Received February 20, 2019, accepted April 5, 2019, date of publication April 12, 2019, date of current version April 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2910865

T-Box: A Forensics-Enabled Trusted Automotive Data Recording Method

SEUNGHO LEE[1], WONSUK CHOI[1], HYO JIN JO[2], AND DONG HOON LEE[1], (Member, IEEE)
[1]Graduate School of Information Security, Korea University, Seoul 02841, South Korea
[2]School of Software, Hallym University, Chuncheon 24252, South Korea

Corresponding author: Dong Hoon Lee (donghlee@korea.ac.kr)

This work was supported in part by the Korea Ministry of Land, Infrastructure and Transport, and in part by the Korea Agency for Infrastructure Technology Advancement under Project 18TLRP-B117133-03.

ABSTRACT Modern vehicles are equipped with numerous electric control units which exchange vehicular status data, providing drivers with convenience, efficiency, and safety. In addition, the autonomous vehicles adopt various sensors that produce high volumes of high-speed data to process and assess internal and external situations. This data is particularly useful to automotive service providers such as car insurers, rental companies, and manufacturers. One way to understand how this data is used is to imagine the scenario in which an automobile insurer would provide a discount to a customer with an accident-free or near accident-free driving record. However, it is still possible that a less than the honest customer could manipulate their driving data in order to receive premium insurance services at preferential rates. To prevent this and similar scenarios, it is then critical to ensure that all data generated in a vehicle upholds integrity, continuity, and non-repudiation. Unfortunately, no such trustworthy data recording system of this caliber exists in any manufactured vehicle to date. This paper attempts to respond to this need, and we present a reliable automotive data recording system that satisfies these requirements and detects malicious manipulations from data deletion, replacement, replaying, and truncation. The proposed method additionally satisfies forward integrity of message authentication keys and is designed to utilize recorded data as automotive forensic evidence. Finally, the evaluation results demonstrate that our system can manage bandwidths of up to 64 MB/s.

INDEX TERMS Forward integrity, digital forensics, audit trail, event data recorder, ARM TrustZone.

I. INTRODUCTION

Vehicles are becoming increasingly complex electronic systems. A modern vehicle is equipped with numerous electronic control units (ECUs) that compose a versatile system known as an advanced driver assistance system (ADAS). In fact, luxury brand vehicles are reported to have more than 100 ECUs in order to provide customers with more effective benefits while driving [1]. ECUs work in concert to construct an in-vehicle network (IVN) that exchanges vehicular data, which means that modern vehicles generate enormous amounts of data within one single vehicle. This trend is also present in autonomous vehicles, which are equipped with a multitude of sensors.

Since vehicular data contains functional data from ECUs and environmental data from sensors, a wealth of meaningful information can be extracted. In many automotive applications, the extracted data is analyzed for specific purposes. For example, car insurers may wish to identify the driving habits of their customers and offer discounts to drivers who are proven to be safe drivers. These companies can identify driving habits through an analysis of automotive data stored in external recording devices [2]. Similarly, this data analysis could even enable car rental companies to monitor all unfortunate incidents involving their vehicles, even if there is no visible sign of impact. Thus, the high utility of automotive data for a wide variety of applications is evident.

Despite this great potential, many IVNs have been designed without adequate consideration for security. In recent years, attention to vehicular cyber attacks has risen largely in part due to the relatively high and feasible threat of lethal accidents resultant of such attacks. Miller and Valasek demonstrate cyber attacks on an unaltered vehicle [3] by sending CAN messages through a cellular network, thus

The associate editor coordinating the review of this manuscript and approving it for publication was Amit Singh.

exploiting the vulnerabilities of a telematics device which was connected to an in-vehicle CAN network. From this research, it is reasonable to conclude that cyber attacks do have the potential to be lethal; moreover, the attacker can stage these attacks to appear as innocently occurring accidents [4].

As automotive data is used in various areas and is becoming a plane for malicious attacks, there must be a method to ensure the trustworthiness of the data. This would, in turn, translate into the satisfaction of security requirements spanning integrity, continuity, and non-repudiation. The integrity check step would prevent unauthorized users from being able to alter data undetected. Next, continuity would provide a chronological order of data that would prevent attacks that might arbitrarily delete, replace, replay, or truncate data. Finally, the non-repudiation step would confirm the origin of the automotive data. These properties consequently would guarantee a higher level of reliability to service providers. In addition, automotive data satisfying these properties could then be utilized as digital evidence in automotive forensics.

We propose an automotive data recording system precisely of this nature that satisfies the above security requirements, coined the Trusted Box (T-Box for short). T-Box operates on a public key infrastructure to support integrity, continuity, and non-repudiation. Also, T-Box uses a message authentication code (MAC) and a digital signature together. The MAC is generated by a two-dimensional hash chain to ensure integrity and continuity of data entry, and this hash chain supports forward integrity of keys, making it advantageous in that it is unable to manipulate previously recorded data even if the current secret key becomes exposed. A digital signature providing the non-repudiation requirement is based on a cryptography system, whose computation speed is slower than that of symmetric-key cryptography. Thus, T-Box generates a digital signature on a block basis. T-Box is capable of recording high-speed automotive data up to 64MB/s by using a three-indexed ring buffer, consuming the minimum quantity of memory. The recorded automotive data is given a particular index by a monotonic counter and timestamp to guard against data deletion, replacement, or replay, and it also includes a special flag to detect data truncation. In addition, data recorded in T-Box is configured such that initial records are managed via a chain of custody by an investigator using Shamir's Secret Sharing Scheme. This is beneficial because it ensures the data is always dependable as dependable legal evidence when needed.

To provide the required security features, securing cryptographic operations are critical. The proposed T-Box operates in the TEE provided by ARM TrustZone and hence provides isolated execution of cryptographic operations to generate keys, MAC, and signatures without key exposure. In fact, ARM has presented a road map for the entire automotive industry [5], and Audi has also announced that they will install ARM-based automotive processors with TEE in their next infotainment system model [6]. Owing to these industry developments, T-Box adopts ARM TrustZone to prevent exposure of sensitive information.

Our main contributions are as follows:

- T-Box is a trusted automotive data recording system that ensures data integrity, continuity, and non-repudiation.
- The recorded data in T-Box can employ as forensic digital evidence.
- T-Box uses a formalized method that consumes minimal memory for recording real-time data generated in a vehicle.
- We provide performance analysis results and development methodology by using an emulator and a real target board.

The remainder of this paper is organized as follows: Section II discusses the proposed T-Box system in the context of related works. Section III provides necessary background information on IVN and TEE. Section IV presents a system model and threat model. Section V details a trustworthy data-recording method for vehicles. The performance results on a real target board and formulas for optimization can be found in Section VI. Section VII discusses limitations and future works regarding T-Box. Finally, a conclusion is given in Section VIII.

