ARTICLE INFO
Article ID: 11-04-01-0002

doi:10.4271/11-04-01-0002

Towards a Blockchain Framework for Autonomous Vehicle System Integrity

Kenneth H. Chan,[1] Matthew Pasco,[1] and Betty H.C. Cheng[1]

[1]Michigan State University, USA

Abstract

Traditionally, Electronic Control Units (ECUs) in vehicles have been left unsecured. Ensuring cybersecurity in an ECU network is challenging as there is no centralized authority in the vehicle to provide security as a service. While progress has been made to address cybersecurity vulnerabilities, many of these approaches have focused on enterprise, software-centric systems and require more computational resources than typically available for onboard vehicular devices. Furthermore, vehicle networks have the additional challenge of mitigating security vulnerabilities while satisfying safety and performance constraints. This article introduces a blockchain framework to detect unauthorized modifications to vehicle ECUs. A proof of concept blockchain prototype framework is implemented on a set of microprocessors (comparable to those used by simple ECUs) as a means to assess the efficacy of using our blockchain approach to detect unauthorized updates.

History

Received: 11 Aug 2020
Revised: 04 Feb 2021
Accepted: 13 Apr 2021
e-Available: 05 May 2021

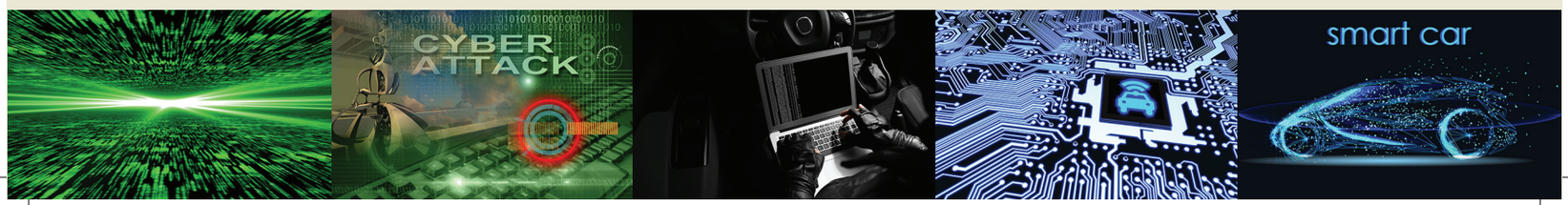
Keywords

Automotive cybersecurity, Firmware integrity, Blockchain, Electronic control units, Over-the-air (OTA) updates

Citation

Chan, K., Pasco, M., and Cheng, B., "Towards a Blockchain Framework for Autonomous Vehicle System Integrity," *SAE Int. J. Transp. Cyber. & Privacy* 4(1):19-38, 2021, doi:10.4271/11-04-01-0002.

ISSN: 2572-1046
e-ISSN: 2572-1054



Introduction

Electronic Control Units (ECUs), the real-time controllers of automotive systems, have traditionally been left unsecured against cyber threats [1, 2]. Exploited ECUs potentially pose a threat to vehicular safety by allowing adversaries to obtain partial and even full control over the vehicle [3, 4, 5, 6]. ECUs may be compromised through several attack vectors, including physical tampering, outward-facing communication attacks, or Over-the-Air (OTA) updates [2, 3]. This article proposes a distributed solution to secure automotive ECUs without requiring additional hardware or imposing adverse performance impact on functional operations.

Traditional cybersecurity protection approaches managed by a centralized authority [7] (e.g., intrusion detection systems [8]) are not directly applicable to handle automotive cybersecurity given the disparate and distributed nature of the automotive attack vectors [2, 3, 6]. Currently, multiple international, cross-organizational groups are developing guidelines and standards for automotive cybersecurity [9], but to date, there is no *formally adopted* or even *de facto* standard for securing OTA updates or ECU integrity [9, 10]. Original Equipment Manufacturers (OEMs) often implement their own proprietary solutions [10] and/or leverage third-party protocols such as Uptane [11] or security modules within AUTOSAR [12]. When considering the security of onboard ECUs, a number of research efforts have explored the use of externally managed blockchains [13, 14] and encryption-based approaches with blockchain to secure the vehicle that requires additional hardware and/or impose performance delays [15, 16, 17, 18]. From a cost- and performance-sensitive perspective, a challenge is how to effectively use available onboard resources to address ECU integrity with respect to outward-facing attack vectors (e.g., OTA updates, malicious signals).

This article introduces an in-vehicle software-based blockchain framework to secure onboard ECUs against adversarial modifications. Leveraging the organizational similarity between the distributed computing nature of the ECU network and a peer-to-peer (P2P) system, we consider ECUs in a vehicle as nodes in a distributed system and apply techniques analogous to those used to secure P2P systems [19]. Specifically, we propose a software virtualization blockchain approach to secure onboard vehicular ECUs. A majority of vehicles communicate via the Controller Area Network (CAN), a bus protocol, to provide broadcast data communication between ECUs [20]. Using the CAN bus, ECUs can broadcast their data to all other ECUs to verify or add to the blockchain. In order to minimize adverse effects on functional performance, our approach leverages ECU multicore computing resources to provide the needed computational support for the blockchain implementation.

The proposed blockchain approach secures ECUs by providing a robust record of firmware data with which ECUs can verify. Using a blockchain, firmware data are resilient against adversarial modification. The blockchain approach removes a single point of failure and is secure as long as the majority of ECUs in the system are not malicious. Our study shows that by employing a blockchain to store valid firmware data of ECUs, we are able to create immutable records that can be used to detect unauthorized changes.

As a proof of concept for our approach, we constructed a lightweight blockchain framework and deployed it across a network of microcontrollers, each mimicking an ECU. We demonstrate the effectiveness of the proposed approach by detecting unauthorized updates to our ECU microcontrollers. Furthermore, we show that a software implementation of the blockchain operates within an acceptable time frame. The remainder of this article is organized as follows. First, we overview background material and overview related work. Next, we describe the proposed blockchain framework and its adaptability to ECUs with different computational resources. As proof of concept, we describe a prototype implementation of our blockchain technology on a small collection of microprocessors and illustrate the ability to detect unauthorized changes to the firmware, including functional and blockchain cores. Finally, we conclude the article and discuss future directions.

Background

This section provides background information for the article. We overview current security vulnerabilities, protection strategies relevant to securing onboard ECUs, the blockchain technology, and briefly describe how blockchains have been used in Internet-of-Things (IoT) contexts.

Security of ECU Updates

Many ECUs on the vehicle are safety critical where errors can cause injury or loss of life (e.g., anti-lock braking system, airbag control, collision avoidance system, etc. [21]). If an adversary compromises the update via a man-in-the-middle attack, then they can potentially upload malicious firmware to ECUs. As a result, securing ECU firmware integrity and OTA updates are important for the safe operation of the vehicle.

Currently, most OEMs use proprietary solutions to deliver OTA update data to vehicles [10]. Traditional OTA implementations often use techniques such as symmetric key encryption [22, 23] or asymmetric key encryption [24] to secure update data delivered over Bluetooth or Wi-Fi [25]. Unfortunately, encryption-based approaches are still vulnerable to attacks. For example, once a key has been compromised, there is no explicit revocation of the compromised key since it is installed directly on the vehicle. Revoking a compromised key would involve recalling all vehicles to a dealership to reinstall a new key. This process is time consuming and costly, as well as difficult to implement, as it depends on car owners to voluntarily bring their vehicles to dealerships for updates. Recently, Uptane [11], an update protocol, has been introduced to promote secure delivery of OTA updates. Uptane divides the OEM repository into a supplier repository and a director repository, which contains offline keys and online keys, respectively. The supplier repository provides signed metadata for each update available for ECUs while the director repository signs the metadata. By introducing a separation of duty between the repositories, Uptane provides the ability for OEMs to revoke compromised online keys.

However, ECUs can still be compromised via other vulnerabilities such as ECU software flashing or wireless device attacks [2, 3, 5, 26]. Current OTA update implementations deliver OTA update data to a designated ECU head unit where each ECU independently verifies its own firmware integrity. But current implementations have several single point-of-failure challenges. Thus new research is required to provide further protection of ECU integrity.

Security of ECU and CAN Bus

Similar to OTA implementations, many OEMs keep their ECU and CAN bus security solutions proprietary [10]. One common approach to secure ECUs is through the secure boot of the firmware when the vehicle is started [27, 28]. While secure booting ensures that only authenticated firmware drivers are allowed to operate, each ECU only verifies its own integrity, which introduces a single point-of-failure problem. If an attack is able to override the integrity check, then the vehicle can still be compromised as a result [28]. Furthermore, as the ECUs are only verified when the system first turns on, attacks that occur during the operation of the vehicle are not detected. If an adversary compromises an ECU's integrity, then they are able to control the ECU and use it to send malicious messages directly over the CAN bus.

