OEM, which ensures the authenticity of the metadata. The OEM can also empower regulator nodes by maintaining the list of public key binding to authorized regulator nodes. When vehicles need to update the software, they obtain the metadata from the master repository, then downloading the software image from image repository according to the image repository's address in the metadata.

**Image Repository:** The image repository managed by software provider accommodate the software image and its metadata, which includes the version, hash of image, filename, and the public key of the image repository. In addition, the image repository would sign the metadata using its private key.

**Regulator Node:** The regulator node managed by government supervisory authority is empowered by master repository. Supervisory authorities can force enterprises to authorize the establishment of regulator nodes through laws and regulations. It pulls the metadata and software image from master repository and image repository, then reviews the software image to detect if there are backdoor or vulnerabilities. Only after the software is attested can the software image be downloaded by vehicles. In the regulator node, a table is maintained and records the attestation of the software image. So vehicles will know if the software image they want to download is allowed to download by regulator.

### *B. Releasing Software Updates*

In this subsection, we present the workflow of releasing software updates. We will explain the whole corresponding procedure of releasing a software update to show how our update system to securely distribute a new software image to vehicles, and illustrate how the aforementioned three roles cooperate with each other to ensure the security of software updates.

First, the OEM as the director of this system will initialize the system. It will store the public key of the software and the regulator node, which can be implemented by Public Key Instructure (PKI). Also, it will notify other roles its public key. Before the vehicle leaving the OEM's factory, the OEM will install its software update client in vehicles, which can visit the master repository and store the public key of master repository so vehicles are able to gain the software updates and verify the signature of master repository. The master repository directs the whole software update process. It will customize the software for different vehicles. In the master repository, the OEM will offer the metadata of software image distributed to according vehicles. However, the software image and the metadata of the image

will be issued by the image repository, *i.e.* software provider. When the provider needs to push a new software image, it will generate the metadata of the image. We define the metadata of an updated image as the tuple $M_i = (U_{id}, U_v, U_{hash}, P_{pk}, P_{sign})$ where $U_{id}$ is the identifier of update software image, $U_v$ is the version of the image, $U_{hash}$ is the hash of image, $P_{pk}$ is public key of software provider, and $P_{sign}$ is the signature of all previous fields signed by the software provider. If the confidentiality of the image is a concern for protecting intellectual property, the image will be encrypted before distributed using the method of broadcast encryption. After the metadata $M_i$ is generated, the $M_i$ will be sent to the OEM, so the OEM will know a new version of $U_{id}$ has been supplied.

When an update notification is sent from image repository, the OEM will verify the authenticity of the metadata. If the master repository confirms the authenticity of this metadata, the master repository will sign this metadata $M_i$ using its own private key and generate the metadata $M_m = (M_i, O_{pk}, O_{sign})$ where $O_{pk}$ is the public key of the OEM and $O_{sign}$ is the signature of the OEM. Then this metadata $M_m$ is sent to the regulator node. In addition, the master repository will generate another metadata $M_v$ pushed to the target vehicles, $M_v = (M_i, R_{pk}, O_{pk}, O_{sign})$ where $R_{pk}$ is the public of regulator node, so the target vehicles are able to verify if the software image is attested by regulator node. For different target vehicles, the master repository will maintain a list of the software image, *i.e.* a list of $M_v$. The target vehicle can obtain the newest version metadata from its corresponding software list.

After the regulator node receives the metadata $M_m$ from the master repository, it will verify the signature. Then it will download the software image from the image repository and its metadata $M_i$, and verify the integrity and authenticity of the image. Finally, it will scan and detect the software image. Only after passing the review, the regulator will generate the $M_r = (M_i, R_{pk}, R_{sign})$ where the $R_{sign}$ is the signature of previous filed. This indicates the software update is attested by the regulator node. Therefore, vehicles can verify the metadata $M_r$ to know if the software update is attested by the regulator node.

### C. Downloading and Verifying Software Updates

On a vehicle, the update gateway obtains updates from repositories and the other nodes obtaining software updates from the update gateway. The whole update process has two phases. In the first phase, the gateway period queries the newest version of the software image from the list of software image metadata in master repository. If the version of the existing software is not the newest version, it will conduct the software update and download the latest software image and metadata. First, if download the metadata $M_v$ from the master repository and verify the authenticity of the metadata by the signature. If the signature is signed by the OEM, the vehicle will obtain the metadata $M_r$ from the regulator node. If the corresponding $M_r$ is signed by the regulator node, the vehicle will verify the signature which indicates the software image is attested by regulator node. Then the vehicle will continue to download the metadata $M_i$ and image from the image repository, then verify the signature is signed by the $P_{pk}$ in $M_i$. Finally, the vehicle will apply hash algorithm on the image, and verify whether the output equals the $U_{hash}$ in $M_i$. If the verification fails, it shows the integrity of the image is attacked, the image will be discarded. Otherwise, the whole verify process carried out by gateway is successful, the metadata $M_i$ and image will be stored and then distributed to the corresponding node in the vehicle.