II. RELATED WORKS

A number of studies have detected compromised data by an attacker or a malicious user. This technical domain is referred to as an audit trail or audit log in [7]. The audit trail is a chronological record that can reconstruct events generated in a system. In the early predecessors to the audit trail, a message authentication code (MAC) was used to ensure log integrity. However, if a recording system uses only a MAC on each log entry, it is not possible to detect attacks that include data deletion, replacement, replay, or truncation. To solve this problem, some researchers adopted a hash chain. Kelsey et al. propose protocols to verify that a given digital image was taken by a specific camera and use digital signatures to ensure authenticity and chained hashes to ensure completeness in [8]. Schneier et al. present a signature format which embeds auditing information within the block and define specified fields in the signature packet for preventing or increasing the difficulty of attacks in [9]. Bellare et al. define the forward integrity among security properties for secure audit logs and propose forward-secure MACs by using a hash chain with an increasing index number in [10], [11].

When a hash chain is used to generate a series of MAC keys, the forward integrity of the keys is satisfied, but the verifier also requires a key, which would then present the issue of the key being identical to the symmetric key used by the log generator. One method to resolve key management issues was to transmit generated logs to a trusted machine. When a verifier needs to verify logs, they would send logs to a trusted machine and request verification in [12]. Schneier et al. show how to build a tamper-proof audit log in [13] and improve their previous works for low-bandwidth environments in [14].

After public key cryptography replaced symmetric-key cryptography for the audit trail, various studies emerged that reduced computation overhead and length of signatures. Holt extends Schneier and Kelsey's previous works to a number of applications using a public key in [15]. Ma et al. suggest the forward-secure sequential aggregate authentication schemes based on bilinear maps to reduce the overhead of communication on wireless sensor networks in [16], then they show practical signature schemes in [17] and propose new secure logging schemes in [18]. Yavuz et al. propose a new signature scheme that is referred to as a hash-based sequential aggregate and forward-secure signature for real-time authentication by using a hash operation on resource-constrained wireless sensor networks in [19], [20]. Although short signature schemes are more efficient for reducing communication resources, we do not consider them because these schemes are not widely found in the industry yet.

Recently even, some studies have proposed a method that does not depend on a trusted machine, but rather uses a hardware-based, trusted execution environment. Chong et al. implement Schneier and Kelsey's secure audit logging protocol by using tamper-resistant hardware to support running of the protocol offline as well as online in [21]. Sinha et al. additionally present a secure logging infrastructure which is capable of detecting tampered logs by using a trusted hardware module (TPM) in [22]. And finally, Karande et al. introduce a new logging system called SGX-Log by using Intel-SGX in a secure container called an Enclave in [23]. In this paper, we adopt ARM TrustZone instead of Intel-SGX to establish a hardware-based execution environment that is isolated and trusted, as ARM-based system-on-chips are rapidly distributing to all domains including IoT, mobile, and automotive.

The previous studies on audit trails are under active research in the server environment. Unfortunately, there is a lack of scholarship on data recording and logs in the automotive environment, despite the rapidly growing state of complex automotive systems. In tandem with this, global automakers, sensor manufacturers, and software companies are also focusing on autonomous vehicle development. As autonomous vehicles are equipped with numerous sensors, they produce massive volumes of high-speed data. Automotive data generated in a vehicle has a major impact on the car accident analysis. Generally, vehicles are equipped with an Event Data Recorder (EDR) that acts as a black box system. Similarly, the IEEE announced IEEE-1616-2010 in 2004, which is a motor vehicle EDR (MVEDR) standard [24]. MVEDR stores CAN packets in an IVN at the time of a car accident. According to a study by the National Highway Traffic Safety Administration (NHTSA) in the United States, the EDR only has a minimal data record that can be investigated for an accident [25]. It means that service providers cannot use the recorded data in MVEDR to provide customized services in real time. Furthermore, MVEDR is neither mandatory, nor does it support various data formats. Regarding MVEDR, Mansor et al. suggest acquiring data after authentication through a personal mobile device when accessing data recorded in an EDR [26]. Patsakis et al. alternatively propose a technique to protect data by using a time-lock puzzle so that malicious user cannot acquire EDR data during a particular time [27]. Unfortunately, these studies focus entirely on protecting data recorded in the EDR, and they do not consider data manipulation or digital forensics. Since self-driving cars are closer than ever to becoming a part of daily public life, automotive data must also be safely stored to have legal credibility in the event of an accident [28], [29]. To overcome these limitations, our new automotive data recording system is based on a trusted hardware-based execution environment.

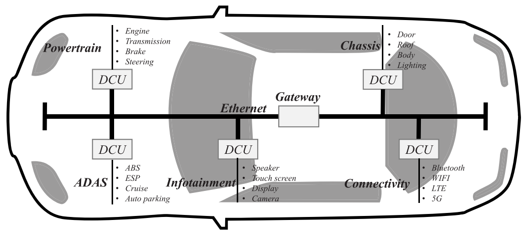


FIGURE 1. Ethernet backbone architecture.

III. BACKGROUND

A. IN-VEHICLE NETWORK

As the number of valuable functions provided in a vehicle increases, amounts of data exchanged in an IVN have also been increasing proportionally. Owing to this, in 1994, the Society for Automotive Engineers (SAE) categorized an automotive communication protocol. According to this categorization, the physical network environment depends on a transmission rate of ECU. Class A supports data at low speeds of less than 10Kb/s, such as door sensors, seat position, lights, and wiper data. Class B supports speeds from 10Kb/s up to 125Kb/s faster than Class A speeds. Class C supports speeds from 125Kb/s up to 1Mb/s for fast processing without loss of important data on functions such as the engine, suspension, and braking. Class D supports speeds over 1Mb/s to process large data, such as multimedia data. Based on this classification, various network protocols have been developed, such as LIN, CAN, FlexRay, and MOST [30], [31].

T-Box is designed with an ARM TrustZone-based high-performance processor to record high-speed, real-time data generated in an IVN. In recent years, a number of ongoing studies have already suggested utilizing an Ethernet backbone architecture which provides high-speed bandwidth between heterogeneous networks by constructing a backbone in the in-vehicle network as shown in FIGURE 1. All separated networks exchange data through a domain control unit (DCU) which is connected to the Ethernet backbone network. ECUs associated with a driveline, such as the engine, transmission, brakes, and steering are connected to the powertrain DCU. An ADAS DCU is connected to modules that are responsible for driver convenience and safety, such as the Anti-lock Braking System (ABS), Electronic Stability Program (ESP),