Methods to secure ECU messages over the CAN bus have been proposed [29, 30, 31]. For example, Ueda et al. [29] use a security monitoring system, mutual authentication nodes, and a keyed-hash Message Authentication Code algorithm to ensure messages passed on the CAN bus are not modified. However, if an adversary compromises an ECU's software integrity, then the malicious ECU can send malicious messages over the CAN bus in place of the original ECU, thus bypassing message authentication measures. Furthermore, as there is no record of messages being stored, OEMs cannot perform a postmortem analysis of the attack.

Blockchain Overview

The blockchain technology introduces a technique in which distributed systems can manage a database without centralized authority [19, 32]. Unlike traditional databases, a blockchain's distributed database, known as the *ledger*, is controlled by the majority of nodes in the system. As the data is distributed across all nodes participating in the blockchain, a blockchain removes a single point of failure and ensures that data is not easily modified. For brevity, herein we refer to each unit in the blockchain as a *node*, and the network of nodes managing the blockchain as the *system*.

A blockchain comprises blocks of data that are structured logically similar to that of a linked list data structure, where each element of data points to the previous item in the list. Each block in a blockchain contains the following elements:

1. **Hash of the previous block.** Each block contains the hash value of the previous block.
2. **Transactions.** A transaction contains information or an asset that is stored on the blockchain.
3. **Proof of work.** A Proof of Work (PoW) is a cryptographic puzzle that is computationally difficult to solve; it is used to deter data modification or malicious block creation.

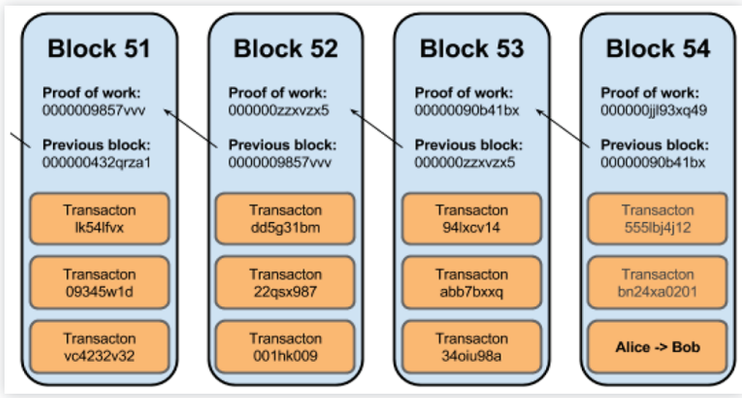
For a distributed system, a blockchain removes the need for nodes to trust information from other unknown nodes. Participants in a distributed network often have no information on other nodes in the system where operating on information from another node is risky as other nodes may be malicious. Due to the PoW, computational resources are needed to create each block on the blockchain. The PoW produces a certificate that can be easily verified by other nodes in the system. As a result, if every node in the system trusts the data on the longest blockchain in the system (i.e., the blockchain with the most amount of computational work), then the blockchain is able to provide data integrity and a consensus in a distributed system without relying on information from a single node. Figure 1 depicts the structure of a typical blockchain.

Blockchain Elements

The following describes blockchain elements in further detail.

1. Hash of the previous block. Each block of the blockchain contains the hash of the previous block, thereby establishing a cryptographic connection between the two blocks. If the information of a block is modified on the blockchain, then the hash of the

FIGURE 1 Example blockchain structure.



Reprinted with permission from Y. Brikman, "Bitcoin by analogy." https://www.ybrikman.com/writing/2014/04/24/bitcoin-by-analogy/.

entire modified block changes. Each subsequent block in the blockchain thus becomes invalid as it no longer contains the correct hash of the previous block. In order for nodes in the system to accept the modified blockchain, a new PoW must be recalculated for the modified block and all subsequent blocks, since their respective hashes are no longer valid. Because the PoW is computationally expensive to calculate, blockchains are resistant to modifications.

2. Transactions. A transaction is a record that is stored on the blockchain. For example, a transaction may include information on an exchange of currency between two users, medical records, or a piece of data to be stored on the blockchain. Each block in a blockchain contains a predefined amount of transactions by the blockchain implementation.

Transactions can be added to the blockchain by any node in the system. When a node wants to insert a transaction into the blockchain, the transaction is first submitted to the system. The system votes, coming to a consensus if the transaction is valid and therefore should be added to the blockchain. If the system agrees, then the transaction is often grouped with other transactions to form a block. Once enough transactions have accumulated, a PoW is calculated and the new block is added to the blockchain.

3. PoW. In a blockchain, a PoW is a piece of data that is difficult to produce, yet easy to verify. Essentially, each block must provide evidence that computational resources were spent to create it in order to be recognized as valid by nodes in the system. In traditional blockchain implementations such as Bitcoin [33], the PoW is to find a *nonce* value that together with the block creates a hash with a certain number of leading zeroes. A PoW has the following properties:

1. *A PoW is (computationally) hard to produce.* Generating a valid PoW requires a node to correctly "guess" the nonce through trial and error. The PoW is created by repeatedly generating a hash with the block and a different nonce until one with a satisfying number of leading zeroes is found.
2. *A PoW is (computationally) easy to verify.* Verifying a PoW requires little computational resources and time since a node can simply generate the hash of the block with the PoW and ensure that it has the correct number of leading zeroes.

The PoW exists to ensure that blocks cannot be quickly generated. With the PoW, an adversary cannot produce a large amount of computationally inexpensive and malicious blocks to be added to the blockchain. Additionally, the PoW requires adversaries to computationally outperform all other nodes in the system to modify or introduce a malicious block to the blockchain. Through the complexity introduced by the PoW, the blockchain is able to guarantee immutability in a decentralized network.

In a blockchain, the number of leading zeroes is known as the *hashing difficulty* [34]. The hashing difficulty varies in each blockchain implementation. By increasing the number of leading zeroes required for each block's hash, the size of the stored hash can be reduced. The difficulty is also used to slow the rate of hashing, which introduces fairness into the blockchain network as it penalizes nodes with faster network connections. The hashing difficulty can be incremented in sequential or random order to change the output of a hashed message. In Bitcoin [33], the current difficulty requires seventeen leading zeros as shown in Table 1. However, as the required number of zeros increases, then the average time to compute a hash that satisfies the targeted number of leading zeroes also increases. As a result, the hashing difficulty can reduce the size of the messages stored on the blockchain in exchange for an increase in the need for computational resources and time.

Resilience to Unauthorized Modifications

The blockchain technology introduces a technique for decentralized networks to store immutable data. In this section, we consider two scenarios in which an adversary can compromise the blockchain and discuss their pragmatics.

Modifying a Transaction First, an adversary may compromise the blockchain by changing a transaction in an existing block (i.e., data modification). The adversary first locates the target transaction and modifies it. A new hash of the block will be produced with the modified block, requiring the adversary to recalculate a correct PoW that satisfies the specified number of leading zeroes. When the hash of the modified block changes due to adversarial modification, the data in the next block also becomes invalid as it contains the hash of the premodified block. Thus the adversary must recalculate each subsequent blocks' PoW to maintain its validity. Since the blockchain is constantly growing in size, the adversary must compromise more than 50% of the computational resources in the network to modify blocks faster than the rate of incoming blocks to eventually modify the entire blockchain to accept the modified transaction [35]. Note that as the number of leading zeroes increases, the modification difficulty also increases as the adversary must find a rarer nonce that satisfies the number of leading zeroes for each block. An adversary must overcome several hurdles before the unauthorized modification is able to successfully compromise the blockchain.