In the second phase, the updated gateway sends the software image and metadata to the other node, *i.e.*, ECUs in the vehicle. Because of the computing resources constrain of ECU, the ECU only verifies the metadata $M_i$ and the image. Like the gateway, the ECU will verify the signature of $M_i$ and computing the hash of the image, then compare with the $U_{hash}$ in $M_i$.

## IV. Prototype Implementation

In this section, we describe the prototype implementation of the secure software update system proposed in the previous section. The implemented prototype system consists of two main building parts. First, the repository node includes the three roles, *i.e.*, the master repository, the image repository and the regulator node to offer the software image and metadata for target vehicles. The second is the local building block including the updated gateway and ECUs.

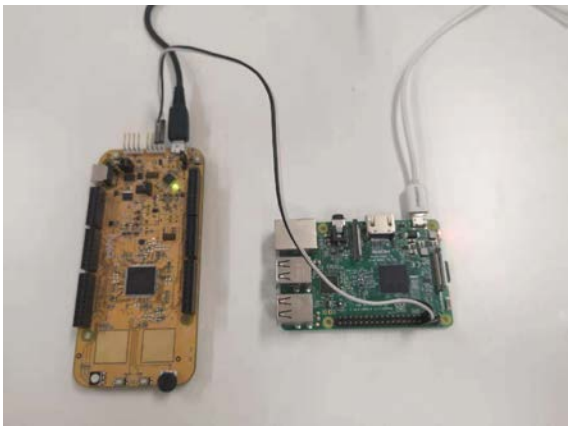

Fig. 2. The prototype implementation

The repository is deployed in the same physical machine but in the different virtual machine. We use the Virtualbox to host the three repositories, which installs Ubuntu operating system and developed used Python3 due to its capability of rapid prototyping development. They use the HTTP protocol to communicate with each other.

As showed in *Fig. 2*, for simulating the process of software update on the vehicle, we use RaspberryPi3 and S32K MCU as the gateway and ECU respectively. Because of the RaspberryPi3 runs a Linux system and has a similar computing resource with the gateway in modern vehicles, and the S32K MCU, developed by NXP, is a common controller used in automobiles. The RaspberryPi3 can communicate with repositories by HTTP, and communicate with ECU by CAN-bus. However, the RaspberryPi3 have no CAN-bus module, so we use the MCP2515 module which is an adapter transforming SPI to CAN. As for the software stack, we utilize the socket can which is provided by Linux. On the gateway, we develop the update manager using Python3, which gets the software from repositories and sends to ECU. On the ECU, we develop the bootloader which can

get software image from the gateway, verify the image and install the image.

## V. CONCLUSION

In this paper, we present a distributed secure software update system for automobile able to involve the OEM, the software provider and regulator of automotive industry. Our update architecture provides the security and trustworthy interconnection between all the involved roles while ensuring the authenticity and integrity of software updates. We also present a preference implementation to show the feasibility of this architecture. But we do not evaluate the performance of our system. We plan to compare our performance with the traditional automotive update system and improve our implementation.

## ACKNOWLEDGMENT

This work was partially supported by the National Key Research and Development Program of China (2016YFB0100902) and the National Science Foundation of China under Grant (61773040).

## REFERENCES

[1] .Hossain, I, Mahmud, S, Analysis of a secure software upload technique in advanced vehicles using wireless links, Intelligent Transportation Systems Conference, 1010–1015, 2007.

[2] Idrees, M, Schweppe, H, et al., Secure automotive on-board protocols: A case of over-the-air firmware updates, Lecture Notes in Computer Science, 6596 LNCS (2011), 224–238, 2011.

[3] Nilsson, D, Larson, U, Secure firmware updates over the air in intelligent vehicles, IEEE Conference on Communications, 380–384, 2008.

[4] Steger, M, Karner, M, et al., SecUp: Secure and Efficient Wireless Soft-ware Updates for Vehicles, IEEE Conference on Digital System Design (DSD), 628–636, 2016.

[5] Mallissery, S, Pai, M, et al., Improving the PKI to build trust architecture for VANET by using short-time certificate mngt. and Merkle Signature Scheme, Asia-Pacific Conference on Computer Aided System Engineering, 146-151, 2014.

[6] Foster, D, Prudhomme, A, et al., Fast and Vulnerable: A Story of Telematic Failures, USENIX Workshop on Offensive Technologies, 2015.

[7] Valasek, C, Miller, C, Remote Exploitation of an Unaltered Passenger Vehicle, White Paper, p. 93, 2015.

[8] Baza, Mohamed, et al. "Blockchain-based Firmware Update Scheme Tailored for Autonomous Vehicles." arXiv preprint arXiv:1811.05905 (2018).

[9] Lee, Boohyung, et al. "Firmware verification of embedded devices based on a blockchain." International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness. Springer, Cham,2016.

[10] Cappos, Justin, et al. "A look in the mirror: Attacks on package managers." *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008.

[11] Eastlake 3rd, D., and Paul Jones.US secure hash algorithm 1 (SHA1). No.RFC3174.2001.