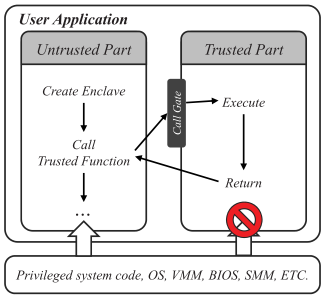


FIGURE 2. **The call flow of a user application using Intel-SGX.**

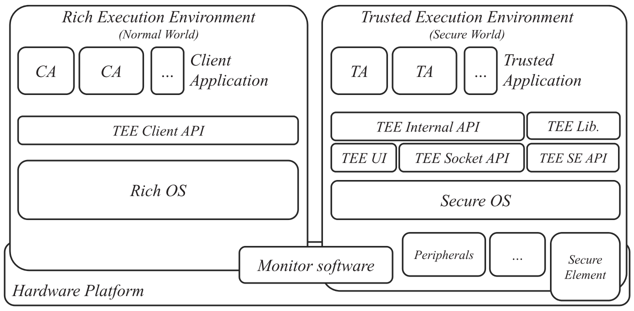


FIGURE 3. **The architecture of ARM TrustZone.**

cruise control, and auto parking. The connectivity DCU is connected to a wireless network such as Bluetooth, Wi-Fi, and the cellular modem. The Ethernet backbone network requires a minimum bandwidth of 100 Mb/s to handle these heterogeneous networks [32]. A gateway is positioned on the Ethernet backbone network to transfer packets between heterogeneous networks. To address high-speed bandwidth, the gateway adopts high-performance processors and, similarly, the infotainment system also requires high-performance processors to handle large multimedia data and provide assistance to the driver. Thus, T-Box can be located in either the gateway or infotainment system to monitor the high-speed Ethernet network.

B. TRUSTED EXECUTION ENVIRONMENT (TEE)

The Trusted Execution Environment (TEE) is based on hardware-based isolation technologies, which are advantageous because a target process can be operated away from other processes in completely separate hardware resources such as the register, cache, or main memory.

Intel-SGX and ARM TrustZone are representative commercial solutions implementing TEE. Intel-SGX has utilized a TEE called Enclave since a Skylake microarchitecture in 2015, and it is primarily used to provide secure computing in a cloud environment. ARM TrustZone was introduced as product ARM1176JZF-S using ARMv6 Architecture in 2003, and it provides a TEE called Secure World that primarily supports isolated execution environments on mobile devices. Intel-SGX and ARM TrustZone are operated via different mechanisms.

An application on Intel-SGX is divided into a non-trusted and trusted part, as shown in FIGURE 2. Specifically, the trusted part operates on the Enclave TEE. When a corresponding application operates, the non-trusted part creates an Enclave, thus loading code and data of the trusted part onto an isolated memory space called Enclave Page Cache (EPC). The code running in Enclave can be called up through a predefined call gate interfaces. Intel-SGX includes the basic features of a Trusted Platform Module (TPM) standardized by the Trusted Computing Group (TCG) [33]. Intel-SGX provides a MRENCLAVE with a hash value of SHA265, which verifies the integrity of the loading binaries against "Root of trust for measurement" in TPM and provides an EREPORT command for an attestation function against "Root of Trust for Reporting" in TPM. It also provides sealing and unsealing functions to encrypt or decrypt sensitive data in Enclave [34]–[37]. The major notable difference between Intel-SGX and ARM TrustZone is that a user application can immediately switch from the non-trusted part to the trusted part while in user mode without additional costly operations. In contrast, ARM TrustZone has extra software overhead to switch from non-trusted code to trusted code.

A model of ARM TrustZone architecture is shown in FIGURE 3. The TEE provided by ARM TrustZone requires a separate operating system called a Secure OS, and this environment is called a secure world. Much general-purpose software such as Linux, Android, and Windows all operate on the opposite side of TEE in systems called a rich execution environment (REE), or a normal world. Applications are classified into client applications (CA) or trusted applications (TA) depending on in which environment they are running. CAs are applications running in the normal world, and TAs are applications running in the secure world. Distinct from Intel-SGX, ARM TrustZone can execute various TAs to support multiple security services simultaneously. It is important to note that CAs and TAs are entirely different applications. CAs can request secure service to TAs through standard interfaces. When a CA calls a TA, the execution environment is converted from REE to TEE via monitor software. The transition from REE to TEE, or conversely from TEE to REE, is referred to as world-switching. The monitor software accesses a secure configuration register (SCR) to set a non-secure (NS) bit to 0 (zero) during the world-switching, and then enters a Secure OS. The NS bit of the SCR is connected to an Advanced eXtensible Interface (AXI) which is ARM's high-speed bus interface. Therefore, setting the NS bit in an application processor supporting ARM TrustZone affects all other intellectual property (IP) connected to the AXI. Some of the typical IPs include TrustZone Address Space Controller (TZASC), TrustZone Protection Controller (TZPC), and TrustZone Memory Adapter (TZMA). TZASC is a controller that configures a particular region of a DRAM to be accessible in a secure world, and TZPC is a controller that set access permission to peripheral devices connected through a memory mapped I/O method. Finally, TZMA configures a part of an SRAM to be used in a secure world before a DRAM is initialized at the system boot phase. In addition, a cache controller and memory management unit (MMU) are also related to the NS bit [38], [39].

ARM provides the relevant security IPs and related software compatible with mobile security standards which are set by GlobalPlatform. GlobalPlatform is a non-profit industry association that facilitates the reliable and secure management of digital services and devices, standing as the leaders of security standards related to TEE, such as TEE Client API, TEE Internal API, TEE Secure Element API, TEE Socket API, and TEE User Interface. TEE Client API provides interfaces used by CA. It supports an interface to establish secure communication channels with and to request security services from TAs. TAs also use the TEE Internal API to provide interfaces to call internal cryptographic functions or to call another TA and request alternate security services. The next item used by TAs is TEE Socket API, which provides interfaces to connect remote servers located externally to devices through UDP or TCP protocol. Continuing, TEE User Interface provides a secure user interface for TAs and, finally, TEE SE API is a standard interface that facilitates communication with an externally separated chip called a secure element (SE) that includes a hardware tamper-resistance function [40], [41]. The key used in our proposed T-Box method can be encrypted with a device's unique key and stored into the REE filesystem, or it can be managed by the Secure Element. T-Box encrypts keys used for signing, or for generating a MAC with a device's unique key, and stores it into the REE filesystem. In a different way, the keys can be managed by an SE.

Perhaps more importantly, developers can apply a secure boot scheme with ARM TrustZone. The secure boot is a technology that provides reliability to an entire boot sequence from the initial boot phase of a device to creation of a user-ready environment. More specifically, secure boot technology is a process of verifying the signature of the next running binaries. When the device's power is on, an initial program in a read-only memory (ROM) verifies the next binary located in a non-volatile memory, such as a flash memory. If the next binary is correct, the program passes execution control to that binary. With this technique, an attacker cannot run any manipulated binaries on a device in the middle of the secure boot period. Under the secure boot technology, programs running on T-Box can neither be modified nor replaced by an attacker. Furthermore, our T-Box method based on ARM TrustZone is able to safeguard sensitive keys without risk of exposure. One of these sensitive keys is an initial key to generate the MAC of recorded data, and it is distributed to other participants through Shamir's Secret Sharing Scheme to make specific constraints. Details are described in Subsection V-D3.