Adding a Malicious Block Another method in which an adversary may compromise the blockchain is by adding their own malicious blocks to the blockchain (i.e., data fabrication). An adversary achieves this goal by either introducing a fabricated transaction or introducing a fabricated block to the system. When introducing a new transaction, each node in the system votes to include the transaction or not. An invalid transaction

TABLE 1 An example of a Bitcoin block.[1]

Block #411940	
Summary	
Number of transactions	1098
Output total	7,253.02558914 BTC
Transaction volume	7227.78980182 BTC
Fee reward	0.23578732 BTC
Height	411940 (main chain)
Timestamp	2016-05-16 00:22:09
Received time	2016-05-16 00:22:09
Relayed by	BTCC Pool
Difficulty	194,254,820,283.44
Bits	403024122
Size	814.78 kB
Weight	3,259,868 WU
Version	0x20000001
Nonce	506,565,497
Block reward	25 BTC
Hashes	
Hash	000000000000000057fcc708cf0130d 95e27c5819203e9f967ac56e4df598ee
Merkle root	f7b19ebf61f25feba68f7e3b139a846f2 b6d7a4eb442085007bccf76b523849c
Previous block	000000000000000337fc316c605be3 392b8b26aaf387c9756915feecb4780f
Next block	000000000000000019d7f57d7b9dd5e 4d36b16dd0355f1cc30933fabf4dcc12

Data taken from https: //www.blockchain.com/btc/block/411940. © SAE International

[1]Information from **blockchain.com**, **"Block #411940"**, https: //www.blockchain.com/btc/block/411940,2016, accessed April 27, 2019.

submitted by a malicious node will be rejected by other nodes. Alternatively, an adversary may attempt to include their malicious block into the system. Consider a fork in the blockchain as the adversary attempts to include their own block in the blockchain. The first fork represents the "correct" blockchain that is maintained by legitimate nodes in the system, while the second fork is maintained by malicious nodes in the system attempting to introduce the malicious block. Since the system uses the longest blockchain, the adversary must computationally outperform all other nodes in finding the correct PoWs. Similar to adding a malicious transaction, both attacks require that more than half the nodes in the system be compromised to execute, which is difficult in practice.

Blockchain for IOT

An increasingly popular approach to secure distributed networks is through the use of blockchain technology [19, 32]. Banerjee et al. [19] propose an abstract authentication layer that authenticates IoT devices on the blockchain. By adding a hardware device to each IoT device, they introduce an authentication layer to the original IoT network. The new hardware handles the blockchain protocol by implementing an abstract layer that provides security. The authentication layer periodically checks device firmware and correlates it against data stored on the blockchain. If the device firmware has been modified, then IoT devices in the network would be able to detect the modification. Furthermore, their approach uses a secure hardware module that is attached to the existing device. As a result, if an adversary compromises either the blockchain device or the original IoT device, then the other device is not immediately affected. For brevity, we refer to Banerjee et al.'s approach [19] as the *Authentication Layer Approach*.

Related Work

A number of researchers have investigated blockchain-related approaches to secure OTA updates and other automotive elements. For example, He et al. [36] explored the use of blockchain to secure OTA updates for IoT devices. However, their implementation employs a traditional heavyweight blockchain that low-level ECUs may not be able to use due to a lack of computational resources. Baza et al. [13] considered using blockchain to ensure OTA update is not modified during delivery, but their approach addresses the integrity of propagating OTA updates in a network of vehicles. Deshpande et al. [15] proposed an approach to provide a secure execution or storage environment for an ECU, which requires additional hardware. Finally, other researchers have proposed using blockchain to provide confidentiality to ECU communications [16, 17]. While encrypting messages for ECU communications seems rudimentary, additional performance overhead for encryption and time constraints introduced by the blockchain may be infeasible for time-sensitive and safety-critical ECUs. In contrast, we propose a solution that can be implemented using onboard resources on current vehicles without incurring detrimental performance degradation.

Other approaches to secure OTA updates have been proposed. Yohan et al. [37] proposed a blockchain framework for updating IoT devices, which verifies the firmware through an intermediary smart contract, existing separately from the firmware vendor and the IoT device. Kent et al. [18] proposed an asymmetric key-based approach to sign and validate ECU firmware, which could be used to ensure that OTA updates are properly and securely delivered to a target system. Recently, Falco and Siegel [14] proposed a similar approach to securing a vehicle. However, the aim of their approach is to allow OEMs to ensure the latest version of approved firmware is installed on their vehicles. As a result, the firmware verification occurs externally on the OEM's servers while our approach focuses on an internal solution within the vehicle. Additionally, their implementation involves the use of a hash table distributed among the ECUs in the vehicle. Their blockchain considers each vehicle (i.e., head unit, the ECU in the vehicle that interacts with the external blockchain) as a node, collecting information from all ECUs

in the vehicle. If an adversary compromises the head unit, then the entire vehicle can be disconnected from the OEM blockchain, thus allowing the rest of the vehicle to be attacked. Furthermore, since the distributed hash table stores an ECU's firmware information on only a few other ECUs in the vehicle, an adversary needs to compromise a small number of ECUs to maliciously change a target ECU's firmware. Our work focuses on an in-vehicle solution that operates locally in the vehicle to ensure ECU integrity.

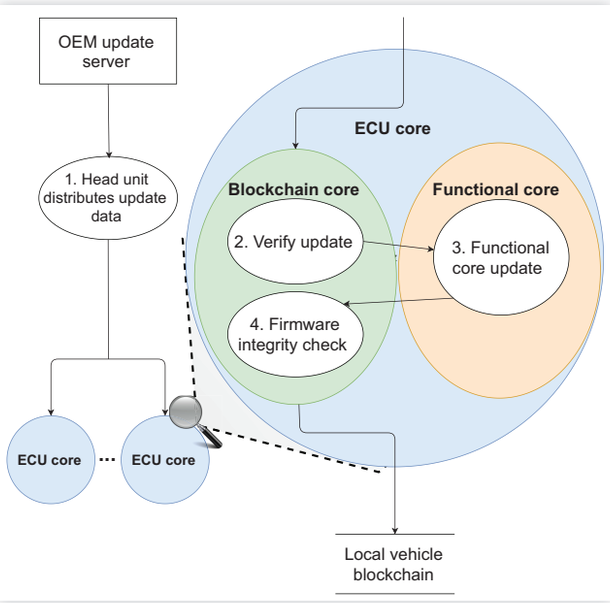
Methodology

This section overviews the proposed blockchain approach and describes how it can be used to ensure ECU integrity. Different ECUs in the vehicle have different architectures and computational resources. As such, we classify them into two categories of models: *Full Architecture* and *Limited Architecture*. Full Architecture refers to subsystem ECUs with computing resources similar to that of traditional desktop computers; example subsystems include Lane Management, Infotainment, and Navigation. Limited Architecture refers to ECUs with limited resources such as those used for the Tire Pressure Monitoring System.

Proposed Blockchain Framework

Most onboard ECUs have underutilized central processing unit (CPU) cores [38, 39, 40, 41, 42] and storage [43, 44, 45]. Inspired by the *Authentication Layer Approach* [19], we propose a software-based variation of the approach to be deployed across ECUs. Specifically, the *authentication layer* can be implemented on each ECU without additional hardware through software virtualization, Role-Based Access Control, and sandboxing [6]. System integrity is verified by a root process, whereas system functionality is performed as a less-privileged user.

FIGURE 2 A CFD for the blockchain approach to secure ECU integrity. Delivery of OTA data from the OEM update server to the vehicle is assumed to use proprietary methods or third-party technologies such as Uptane. The blue ellipse indicates a single ECU blockchain core's process to verify an update's integrity and its installation.



Data taken from https://www.blockchain.com/btc/block/411940. © SAE International

Figure 2 shows a Control/work Flow Diagram (CFD) for our blockchain approach to secure ECU firmware, where ellipses represent computations, arrows indicate workflows, data stores (i.e., persistent data) are denoted with double lines, and external entities are represented by rectangles. Each ECU is represented with a blue ellipse where we use the magnifying icon to expand the view of a representative ECU, revealing the workflow between the respective blockchain processing elements in a blockchain core and its corresponding functional core. Next, each of the key steps for our blockchain approach is described, where diagram elements are italicized for clarity.

1. *Head Unit Distributes Update Data*. Update information is transferred from the *OEM Update Server* to the head unit (i.e., the unit that receives the update) of the vehicle via a proprietary method [10] or state-of-the-art technique such as Uptane. The head unit then distributes the update data to the blockchain network.
2. Verify Update. Data distributed by the head unit is received by each ECU's *blockchain core*. Each ECU in the blockchain network then votes (via the CAN bus) to ensure the safety and correctness of the software data. If the ECU network votes that the update is safe and correct, then the update is delivered to the *ECU Functional Core* for installation. If the ECU network votes that the update is incorrect, then the incorrect firmware data is added to the *Local Vehicle Blockchain*, enabling the information to be subsequently analyzed by the OEM.

3. *Functional Core Update*. Once the update has been verified safe by the blockchain network, the update data is installed on the *Functional Core*.
4. *Firmware Integrity Check*. After the *Functional Core* installs the update, then the *Blockchain Core* verifies the integrity of the new firmware results and adds them to the *Local Vehicle Blockchain*. To ensure ECU integrity during vehicle operation, the *Blockchain Core* periodically checks the firmware of the *Functional Core* every *T* milliseconds. The results are then added to the *Local Vehicle Blockchain*. If the ECU network votes that the update or firmware is not safe or has been compromised, then the vehicle will notify the human driver and mitigate the invalid firmware. The invalid firmware information is then pushed onto the *Local Vehicle Blockchain*, thus enabling the OEM to analyze the corrupted or malicious firmware.