C. SHAMIR's SECRET SHARING

One of the key management schemes is a secret sharing scheme. The function of such a scheme is to distribute an original secret to all participants. The distributed secret given to each participant, though, is only a piece of the original secret. Thus, it is appropriately called a "shared secret." In particular, individual participants cannot restore an original secret by using their own shared secret. In fact, the original secret can only be restored through the combination of a minimum number of shared secrets. Shamir's Secret Sharing Scheme (SSS) is based on polynomials. To illustrate the algorithm, imagine that when a participant receives two points, he/she can find the correct linear first-degree polynomial which meets two points. Three points are a quadratic polynomial, and four points are third-degree polynomial. In practice, a system that divides an original secret to all n participants and restores the original secret by using k shared secrets, we need a k-1 degree polynomial. The SSS distributes n coordinates generated by the k-1 degree polynomial to each participant. The k-1 degree polynomial can be recovered when some participants collectively possess a number of shared secrets more than the number of k. The constant value of this recovered polynomial is the original secret. Thus, SSS is also called a (k,n) threshold scheme [42]. T-Box applies SSS to recorded data as digital forensic evidence. The detailed methods are explained in Subsection V-D3.

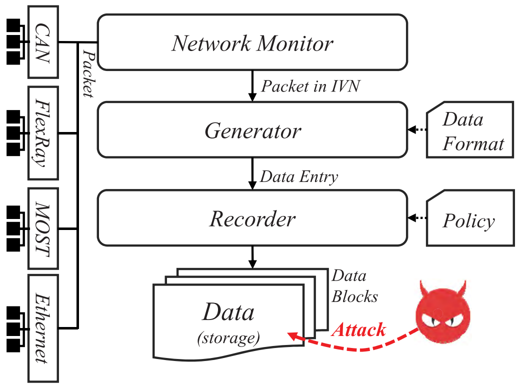


FIGURE 4. The overview of an automotive data recording system.

IV. SYSTEM MODEL

A. SYSTEM OVERVIEW

To restate, the goal of T-Box is to ensure integrity, continuity, and non-repudiation of data generated in a vehicle. In FIGURE 4, an automotive data recording system consists of a network monitor, generator, and recorder. The network monitor transfers packets from a source network to a destination network in an IVN. It is able to monitor whole packets on a backbone network.

In a real vehicle, it does not have a network monitor, so we assume that the gateway could be the network monitor. The generator reconstructs data provided by the network monitor according to data format and delivers it to the recorder. The recorder stores data entries received from the generator according to a specified policy. In this context, a policy is a predefined configuration intended to categorize and manage data generated in a vehicle, such as by key generation method, time sequence, data ordering, or cryptography type. The recorder stores an individual data entry into a block and these data blocks are stored in a local persistent storage or an external storage device connected to standard interfaces, such as USB or onboard diagnostics (OBD) ports. Alternatively, data blocks can be transmitted to a remote server of service providers to facilitate and expedite services in real time.

B. THREAT MODEL

In the system shown in FIGURE 4, insider attackers and remote attackers are able to manipulate stored data. Insider attackers are parties who have the authority to directly access stored data–such as a car owner, or Original Equipment Manufacturer (OEM)–and maliciously attempt to manipulate data by abusing their privileged access. For example, in the case of a car accident, a driver could potentially delete stored data or overwrite it with normal data in attempts to cover up a driving misstep. Also, an OEM can modify or replace stored data to avoid exposing manufacturing defects. On the other hand, remote attackers are parties who want to remotely manipulate data stored in a vehicle. For example, a remote service shop can connect to a vehicle without the driver's knowledge or consent, gather vehicular status data, and then delete a connection log to conceal evidence of the remote log-in. Presently, data manipulated by such attackers cannot be used as forensic evidence in critical situations. In response to this, we designed T-Box to detect when such insider attackers or remote attackers maliciously manipulate stored data, and to support the chain of custody over all of the data through a verification process.

According to the assumptions for this research, the network monitor, generator, and recorder are loaded securely through a secure boot technology at system boot time. Therefore, attackers cannot arbitrarily change those binaries. In addition, a data entry is normally generated by a verified generator that is not maliciously modified. In this paper, attackers can only manipulate stored data in storage. To inject packets and modify them via direct connection to an IVN is out of the scope of this paper.

V. DESIGN

A. NOTATION

In the remainder of this paper, we use the following notation described in TABLE 1.

B. REQUIREMENTS

As shown in the system model and the threat model, T-Box must satisfy the following requirements.

- R-1. All recorded data must satisfy integrity, continuity, and non-repudiation. Thus, if the data is manipulated, it must be detected.
- R-2. All data generated in an IVN must be completely stored using minimal memory in real time without loss.
- R-3. All recorded data must be utilized as forensic digital evidence.

TABLE 1. The notation of this paper.

Notation	Description
x	x is a set of whole parties. $x \in \{Owner, OEM, Insurer, Rental, Investigator\}$
y	y is a set of parties except for an investigator. $y \in \{Owner, OEM, Insurer, Rental\}$
PKE_x	public key encryption with x's public key
$pubKey_x$	public key in x's certificate
Key_{init}	Initial seed key for generating a MAC key
$ssKey_x$	Shared secret for party x
$W(x)$	Weight of x
NP	Number of parties
K_{th}	Threshold level of secret sharing
Key_{list}	MAC key list for data entries
bID	Block index of recorded data
$bKey$	Initial block key for generating Key_{list}
eID	Data entry index in a block
DE	Data entry
KDF	Key derivation function
$Sign$	Sign operation
$Hash$	Hash operation
$HMAC$	HMAC operation with SHA256
L_{DE}	Size of data-entry in a block
T_{WS}	Time for world switching
T_{KC}	Time for generating MAC_{list} for a block
T_{MAC}	Time for MAC generation
T_{SIGN}	Time for signing
F_{GEN}	The frequency of data generation
T_{GEN}	Time for generating data entry
T_{Block}	Time for processing a block
N_{DE}	Number of data-entries in a block
N_{RB}	Number of buffers in a circular ring buffer

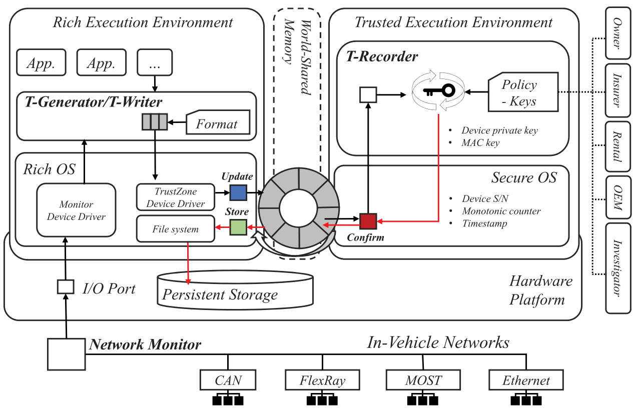


FIGURE 5. The architecture of T-Box.