A blockchain approach has several key advantages over other security approaches. First, the blockchain removes a single point of failure by decentralization. If ECUs only verify their own firmware during the boot process, then a compromised ECU can potentially ignore or fabricate the verification. By requiring other ECUs to verify the firmware information, nonmalicious ECUs can detect the malicious firmware of a compromised ECU. Second, an adversary cannot replace an ECU (digitally via firmware or physically) as the blockchain network will detect the change. Furthermore, messages that are sent by malicious ECUs can be recorded on the blockchain and analyzed by the OEM in the future, should an attack occur. Since data on the blockchain is resistant to modification, an adversary cannot remove a malicious record from the blockchain to conceal the evidence of their attack. Finally, correct firmware information for each ECU is persistent on the blockchain. As such, the blockchain approach is able to verify firmware and detect attacks while the vehicle is in operation, rather than only during secure boot.

Securing vehicular ECUs is difficult as the architectures of the ECUs have a wide range of capacity for computations. From smaller microprocessor ECUs using single-core processors with constrained resources (i.e., Limited-Architecture) to architectures more akin to desktop computers with x86 or ARM processors with gigabytes of RAM (i.e., Full Architecture), each ECU requires a tailored approach.

Both the Full-Architecture and Limited-Architecture models generate the same blockchain messages. Table 2 shows the blockchain message structure used by both models. Each message contains only 128 bytes of data, but through the use of nonces (i.e., number of leading zeroes), the overall message size can be reduced.

Full Architecture

The *Full-Architecture* model follows a traditional desktop computing architecture with multicore support [38, 39, 40, 41]. The additional computational resources also provide methods for increased security. Using a common security practice known as *process sandboxing* [46], we can create a resilient method to verify ECU firmware integrity. Sandboxing implies if one process is compromised, then it will have no effect on other

TABLE 2 Blockchain message.

Property	Type	Size (bytes)
ID	long	8
CreatedTime	long	8
BootTime	long	8
Nonce	long	8
FirmwareHash	byte[]	32
PreviousHash	byte[]	32
CurrentHash	byte[]	32
	Total	**128**

processes due to process isolation. Furthermore, through the use of a hypervisor layer, processes share hardware resources, thus alleviating the need for additional hardware.

Similar to the *Authentication Layer Approach* [19], we propose the use of sandboxing for the ECU processes. Sandboxing the ECU process isolates the functional code, thus allowing for firmware inspection by the blockchain thread. A software virtualization approach also eliminates additional hardware requirements, a primary concern of automotive manufacturers. However, a hardware-based approach does have advantages over this model. Specifically, there have been known attacks or malware with the ability to detect and break out of the sandboxed environments [47]. However, as the blockchain thread constantly checks the state of the functional thread, it will have the ability to detect the malicious payload.

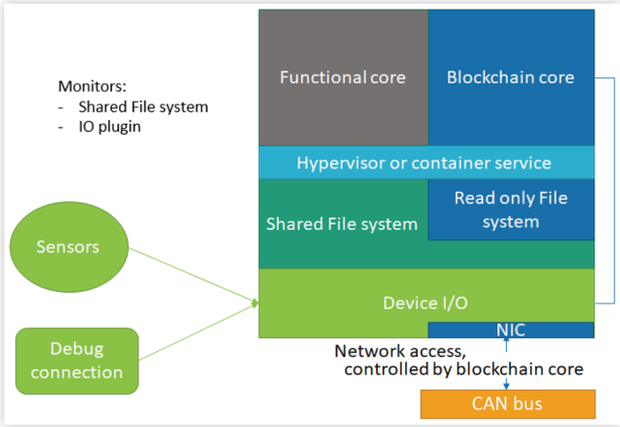
Furthermore, our approach proposes an architecture based on a hypervisor or a container-based service [48]. Container services play the role of a traditional virtual machine. Since the container shares similar components, the total overhead is reduced by removing redundant system resources. Using the additional processor cores, a containerized service will be implemented to run the pre-existing functional code. An additional processor core will be dedicated to facilitating blockchain requirements.

Another advantage of a blockchain approach is that the blockchain thread controls the Network Interface Card where, in many automotive systems, it is connected to the CAN bus. The *authentication layer* protects the rest of the automotive network from a malicious functional thread that is broadcasting messages since the functional thread will not be able to spoof messages. The full system architecture is shown in Figure 3. With multicore ECU processors, blockchain code will be deployed to execute on underutilized CPU cores without adversely affecting system performance [39]. This distributed approach imposes no additional requirements for vehicle CPUs (i.e., centralized trust authority), thus allowing for ECUs by different manufacturers to be authenticated by the same layer.

Full-Architecture ECUs often run a form of the Linux Operating System (OS) [49], such as Automotive Grade Linux [49, 50]. Linux allows for whitelisting processes, thereby allowing only trusted processes to execute. If an adversary is able to load a script onto an ECU, then they would additionally have to compromise the Linux OS to execute an untrusted application. Additional resources must be checked and verified against the blockchain, such as the list of scripts and applications. The data stored in the blockchain for these scripts can be hashed to ensure no modifications have occurred, as well as reducing the size of the transactions stored on the blockchain.

Full-Architecture ECUs, such as the infotainment system, have significant computational resources and large storage capacity. As such, these ECUs should store the full blockchain until the reserved storage is full. Once the storage capacity has been reached, the oldest blocks will be removed when adding new blocks to the blockchain. As the verification of the blockchain only involves the latest blocks, removing the oldest blocks will not affect the verification process.

FIGURE 3 Architecture for Full-Architecture ECUs.



Limited Architecture

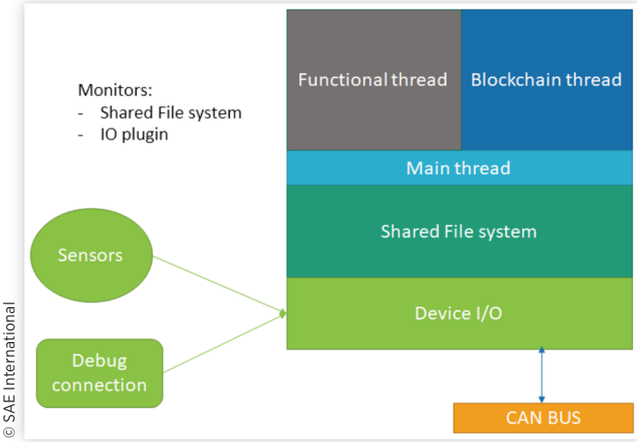
While Full-Architecture ECUs have high computing capability, other ECUs have resource-constrained microprocessors as their primary CPU (e.g., power electric module). These ECUs are resource constrained and have minimal overhead to reduce the latency of the real-time processes. Thus traditional computer security techniques, such as an intrusion detection system, firewall, Anti-Virus, or message encryption, are too resource intensive for use on Limited-Architecture ECUs.

A major attack vector targeting Limited Architectures includes buffer overflow [51] and distributed denial of service. Buffer overflow is the most common vulnerability identified among attacks on industrial automation systems [52]. Since Limited-Architecture ECUs have limited computational

resources to detect and mitigate attacks, these devices may be perpetrated through attack vectors such as buffer overflow as it requires little domain knowledge.

While a number of techniques have been developed to prevent buffer-overflow attacks, most assume access to resources beyond that available to onboard ECUs. As such, to secure these ECUs, security by design must be a manufacturer requirement to minimize the attack surface. Despite their limited resources, a lightweight blockchain structure can still be implemented on Limited-Architecture ECUs. To realize the lightweight blockchain, the Limited-Architecture ECU processor, in essence, allocates two threads. The first thread is the *Functional* thread that provides the functional objective of the ECU. For example, the brake actuator's functional thread is responsible for controlling the brakes. A secondary thread, the *Blockchain* thread, serves to broadcast and handle messages related to the blockchain. The blockchain thread captures system snapshots and broadcasts them to the blockchain network. Figure 4 shows the Limited-Architecture model.

FIGURE 4 Architecture for Limited-Architecture ECUs.



As an aside, we acknowledge that there are ECUs with even more limited computational resources and little to no storage, termed basic nodes (e.g., nodes on the Local Interconnect Network bus [53]). These nodes can still participate in the verification of their firmware by sending their firmware data for other ECUs in the blockchain network to verify, leveraging their extra resources. However, basic nodes do not contribute to the voting processes of the blockchain approach due to their resource limitations.