C. ARCHITECTURE

FIGURE 5 illustrates the architecture of T-Box. T-Box generates a MAC by using a symmetric key and generates a signature by using a private key; both keys are located in TEE. T-Box consists of three components: T-Generator, T-Recorder, and T-Writer. T-Generator operates in REE and is responsible for generating a data entry with a packet sent by a network monitor. When creating a data entry, T-Generator reconstructs the packet according to the predefined data format. The data format has additional fields such as a block index, entry index, counter value, timestamp, and packet data.

T-Generator and T-Recorder use shared memory to share data entries. Shared memory is a physically contiguous memory. Both of these components need to map the same physical memory address to a different virtual memory address space of each process. This memory mapping operation is provided by the TEE Client API. T-Box uses a circular ring buffer structure for consuming minimal memory. After T-Generator creates a data entry according to the data format and writes a data entry to a ring buffer, T-Generator requests T-Recorder to generate a MAC.

Following this, T-Recorder adds additional metadata in data entry. The metadata is a set of information that consists of information accessible to TEE, such as monotonic counter and timestamp. Then, T-Recorder creates a MAC corresponding to the data entry. When the number of data entries becomes equal to the block size, the T-Recorder generates a signature on a block-by-block basis by using a private key of T-Box. As soon as T-Recorder generates a MAC or a signature, T-Writer stores recorded data entries and a signature to an REE file system and removes it from the ring buffer.

During the T-Box initializing phase, a public key pair of a device is securely generated in TEE. The generated private key is securely stored in TEE, but the public key is sent to a Certificate Authority (CA) to be issued a certificate. In addition, T-Recorder stores the certificates for each party, including the car owner, OEM, insurer, renter, or investigator. There are two ways to connect T-Box and the CA. First, for the recent V2X technology, there is a tendency to incorporate a modem chip in vehicles [1], [6], which T-Box can use to communicate with the CA. Second, T-Box can acquire user information through wireless networks such as Bluetooth during the user registration process and communicate with a remote CA by using the modem function of a smartphone.

An initial MAC key (Key_{init}) is securely generated in TEE, and it is distributed by T-Recorder to each party using Shamir's Secret Sharing, as described in Section III-C. When T-Box distributes shared secrets to each party, T-Box also encrypts these shared secrets with a public key in a certificate specific to each party. The reason for distributing Key_{init} with SSS is to make a specific case in which it is only possible to restore the Key_{init} when k-parties are in agreement with all n-parties. It prevents attacks by insider and remote attackers who are described in Subsection IV-B. The details of each module are described in the next section.

D. PREPROCESSING

1) THREE-INDEXED CIRCULAR RING BUFFER

T-Generator and T-Writer share data entries with T-Recorder through a global shared memory. We use a circular ring buffer for a global shared memory to consume minimal memory. The circular ring buffer used in T-Box has three indexes, as shown in FIGURE 6: updated, confirmed, and stored.

The updated-index is used by T-Generator to update a data entry to the position of updated-index, which it then moves to the right. The confirmed-index is used by T-Recorder to generate a MAC or a signature if there is an updated data entry to the right of the confirmed-index, after which it moves the confirmed-index to the right. Finally, T-Writer stores a data entry into persistent storage when the confirmed data entry exists to the right of the stored-index. Following this, it deletes a corresponding data entry from the circular ring buffer and moves the stored-index to the right. In this way, T-Generator, T-Recorder, and T-Writer sequentially process data entries through one circular ring buffer. This three-indexed circular ring buffer satisfies the second system requirement (R-2).

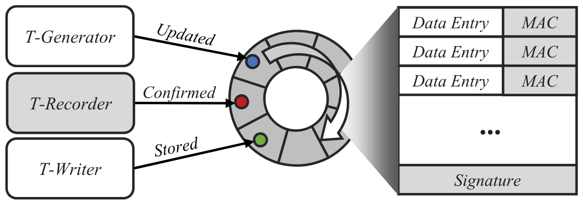


FIGURE 6. The 3-indexed circular ring buffer used in T-Box.

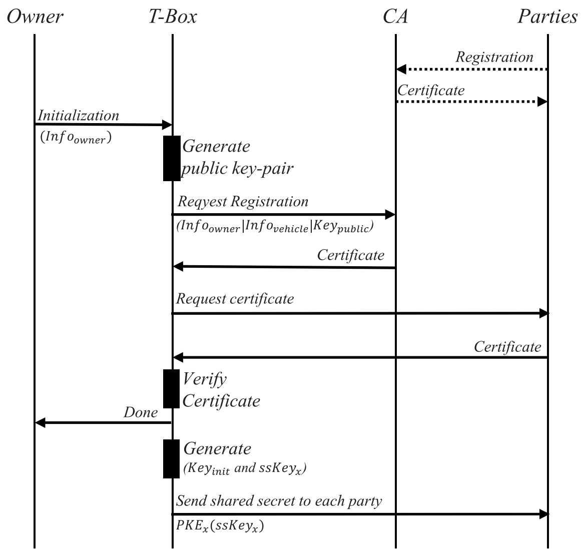


FIGURE 7. The sequence diagram of T-Box initialization phase.

2) PUBLIC KEY INFRASTRUCTURE (PKI)

Automotive data recorded by T-Box would be useful for a number of agents, including insurance companies that provide customized premium services, car rental shops that manage the condition of rental cars, and car manufacturers that provide service on their vehicles. We define these agencies and companies as parties.

T-Box operates based on public key infrastructure (PKI), and, as shown in FIGURE 7, when an owner registers his or her information with T-Box during an initialization phase, a public key pair is generated. T-Box securely keeps a private key in TEE and sends a public key with additional information to a CA to request a certificate for the T-Box. T-Box also downloads and maintains certificates for all parties to TEE. After T-Box generates Key_{init}, T-Box encrypts $ssKey_x$ which is produced from SSS with a public key in each party's certificate and distributed to each. The detailed sequence of the $ssKey_x$ distribution and a subsequent reconstructing process of Key_{init} is described in the next section.

3) INITIAL MAC KEY AND SECRET SHARING

T-Recorder produces an initial MAC key (Key_{init}) at first. Key_{init} is a seed key for generating a number of MAC keys through a hash key chain. The purpose of generating a MAC key list is to derive all of the keys for all data entries in blocks. This reduces hash operation preparation overhead in TEE by deriving the data simultaneously. As described in Subsection III-C, T-Box transmits $ssKey_x$ derived from Key_{init} to each party through Shamir's Secret Sharing. The method for sharing the shared secret ($ssKey_x$) for each party is determined by the corresponding T-Recorder policy.

The policy is akin to a set of rules for distributing Key_{init} to each party. It defines two numbers. The first number is variable N to make a decision about how many shared secrets are generated, and the second number is variable K to configure the threshold number for how many shared secrets should be combined when some parties want to restore Key_{init}. T-Box derives $ssKey_x$ from Key_{init} according to this policy and encrypts the derived $ssKey_x$ with a public key in a certificate. Subsequent to sending the encrypted $ssKey_x$, T-Box generates a hash value of Key_{init} to employ the initial key of the hash chain. As a result, T-Box does not have the Key_{init} anymore. This step protects the Key_{init} through a hash function, satisfying forward integrity. Even if attackers were to obtain a MAC key through side-channel attacks utilizing cache memory, they would not be able to manipulate previously recorded data.

T-Box applies a hash chain to prevent insider and remote attackers from arbitrarily manipulating stored data. This, in turn, satisfies the first requirement (R-1). In addition, when shared secrets of Key_{init} are distributed to all parties, the investigator is given significant responsibility. To execute this approach, the policy must be configured to meet the following constraints.

- $ssKey_x$ are securely delivered to all parties.
- Negotiated parties cannot restore Key_{init} by using their own $ssKey_x$ except for the investigator.
- A reconstruction process of Key_{init} must include an investigator.
- The investigator cannot reconstruct Key_{init} by oneself.
- The investigator requires at least two or more parties at the time of Key_{init} reconstructing.
- The investigator is responsible for managing related status in sequence when Key_{init} is reconstructed by requesting from other parties.

Our design is constructed such that recorded data by T-Box can be utilized as digital evidence for automotive forensics. Consequently, the role of an investigator becomes more important. An investigator is a trusted authority like a national crime scene investigator or national forensic service agency. Since T-Box generates a signature on a block-by-block basis, some data entries may not be included in a signed block. Therefore, remaining data entries must be verified through their MACs. When some parties want to verify the MACs of remaining data entries that are not included in a signed block, each party should send the investigator necessary information such as a $ssKey_x$, the hash value for all of the data, the vehicle ID, and signature. This ensures that the chain of custody for the requested data is maintained. The investigator records this information and reconstructs Key_{init} by using his or her own $ssKey_{investigator}$ and received $ssKey_x$. The reason for using this procedure is to consider the situation of a car accident, which we will explain next.

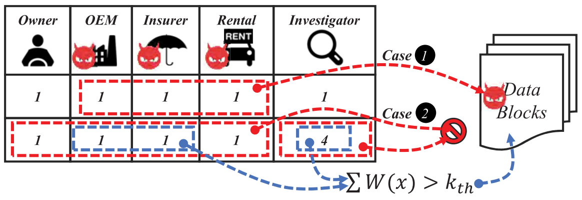


FIGURE 8. The attack by malicious parties with an agreement.

In a car crash situation, the remaining data entries contain critical information about why the accident happened. If a vehicle has been shut down through a normal procedure, we can be assured that recorded data was correctly generated in the vehicle by verifying a signature that would have been signed with the vehicle's private key. In contrast, the situation of a car crash may not completely generate a signature, which is caused by an abnormal shutdown. When some data entries are not included in the latest signed block, remaining data entries can be arbitrarily manipulated by malicious associated parties who can restore Key_{init} through a MAC operation. In this case, it is not possible to keep relevant information regarding a chain of custody after this point. In order to prevent any manipulation of the remaining data entries, we apply a weight-based secret sharing scheme. In the policy described above, restoring Key_{init} requires reaching an agreement between at least two parties except for the investigator. Of course, we can define the policy to reconstruct Key_{init} when all parties reach an agreement. However, the potential death of the car operator (driver) due to an accident should also be taken into consideration. Accordingly, we have set the policy to obtain consent for at least two parties, excluding the investigator. To achieve the above policy, parties such as car owners, OEMs, insurers, or rental companies are equal in the distribution of shared secrets, but the investigator has access to more. This is to prevent arbitrary parties from manipulating previously recorded data entries.

In FIGURE 8, Case-1 is (3,5) exhibits Shamir's Secret Sharing Scheme in which all parties have the same weight of shared secrets. If all parties have the same weight, an OEM, insurer, and rental company could restore Key_{init} without an agreement with the investigator. If there are many service providers, it means that maliciously negotiating parties having the same weight can alter data entries under a mutual agreement. To resolve this challenge, we increase the weight of the investigator's shared secret ($ssKey_{investigator}$) as shown in Case-2. In (K, N) secret sharing, the weight of $ssKey_{investigator}$ should be equal to sum of the weight of all other parties excluding the investigator. Thus, we set a variable N as 8 which is the weight of all parties, and we set a

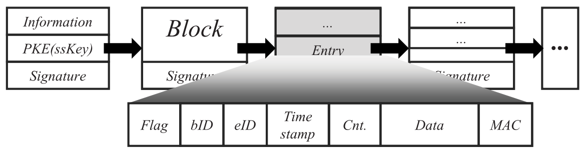


FIGURE 9. The structure of a data block in T-Box.

variable K as 6, which is the weight that the investigator can receive from two parties. Therefore, we use (6, 8) Shamir's Secret Sharing Scheme to satisfy the T-Box policy. The policy is explained below.

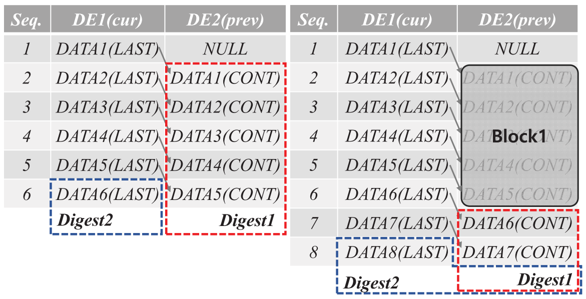
$$\begin{aligned} W(investigator) &= \sum W(y) \\ K_{th} &= W(investigator) + \sum W(y) \\ N &= \sum W(x) \end{aligned}$$

In this case, no party is able to restore the Key_{init} without the investigator's agreement. Also, an investigator is not able to restore the key alone without requests from other parties. To use the above (6, 8) secret sharing scheme, T-Box calculates eight coordinate values (i, j) using a randomly selected fifth-degree polynomial and transmits the corresponding coordinate values to the parties according to assigned weight. Since T-Box possesses all of the certificates for all of the parties, T-Box is also responsible for encrypting all $ssKey_x$ with public keys in certificates and transmitting them to each party. The encrypted $ssKey_x$ will then be decrypted with a private key that is stored by each party in respective designated locations. All of the procedures described thus far are handled during the T-Box initialization phase. The purpose of applying this scheme is to satisfy the previously discussed third requirement (R-3).