Blockchain Verification

In both architecture models, the blockchain is distributed among all or a subset of ECUs in the vehicle. When an ECU first boots, a message is created that includes the system firmware hash, firmware version, ECU up-time, message time, and the previous hash. The exact block contents vary based on the system architecture. The message is then broadcasted to the blockchain network where the network votes if the ECUs system information matches the blockchain's record for that ECU through a consensus algorithm. Checking at boot time also verifies that the firmware has not been modified while the system was off, as the firmware hash will not match the last recorded hash if it has been modified.

If the system information is valid, then the message is added to the blockchain. Adding the new message to the blockchain increases the length of the blockchain, which in turn increases the computational security of the blockchain. If the system information is invalid, then the ECU has been compromised or corrupted. Additional mitigation strategies must be enacted to contain the malicious behavior of the ECU.

To ensure ECUs are not compromised during vehicular operation, each ECU will periodically send messages to the blockchain. Every T milliseconds, a new message should be sent from each ECU to the blockchain network. If an ECU does not respond within an acceptable variance of T, then that ECU is no longer trusted by the network. Furthermore, if an ECU loses power, or is functionally damaged, then the network still assumes the ECU has been compromised. Once the ECU has been flagged as compromised, additional measures must be taken to ensure the node is safe before reintegrating it back to the network or replacing the node. The value of T may vary for each ECU depending on system capabilities and power consumption. Using this model, the blockchain distributes trust among ECUs in the vehicle and behaves similarly to an intrusion detection system.

Architecture Interactions

Given the difference in resources between ECUs, blockchain roles should be delegated between Full-Architecture and Limited-Architecture ECUs accordingly. In current implementations of blockchain such as Bitcoin [33], nodes that compete to compute hash

values and create blocks are known as *miners*. Miners are often equipped with chipsets that are efficient for computing hashes, such as Graphics Processing Units. Next, nodes such as a typical user seeking to add transactions or obtain information from the blockchain only store a portion of the blockchain. These nodes rely on the miners to maintain the blockchain. Finally, there are entities that store the blockchain ledger. The proposed architecture setup is analogous to current implementations of blockchain networks [33]. Specifically, Full-Architecture ECUs perform the role of a blockchain miner. Full-Architecture ECUs have additional unused system resources that may be used to decrease the strain on Limited-Architecture ECUs. Limited-Architecture ECUs have limited resources; thus they should only be creating transactions and store a limited portion of the blockchain needed for firmware verification. By delegating the roles of the blockchain accordingly, a lightweight blockchain can be deployed on the ECU network in a vehicle to ensure ECU integrity.

A blockchain approach secures ECUs on a vehicle while remaining resilient towards modification attacks. As the blockchain uses secure cryptographic hashing, forging transactions on the blockchain is computationally infeasible. Additionally, to control the ledger, at least 50% of the blockchain members must be compromised before the ledger can be maliciously modified [54]. To compromise the proposed blockchain, an adversary must either add malicious computational resources on the vehicle or compromise more than half of the ECUs on the vehicle at the same time. As the former requires physical access and the latter requires more than half of the vehicle to be compromised, compromising the blockchain framework requires the adversary to have unrealistic capabilities. Finally, the blockchain will be initially created by the vehicle manufacturer. All ECUs at creation are assumed to be trusted as they come directly from the OEM.

Different ECUs on the vehicle have different storage capacities. For Limited-Architecture ECUs, only the most recent blocks of the blockchain are stored. As such, if an adversary manages to modify the entire blockchain section stored on a Limited-Architecture ECU with a 50% attack, then the ECU votes incorrectly. However, the blockchain approach can still detect the malicious ECU firmware when an adversary controls more than 50% of the blockchain members. Specifically, Full-Architecture ECUs have more storage capacity, thus allowing them to store a longer blockchain (i.e., a longer history of the vehicle's ECU firmware). If an adversary controls more than 50% of the blockchain and successfully modifies or adds malicious blocks to the blockchain, then Limited-Architecture ECUs may not detect the attack. However, Full-Architecture ECUs have access to more blocks, thus allowing them to identify the maliciously modified firmware data. When Full-Architecture ECUs detect the malicious firmware data, a dissenting vote is broadcasted, notifying the network of an attack. Even when most of the network votes incorrectly, a single dissenting vote indicates disagreement in firmware integrity. Thus the blockchain approach is able to detect the attack. As a result, to successfully modify ECU firmware, an adversary must control more than 50% of the blockchain and compromise all Full-Architecture ECUs.

Handling OTA Updates

ECUs check against the blockchain to ensure that OTA updates are valid for other ECUs. Updates should be signed with a certificate, using approaches such as the recent work by Kent et al. [18]. To ensure the safety of passengers and the vehicle's environment, updates should only occur when the engine is off.

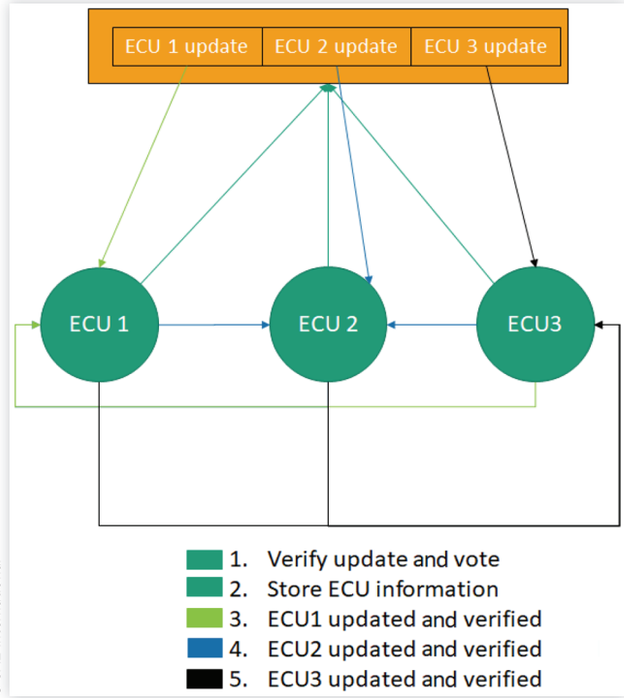
To ensure valid firmware is accepted by the blockchain and ECUs, firmware must be installed through the following process. First, each ECU verifies the update authenticity. Second, all ECUs vote if the update is valid. If the update is voted to be safe, then ECUs temporarily store the new firmware version. ECUs are then updated synchronously. Once an ECU is updated, it creates a new message, and all other ECUs check if the new version matches the update log for each ECU. If the network votes that the ECU was updated properly, then the new firmware version information message is added to the blockchain. This update process is shown with three ECUs in Figure 5. Note that the

ordering of ECU updates does not matter. In fact, a randomized order would make spoofing an update more difficult for an adversary.

Replacing or Adding an ECU

During the lifetime of a vehicle, ECUs may be added or replaced on the vehicle. In a blockchain approach, there must be an amendment process to introduce or remove ECUs. Specifically, ECUs on the vehicle blockchain must attest to the addition or removal of the new ECU(s) via a consensus algorithm. For the addition of new ECU(s), the new ECU(s) to be installed must have an OEM or trusted supplier-provided private key to enable existing ECUs to verify if the new ECU(s) can be trusted. Similar to current blockchain implementations, new ECUs added to the blockchain must contain a copy of the existing blockchain. The copy of the existing blockchain is often simply copied over from other nodes in the network to the new node as it joins the blockchain network.

FIGURE 5 Update process for ECU firmware.



Implementation

As a proof of concept demonstration of our approach, we implemented our blockchain framework using a set of Particle Core devices.[2] The Particle Core is a low-level IoT device with an embedded Wi-Fi chip. As shown in Figure 6, the Particle Core is a resource-constrained device with limited memory similar to that of a Limited-Architecture ECU. For proof of concept, we use a set of resource-constrained devices and show the feasibility of our approach. Since the ECUs of the Full-Architecture model have more computational resources than those of the Limited-Architecture model, the results of the experiment apply to both models of ECUs. The Particle Core is programmed in C/C++ with access to the C++ standard library.[3] Particle provides libraries to automatically communicate with the Particle Cloud, a web service that provides communication and management of Particle Devices. The test framework comprises four Particle Cores, communicating wirelessly, as shown in Figure 7. Following is a high-level specification of the configuration.