E. T-GENERATOR AND T-RECORDER

T-Generator is a process which operates during the system boot phase. It functions to establish a communication channel with the T-Recorder, operating in TEE, and requests shared memory be used as a circular ring buffer. Then, the T-Generator reconstructs packets delivered by a network monitor according to the defined format. A structure of data entry is shown in FIGURE 9.

A data block contains several data entries, including additional metadata: a flag of data type, a distinguishable block index (bID) and a data entry index (eID), a timestamp of the recording time when the data entry is generated, data for a packet received from a network monitor, a monotonic counter for ensuring continuity, a MAC to ensure integrity of data entries, and a signature to verify non-repudiation. The T-Recorder writes this metadata through special function registers in TEE. Note that the first block contains general information such as the T-Box serial number. The serial number included in the initial block is a vehicle identification number (VIN), which is a combination of 17 characters defined in ISO 3779. Each character can be expressed in 136 bits using



Seq.	DE1(cur)	DE2(prev)
1	DATA1(LAST)	NULL
2	DATA2(LAST)	DATA1(CONT)
3	DATA3(LAST)	DATA2(CONT)
4	DATA4(LAST)	DATA3(CONT)
5	DATA5(LAST)	DATA4(CONT)
6	DATA6(LAST)	DATA5(CONT)
	Digest2	Digest1

Seq.	DE1(cur)	DE2(prev)
1	DATA1(LAST)	NULL
2	DATA2(LAST)	DATA1(CONT)
3	DATA3(LAST)	DATA2(CONT)
4	DATA4(LAST)	DATA3(CONT) Block1
5	DATA5(LAST)	DATA4(CONT)
6	DATA6(LAST)	DATA5(CONT)
7	DATA7(LAST)	DATA6(CONT)
8	DATA8(LAST)	DATA7(CONT)
	Digest2	Digest1

FIGURE 10. The countermeasures for preventing a truncated tail data attack.

ASCII code. The initial block only contains the VIN, and all subsequent blocks contain only packet data in the in-vehicle network. If we separately analyze packets generated by one ECU, an efficient compression algorithm may be applied to reduce data because there exists the same data in a packet field. In this paper, we propose a TEE-based T-Box by selecting the specific parameters needed to record packets in real time and finding the maximum bandwidth it can record. Although efficient compression algorithms can reduce storage and network costs if the data size is reduced, we would be required to re-visit the parameters associated with compression algorithms, so this paper does not deal with these but instead offers a discussion about relevant points in Section VII.

When T-Generator sequentially generates data entries, it processes two data entries. One of these is a current data entry (DE1) and the other is a previously registered data entry (DE2). DE1 is a tail data entry (i.e., the latest data entry) and has a flag labeled "LAST", and DE2 is a previously registered data entry which was generated before DE1. Thus, DE2 has a flag "CONT" (continue), which signifies that a subsequently generated data should follow. When T-Recorder generates a MAC for each data entry, T-Writer stores it sequentially to the persistent storage. At this point, when a flag is changed from "LAST" to "CONT", the T-Writer overwrites any previously stored data entries with the flag "LAST" and updates these data entries to "CONT".

This is a countermeasure to prevent truncated tail data attacks. Simply put, a tail attack is when an attacker deletes the last stored data entry. If T-Box only adopts a hash chain without a means to distinguish flag values, it would not be able to detect these types of attacks. As such, T-Box overwrites each updated data entry with a new flag value and a new MAC. Thus, during the verification procedure, any last data entry set to "CONT" would indicate that the next data entry or entries have been truncated. T-Recorder has a digest value of Digest1, which accumulates data entries with the "CONT" flag. If the number of data entries is larger than the predefined block size, the digest value is signed with a private key by T-Box. For example, FIGURE 10 shows an instance when the block size is 5. When DATA6 (LAST) is delivered to the T-Recorder, the T-Recorder changes DATA5 flag to "CONT" and regenerates a MAC. When the number of data

Algorithm 1 T-Recorder

```
Require: $DE_{prev}, DE_{cur}$
Ensure: Signature or MAC
 1: if Request is MAC then
 2:     $idx \leftarrow monotonic\_counter$
 3:     $eID \equiv idx \pmod{block\_size}$
 4:     $bID = idx / block\_size$
 5:     if $eID$ is the first entry in a block then
 6:         /* Update initial block key */
 7:         $bKey \leftarrow Hash(bKey_{bID-1}|bID)$
 8:         for each key-index in $Key_{list}$ do
 9:             /* Update MAC key-list for each data entry */
10:             $Key_{list}$[key-index]$\leftarrow Hash(bKey|$key-index$)$
11:         end for
12:     end if
13:     $DE1 \leftarrow DE_{cur}$
14:     $DE2 \leftarrow DE_{prev}$
15:     /* Update flag and regenerate MAC /*
16:     if $DE2.flag$ is $LAST$ then
17:         /* Verifying MAC */
18:         $MAC_{temp} \leftarrow HMAC(Key_{prev}, DE2)$
19:         if $MAC_{temp} \neq DE2.mac$ then
20:             /* Error: Failed to verify MAC */
21:         end if
22:         $DE2.flag \leftarrow CONT$
23:         $DE2.mac \leftarrow HMAC(Key_{prev}, DE2)$
24:         /* Update digest for the signature of block */
25:         $Digest1 \leftarrow Hash(Digest1|DE2)$
26:     else
27:         /* Error: flag is not an expected value */
28:     end if
29:     $DE1.flag \leftarrow LAST$
30:     $DE1.eID \leftarrow eID$
31:     $DE1.bID \leftarrow bID$
32:     /* Generate a MAC using eID'th key in $Key_{list}$
33:     $DE1.mac \leftarrow HMAC(Key_{list}[eID], DE1)$
34:     $Key_{prev} \leftarrow Key_{list}[eID]$
35:     /* Update digest for signature*/
36:     $Digest2 \leftarrow Hash(Digest1|DE1)$
37: else if Request is Signature then
38:     /* Sign the digest and clear it */
39:     if block is full then
40:         $Signature \leftarrow Sign_{privateKey}(Digest1)$
41:     else if block is not full then
42:         $Signature \leftarrow Sign_{privateKey}(Digest2)$
43:     end if
44:     $Digest1 \leftarrow Null$
45:     $Digest2 \leftarrow Null$
46: end if
```

entries with the "CONT" flag becomes equal to the block size 5, a signature is generated by Digest1. After that, the T-Recorder generates a MAC of DATA6 (LAST). T-Writer overwrites DATA5 (LAST) with DATA5 (CONT) and stores