Particle Core

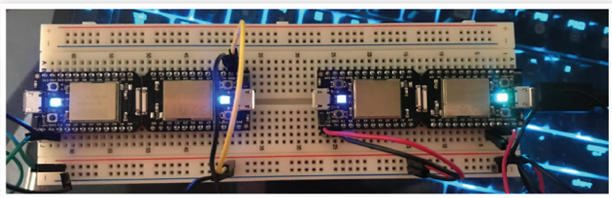
One major concern regarding the hash function SHA256 is the computational complexity and computational time. Given that many ECUs have lightweight processors, SHA256 may not be feasible for Limited-Architecture processors [6]. In order to assess this concern, a variety of message sizes (in bytes) were computed on the Particle Cores. The SHA256 algorithm has been provided by the courtesy of Oliver Gay[4] and was modified to work with Particle's String data structure. Figure 8 shows the hashing times in milliseconds for the corresponding message sizes. To ensure consistent results, each message size was computed 1000 times and averaged. The 95% confidence interval is sufficiently small; thus they are not included. Notably, even

FIGURE 6 Particle core specification.

- ARM 32-bit Cortex™-M3 CPU Core
- 72 MHz operating frequency, 1.25 DMIPS/MHz (Dhrystone 2.1)
- 128 kB of Flash memory
- 20 kB of SRAM
- 12 bit ADC
- USB 2.0 full-speed interface
- USART, SPI and I2C interfaces
- JTAG Debug mode

[2] Particle, "Your edge-to-cloud IoT platform," https://www.particle.io/.
[3] Particle, "Photon device OS API," https://docs.particle.io/reference/device-os/firmware/photon/.
[4] O. Gay, "C++ sha256 function," http://www.zedwood.com/article/cpp-sha256-function.

FIGURE 7 Particle core test bed.



for a larger message (e.g., 1 kB), the Particle Core computed the hash at an average of 0.952 milliseconds.

Another concern is the trade-off between storage and computational complexity. As previously discussed, nonces can be introduced to limit the size of hash values on the blockchain. However, enumerating nonces will add additional computational delay. To test the feasibility of the hashing difficulty, a series of tests measure the time to find hashes that comply with the targeted difficulty. The log scale in Figure 9 demonstrates that the data grows in $\mathbf{O(10^n)}$. With a difficulty of five, the time to find a nonce required approximately five minutes. In a real-time system, it is evident that storage complexity must be favored over computational complexity and thus requiring high-level ECUs to store larger portions of the blockchain ledger.

Modification Detection Experiments

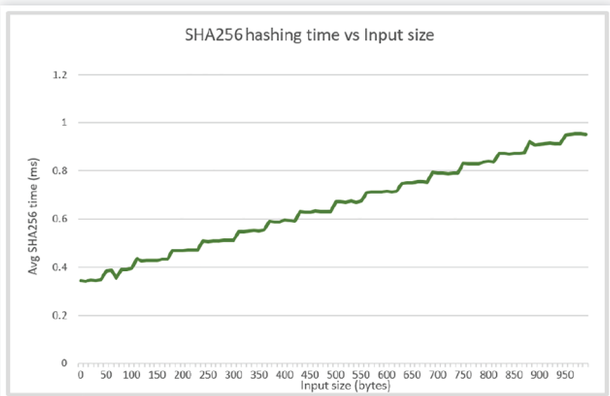
The framework presented in this paper is designed to detect malicious modifications to ECU firmware. To assess the feasibility of our approach, we created several experimental setups, each compromising a different number and/or type of cores/devices (e.g., functional vs. blockchain core). In each experiment, the four Particle Cores each contain a functional core and a blockchain core. Malicious blockchain votes are denoted in **bold red text** and legitimate blockchain votes are denoted in *bold blue Italics*.

1. Baseline Experiment. In the first experiment, no Particle Core is compromised. An experiment with no compromised core shows the normal operations of the blockchain. During each broadcast interval, all devices broadcast their firmware information for all other devices to verify. All devices then vote, indicating whether the firmware matches the version stored on the blockchain. Since no device is compromised, the blockchain cores broadcast a vote that the Firmware hashes match; see the ***blue text*** in the output of a Particle Core's vote shown in Program 1.

2. Compromising 1-4 functional cores. In the second experiment, we first modify one device's functional core with a malicious firmware. As shown in Table 3, the FirmwareHash has changed (see ***blue*** and **red** text to respectively indicate valid and compromised firmware hashes). The new message is broadcasted to the blockchain network. The data is then verified against the blockchain's record by each device's blockchain core. Since the malicious firmware will not match the firmware information stored on the blockchain, all four devices' blockchain cores (including the device whose functional core has been compromised) detect the malicious firmware for device ID …547373336323230. The Particle Cores broadcast their votes indicating that the FirmwareHash does not match for the compromised device, as shown in Program 2. As we compromise more functional cores, the blockchain cores are able to detect the malicious firmware and broadcast similar messages, indicating that the corresponding functional cores have been compromised. Through the blockchain core's consensus protocol, the network is able to identify when a firmware is modified in an unauthorized manner.

3. Compromising 1-2 blockchain cores. In the third experiment, in addition to modifying one device's functional core with a malicious version, we also modify the blockchain core of the same device to broadcast a malicious vote. With malicious blockchain core(s) in the blockchain, malicious votes indicating that the malicious firmware hash matches stored hash are observed in the blockchain votes (see Program 3). Program 3(a) shows the malicious vote of a single compromised blockchain core (see **red**

FIGURE 8 SHA256 hashing times on particle core.



(a) Average hashing time vs input size (bytes)

Size (bytes)	AVG time (ms)
10	0.344
100	0.396
500	0.671
1000	0.952

(b) Input size vs hashing time

text). However, even though the other three devices detect the invalid firmware, the network is still able to detect a device that has been compromised as the devices reach consensus (see ***blue*** text).

Next, we modify half of the devices to contain malicious functional and blockchain cores. As half of the blockchain nodes are compromised, half the blockchain maliciously votes to indicate the firmware hash matches, while the nonmalicious nodes in the blockchain vote that they do not match Program 3(b) shows the votes that are broadcasted by the blockchain cores regarding the validity of the compromised core's firmware data (again, **red** and ***blue*** indicate malicious and legitimate votes, respectively). Since the blockchain protocol cannot reach a consensus on the validity of the firmware hash, the results reaffirm that the approach can detect suspicious network activity even with fifty percent of the network compromised.

4. Compromising 4 functional and 3 blockchain cores. In the last experiment, we modify all four devices' functional cores to receive a malicious firmware update, and we compromise three of the blockchain cores to vote maliciously. Modifying three blockchain cores (i.e., more than half of the blockchain nodes) to vote maliciously implies that the blockchain consensus should maliciously agree that the firmware hash matches the stored version (see the votes in Program 4). With more than half the network compromised, the malicious cores have the majority in the consensus algorithm. Since the noncompromised core broadcasts a legitimate vote that it has an invalid firmware message (line 17 in Program 4), an external entity reading the blockchain data can still ascertain a compromised blockchain node via the vote discrepancies.

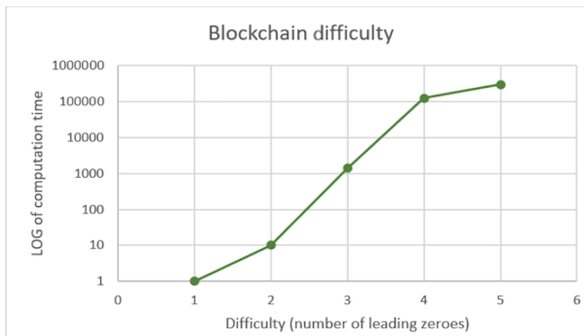
Conclusions

Automotive cybersecurity is an emerging vulnerability [3, 6]; thus security by design [55, 56, 57, 58] must be incorporated to minimize cybersecurity vulnerability. This article introduced a software-based blockchain framework to secure unauthorized modifications to onboard ECUs. The proposed framework can be used to address the security of ECUs updates originating from a variety of sources, including OTA updates, firmware flashing, etc. Additionally, as low-level ECUs are at risk against unknown threats [6], the proposed approach provides a method to identify and capture unauthorized firmware modification without incurring prohibitive performance costs.

Several areas are being considered for future work. For example, an area open for research is the interaction between High-Level and Low-Level ECUs for blockchain management. Specifically, different distributions of roles for blockchain ledger management can be explored to optimize computational resource usage. Limited resource networks such as Local Interconnect Network [53] buses can be further explored for their potential use with the blockchain approach. Additionally, the hashing difficulty on High-Level ECUs should be verified as it may potentially eliminate the need to hash on Low-Level ECUs.

We are continuing to work with our OEMs and Tier 1 supplier collaborators with exploring the next steps of this work, including applications to other attack vectors.

FIGURE 9 SHA256 hashing times for varying difficulties.



(a) Time vs Hashing Difficulty, in Log 10 scale

Difficulty	Time (ms)	Hash
1	1	0e32f39dc5b533cfaefeb15d9956e15b6452327eef86974408c5308d17800a4d
2	10	00dfb5dc0674cc509c2afdfaae71bb9338a89496a73c838cd4a2c88ba11a1dfc
3	1381	000f1729c1cec8c6d3814fdfb0989eaf4151c7bcca2b7dddad7ee06981e90847
4	124712	0000fd7b49ef7c8ab2b265c14d633fcee36a03ac209384d3ee8395f7ad9fa281
5	295652	000002b7db4557bee49e054ffd586194056ddd3bab4dc27e37e1293c39310ef6

(b) Time vs hashing difficulty with corresponding number of leading zeroes

PROGRAM 1 Output when no cores are compromised.

```
1  Dev:...547373336323230
2  Firmware matches stored Firmware;
3  Hash: 0d970313219e0bac056419d62e35c2b6
4        47efa3db85c46aa1203bbe649fd4da7b
```

TABLE 3 Blockchain message comparison.

Property	Valid	Attacked
ID	...8505553211367	...8505553211367
CreatedTime	1375	1375
BootTime	0	0
Nonce	0	0
FirmwareHash	sha256(Firmware 1.0)	sha256(Firmware 1.1)
PreviousHash	...	...
CurrentHash	...	...
	Total	**128**

PROGRAM 2 Output of the Particle Core with a compromised functional firmware.

```
1  Dev:...547373336323230
2  Firmware does not match stored Firmware
3  Hash: fbf2eddc25eb84e9f840dfd50e064f73
4        ec09547f968f67b92f26f6dc8b4a3831
```

PROGRAM 3 Voting results from compromised blockchain cores.

```
1  Dev:...547373336323230
2  Firmware matches stored Firmware;
3  Hash: 3ce8e7f304c64df033e71feb62b7d236
4        64c8d9ed79473f96501b48a73658d2e0
5
6  Dev:...947363335343832
7  Firmware does not match stored;
8  Hash: 3ce8e7f304c64df033e71feb62b7d236
9        64c8d9ed79473f96501b48a73658d2e0
10
11 Dev:...e47373334323233
12 Firmware does not match stored;
13 Hash: 3ce8e7f304c64df033e71feb62b7d236
14       64c8d9ed79473f96501b48a73658d2e0
15
16 Dev:...747373336323230
17 Firmware does not match stored;
18 Hash: 3ce8e7f304c64df033e71feb62b7d236
19       64c8d9ed79473f96501b48a73658d2e0
```

(a) Voting results with one malicious blockchain

```
1  Dev:...547373336323230
2  Firmware Matches stored Firmware;
3  Hash: 9c4e6389a201ea78a05c6ef593ea519b
4        d36e9647f8b8af5da7df41ecbef20485
5
6  Dev:...947363335343832
7  Firmware Matches stored Firmware;
8  Hash: 9c4e6389a201ea78a05c6ef593ea519b
9        d36e9647f8b8af5da7df41ecbef20485
10
11 Dev:...e47373334323233
12 Firmware does not match stored;
13 Hash: 9c4e6389a201ea78a05c6ef593ea519b
14       d36e9647f8b8af5da7df41ecbef20485
15
16 Dev:...747373336323230
17 Firmware does not match stored;
18 Hash: 9c4e6389a201ea78a05c6ef593ea519b
19       d36e9647f8b8af5da7df41ecbef20485
```

(b) Voting results with half of blockchain cores compromised

PROGRAM 4 Even with three out of four functional cores compromised, a single dissenting vote is still broadcasted.

```
Dev:...547373336323230
Firmware Matches stored Firmware;
Hash: e212e1a7c8197845cf88c7666b4a252d
      8fb91bc54a1343cb69a36567014d8139

Dev:...947363335343832
Firmware Matches stored Firmware;
Hash: e212e1a7c8197845cf88c7666b4a252d
      8fb91bc54a1343cb69a36567014d8139

Dev:...e47373334323233
Firmware Matches stored Firmware;
Hash: e212e1a7c8197845cf88c7666b4a252d
      8fb91bc54a1343cb69a36567014d8139

Dev:...747373336323230
Firmware does not match stored;
Hash: e212e1a7c8197845cf88c7666b4a252d
      8fb91bc54a1343cb69a36567014d8139
```

Contact Information

Dr. Betty H.C. Cheng
chengb@msu.edu

References

1. Daimi, K., Saed, M., Bone, S., and Robb, J., "Securing Vehicle's Electronic Control Units," in *ICNS 2016*, Herndon, VA, 2016.
2. Avatefipour, O. and Malik, H., "State-of-the-Art Survey on In-Vehicle Network Communication (CAN-Bus) Security and Vulnerabilities," arXiv preprint arXiv:1802.01725, 2018.
3. Checkoway, S., McCoy, D., Kantor, B., Anderson, D. et al., "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *USENIX Security Symposium*, San Francisco, CA, vol. 4, 447-462, 2011.
4. Greenberg, A., "Hackers Remotely Kill a Jeep on the Highway—With Me in It," *Wired* 7 (2015): 21.
5. Miller, C. and Valasek, C., "Remote Exploitation of an Unaltered Passenger Vehicle," Black Hat USA, vol. 2015, no. S 91, 2015.
6. Parkinson, S., Ward, P., Wilson, K., and Miller, J., "Cyber Threats Facing Autonomous and Connected Vehicles: Future Challenges," *IEEE Transactions on Intelligent Transportation Systems* 18, no. 11 (2017): 2898-2915.
7. McNair, B.E., "Centralized Security Control System," U.S. Patent 5,276,444, January 4, 1994.
8. Rowland, C.H., "Intrusion Detection System," U.S. Patent 6,405,318, June 11, 2002.
9. Schmittner, C. and Macher, G., "Automotive Cybersecurity Standards—Relation and Overview," in *Computer Safety, Reliability, and Security (SAFECOMP 2019)*, Lecture Notes in Computer Science (Romanovsky, A., Troubitsyna, E., Gashi, I., Schoitsch, E. et al., eds.), vol. 11699 (Cham: Springer, 2019).

10. Cheng, B.H.C., "Research into Securing OTA Updates and ECUs," Personal Communication with Ford Motor Company, January 2021.
11. Kuppusamy, T.K., DeLong, L.A., and Cappos, J., "Uptane: Security and Customizability of Software Updates for Vehicles," *IEEE Vehicular Technology Magazine* 13 (2018): 66-73.
12. Füst, S., Mössinger, J., Bunzel, S., Weber, T. et al., "AUTOSAR—A Worldwide Standard Is on the Road," in *14th International VDI Congress Electronic Systems for Vehicles*, Baden-Baden, vol. 62, 5, 2009.
13. Baza, M., Nabil, M., Lasla, N., Fidan, K. et al., "Blockchain-Based Firmware Update Scheme Tailored for Autonomous Vehicles," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, Marrakech, Morocco, 1-7, IEEE, 2019.
14. Falco, G. and Siegel, J.E., "Assuring Automotive Data and Software Integrity Employing Distributed Hash Tables and Blockchain," CoRR, vol. abs/2002.02780, 2020.
15. Deshpande, V., George, L., and Badis, H., "SaFe: A Blockchain and Secure Element Based Framework for Safeguarding Smart Vehicles," in *2019 12th IFIP Wireless and Mobile Networking Conference (WMNC)*, Paris, France, 181-188, September 2019.
16. Alam, M., Iqbal, S., Zulkernine, M., and Liem, C., "Securing Vehicle ECU Communications and Stored Data," in *ICC 2019—2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, 1-6, May 2019.
17. Salem, M., Mohammed, M., and Rodan, A., "Security Approach for In-Vehicle Networking Using Blockchain Technology," in *Advances in Internet, Data and Web Technologies* (Barolli, L., Xhafa, F., Khan, Z.A., and Odhabi, H., eds.) (Cham: Springer International Publishing, 2019), 504-515.
18. Kent, D., Cheng, B.H.C., and Siegel, J., "Assuring Vehicle Update Integrity Using Asymmetric Public Key Infrastructure (PKI) and Public Key Cryptography (PKC)," *SAE Int. J. Transp. Cyber. & Privacy* 2, no. 2 (2019): 141-158, doi:https://doi.org/10.4271/11-02-02-0013.
19. Banerjee, M., Lee, J., Chen, Q., and Choo, K.-K.R., "Blockchain-Based Security Layer for Identification and Isolation of Malicious Things in IoT: A Conceptual Design," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, Hangzhou, China, 1-6, IEEE, 2018.
20. ISO Standard, "ISO 11898: Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication," 1993.
21. Nilsson, D.K., Phung, P.H., and Larson, U., "Vehicle ECU Classification Based on Safety-Security Characteristics," in *IET Road Transport Information and Control—RTIC 2008 and ITS United Kingdom Members' Conference*, Manchester, 1-7, June 2008.
22. Mahmud, S.M., Shanker, S., and Hossain, I., "Secure Software Upload in an Intelligent Vehicle via Wireless Communication Links," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005*, Las Vegas, NV, 588-593, 2005.
23. Mansour, K., Farag, W., and ElHelw, M., "AiroDiag: A Sophisticated Tool That Diagnoses and Updates Vehicles Software over Air," in *2012 IEEE International Electric Vehicle Conference*, Greenville, SC, 1-7, 2012.
24. Steger, M., Karner, M., Hillebrand, J., Rom, W. et al., "Generic Framework Enabling Secure and Efficient Automotive Wireless SW Updates," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Berlin, Germany, 1-8, 2016.
25. Halder, S., Ghosal, A., and Conti, M., "Secure Over-the-Air Software Updates in Connected Vehicles: A Survey," *Computer Networks* 178 (2020): 107343.
26. Carsten, P., Andel, T.R., Yampolskiy, M., and McDonald, J.T., "In-Vehicle Networks: Attacks, Vulnerabilities, and Proposed Solutions," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference, CISR '15*, New York, Association for Computing Machinery, 2015.
27. Wang, Q., "Secure Boot for Vehicular Systems," U.S. Patent 9,792,440, October 17, 2017.
28. Sanwald, S., Kaneti, L., Stottinger, M., and Bohner, M., "Secure Boot Revisited: Challenges for Secure Implementations in the Automotive Domain," *SAE Int. J. Transp. Cyber. & Privacy* 2, no. 2 (2020): 69-81, doi:https://doi.org/10.4271/11-02-02-0008.

29. Ueda, H., Kurachi, R., Takada, H., Mizutani, T. et al., "Security Authentication System for In-Vehicle Network," *SEI Technical Review* 81 (2015): 5-9.
30. Wang, Q. and Sawhney, S., "VeCure: A Practical Security Framework to Protect the Can Bus of Vehicles," in *2014 International Conference on the Internet of Things (IOT)*, Cambridge, MA, 13-18, 2014.
31. Farag, W.A., "CANTrack: Enhancing Automotive CAN Bus Security Using Intuitive Encryption Algorithms," in *2017 7th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO)*, Marrakesh, Morocco, 1-5, 2017.
32. Koushanfar, F., Sadeghi, A.-R., and Seudie, H., "EDA for Secure and Dependable Cybercars: Challenges and Opportunities," in *Proceedings of the 49th Annual Design Automation Conference*, San Francisco, CA, 220-228, ACM, 2012.
33. Nakamoto, S., "Bitcoin: A Peer-to-Peer Electronic Cash System," Oct. 31, 2008, https://bitcoin.org/bitcoin.pdf, also available at https://git.dhimmel.com/bitcoin-whitepaper/, accessed Aug. 2, 2020.
34. Aggarwal, D., Brennen, G., Lee, T., Santha, M. et al., "Quantum Attacks on Bit-Coin, and How to Protect against Them," *Ledger* 3:68-90, (2018).
35. Baliga, A., "Understanding Blockchain Consensus Models," *Persistent* 2017, no. 4 (2017): 1-14.
36. He, X., Alqahtani, S., Gamble, R., and Papa, M., "Securing Over-The-Air IoT Firmware Updates Using Blockchain," in *Proceedings of the International Conference on Omni-Layer Intelligent Systems, COINS2019*, New York, 164-171, Association for Computing Machinery, 2019.
37. Yohan, A., Lo, N.-W., and Achawapong, S., "Blockchain-Based Firmware Update Framework for Internet-of-Things Environment," in *Proceedings of the International Conference on Information and Knowledge Engineering (IKE)*, Las Vegas, NV, 151-155, The Steering Committee of The World Congress in Computer Science, Computer Engineering, & Applied Computing, 2018.
38. Bjelica, M.Z. and Lukac, Z., "Central Vehicle Computer Design: Software Taking Over," *IEEE Consumer Electronics Magazine* 8 (2019): 84-90.
39. Cheng, B.H.C., "Research into Onboard Multicore Resources," Personal Communication with Ford Motor Company, August 2015.
40. Gai, P. and Violante, M., "Automotive Embedded Software Architecture in the Multi-core Age," in *2016 21th IEEE European Test Symposium (ETS)*, Amsterdam, the Netherlands, 1-8, May 2016.
41. Goebel, A., Mader, R., and Tripon, O., "Performance and Freedom from Interference—A Contradiction in Embedded Automotive Multi-core Applications?," in *ARCS2017; 30th International Conference on Architecture of Computing Systems*, Vienna, Austria, 1-9, April 2017.
42. Brooke, L., "Aptiv Centralizing Computing Power to Speed Autonomous Vehicle Development," Dec. 11, 2019, https://www.sae.org/news/2019/12/aptiv-centralizing-av-computing, accessed Aug. 2, 2020.
43. Van den Herrewegen, J. and Garcia, F.D., "Beneath the Bonnet: A Breakdown of Diagnostic Security," in *Computer Security* (Lopez, J., Zhou, J., and Soriano, M., eds.), (Cham: Springer International Publishing, 2018), 305-324.
44. Nakamoto, Y., Yabuuchi, K., and Osaki, T., "Virtual Software Execution Environments for Distributed Embedded Control Systems," in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, Newport Beach, CA, 288-296, 2011.
45. Lind, K. and Heldal, R., "On the Relationship between Functional Size and Software Code Size," in *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*, Cape Town, South Africa, 47-52, 2010.
46. Brickell, E.F., Hall, C.D., Cihula, J.F., and Uhlig, R., "Method of improving computer security through sandboxing," U.S. Patent 7,908,653, March 15, 2011.
47. Sun, B., Chen, X., Xu, C., and Joseph, H., "Detecting java sandbox escaping attacks based on java bytecode instrumentation and java method hooking," U.S. Patent 9,223,964, December 29, 2015.

48. McCorkendale, B. and Ferrie, P., "Using a hypervisor to provide computer security," U.S. Patent 7,996,836, August 9, 2011.
49. Sivakumar, P., Sandhya Devi, R.S., Neeraja Lakshmi, A., VinothKumar, B. et al., "Automotive Grade Linux Software Architecture for Automotive Infotainment System," in *2020 International Conference on Inventive Computation Technologies (ICICT)*, Coimbatore, India, 391-395, 2020.
50. Automotive Grade Linux, "Automotive Grade Linux," 2019, https://www.automotivelinux.org/.
51. Gerg, I., "An Overview and Example of the Buffer-Overflow Exploit," *IAnewsletter* 7, no. 4 (2005): 16-21.
52. Kaspersky ICS CERT, "Threat Landscape for Industrial Automation Systems," Tech. Rep. 1.0, Kaspersky, 2019.
53. Ruff, M., "Evolution of Local Interconnect Network (LIN) Solutions," in *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall* (IEEE Cat. N0.03CH37484), Orlando, FL, vol. 5, 3382-3389, October 2003.
54. Lin, I.-C., and Liao, T.-C., "A Survey of Blockchain Security Issues and Challenges," *IJ Network Security* 19, no. 5 (2017): 653-659.
55. Dougherty, C., Sayre, K., Seacord, R., Svoboda, D. et al., "Secure Design Patterns," Tech. Rep. CMU/SEI-2009-TR-010, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2009.
56. Martin, H., Ma, Z., Schmittner, C., Winkler, B. et al., "Combined Automotive Safety and Security Pattern Engineering Approach," *Reliability Engineering and System Safety* 198 (2020): 106773.
57. Cheng, B.H.C., Doherty, B., Polanco, N., and Pasco, M., "Security Patterns for Automotive Systems," in *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019*, Munich, Germany, September 15-20, 2019 (Burgueño, L., Pretschner, A., Voss, S., Chaudron, M. et al., eds.), 54-63, IEEE, 2019.
58. Cheng, B.H.C., Doherty, B., Polanco, N., and Pasco, M., "Security Patterns for Connected and Automated Automotive Systems†," *Journal of Automotive Software Engineering* 1 (2020): 51-77.