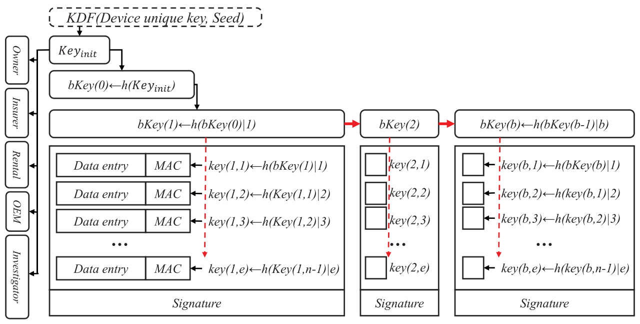

KDF(Device unique key, Seed)
Owner
Insurer
Rental
OEM
Investigator
Key_{init}
$bKey(0) \leftarrow h(Key_{init})$
$bKey(1) \leftarrow h(bKey(0)|1)$
$bKey(2)$
$bKey(b) \leftarrow h(bKey(b-1)|b)$
Data entry
MAC
$key(1,1) \leftarrow h(bKey(1)|1)$
$key(1,2) \leftarrow h(Key(1,1)|2)$
$key(1,3) \leftarrow h(Key(1,2)|3)$
$key(1,e) \leftarrow h(Key(1,n-1)|e)$
$key(2,1)$
$key(2,2)$
$key(2,3)$
$key(2,e)$
$key(b,1) \leftarrow h(bKey(b)|1)$
$key(b,2) \leftarrow h(key(b,1)|2)$
$key(b,3) \leftarrow h(key(b,2)|3)$
$key(b,e) \leftarrow h(key(b,n-1)|e)$
Signature

FIGURE 11. The structure of 2-dimensional hash key chain.

a signature. Finally, it stores DATA6 (LAST) and its MAC. If a normal shutdown has arisen while DATA8 (LAST) is stored in the persistent storage, T-Box generates a signature with Digest2, which is the accumulation of Digest1 and DATA8 (LAST).

In the case of an abnormal shutdown, such as a car accident, T-Box cannot generate a signature for the last block. In this case, each party can request the investigator to reconstruct key_{init} to generate a MAC key list. By using regenerated keys from key_{init}, parties are able to verify all data entries through a MAC and signature verification. The algorithm 1 is a pseudo-code of T-Recorder. When the first data entry is delivered, T-Recorder generates a MAC key list for one block by using an initial block key (in line 5). As described before, T-Recorder addresses two data entries to prevent tail attacks, one which is current and one which is a previously registered data entry (in line 13-36). If a block is full, T-Recorder generates a signature using Digest1 which are hashed from data entries labeled "CONT". If a block is not full–as is the case during a normal shutdown– T-Recorder uses Digest2 which processes the latest data entry labeled "LAST" (in line 40, 42).

1) TWO-DIMENSIONAL HASH KEY CHAIN

The Key_{init} which will have been distributed to all parties through Shamir's Secret Sharing Scheme is generated with a device's unique key and a seed value through a one-way hash function, and it is used for generating an initial block key (*bKey*). A two-dimensional hash key chain uses a *bKey* to derive the next initial block key that subsequently generates a MAC key list to be used for whole data entries in each block (shown in FIGURE 11). This concept has been introduced in related studies already. Sinha et al. use a two-dimensional hash key chain separated by epoch units in an enhancement protocol to work with limited disk space and verify an arbitrary subset of logs in [22]. Likewise, Karande et al. use a log block unit to efficiently access an arbitrary log and increase the security of a current syslog program through Intel-SGX in [23]. The authors ensure the integrity of logs by using a MAC that is generated by symmetric cryptography. Using these existing studies as a foundation, T-Box creates a signature on a block basis to support non-repudiation as one of the security requirements. Also, the two-dimensional hash key chain prevents exposure of an initial MAC key during side-channel attacks that feed off of the vulnerabilities of cache memory on ARM TrustZone [43], [44].

According to this scheme that generates a MAC and a signature, all parties can verify the signature of a data block by using a public key in the T-Box certificate. Even if they do not know key_{init}, they can provide customers with customized services by analyzing the recorded data, including the signature. Typically, to satisfy all security requirements, we could imagine creating signatures for all data entries. However, due to the larger computational overhead required for a public key cryptosystem, signing each data entry is unreasonable for a real-time recording system. As T-Box applies a MAC and a signature simultaneously, it can reduce the time overhead that would be lost generating data blocks. Streamlining the security requirements in this way means that T-Box can handle generated data in real-time while still satisfying the second requirement (R-2). Furthermore, to reduce key generation time overhead for all data entries, T-Box generates a key list instantaneously when the first data entry arrives in a block. Since the MAC keys are generated through a hash function including a specific index number (*eID*) for each entry, each data entry uses the defined MAC key generated by using the same index number. These characteristics ensure continuity requirements for security.

F. VERIFICATION OF THE RECORDED DATA

There are two methods to verify recorded data entries: simple verification and full verification.

1) SIMPLE VERIFICATION

The purpose of the simple verification method is to verify each signed block. T-Box generates a signature on each data block, such that service providers can quickly verify these data blocks using a public key contained in the T-Box certificate.

$$hash(Data\ block) = Verify_{pubKey}(signature)$$

A digital signature can check the integrity and non-repudiation of a message at the same time. Thus, service providers can provide suitable and rapid services to customers based on analyzed data blocks after initiating this simple verification process. In short, the simple verification method is identical to the signature verification procedure. Since T-Box signs all blocks with a private key, anyone can verify data blocks with the public key in the T-Box certificate.

2) FULL VERIFICATION

Alternatively, the purpose of a full verification method is to check whole data entries. Even if there are remaining data entries excluded from a particular signed data block, this method can still verify them. In effect, this method reverses the T-Recorder procedure. In contrast, full verification is a set of simple verification plus MAC verification procedures for each data entry. It provides a precise verification procedure that enables analysis of crucial situations, such as car accidents or vehicle defects. Since the latest data entry contains the most important information, the MACs of data entries excluded from a signed block must also go through the verification process.

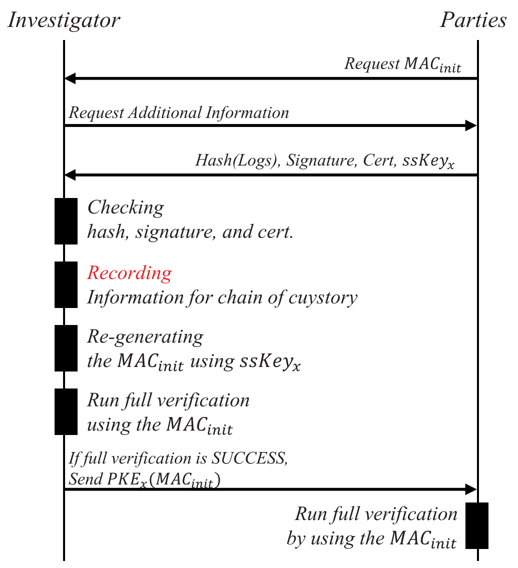

Investigator
Parties
Request MAC_{init}
Request Additional Information
Hash(Logs), Signature, Cert, $ssKey_x$
Checking hash, signature, and cert.
Recording Information for chain of cuystory
Re-generating the MAC_{init} using $ssKey_x$
Run full verification using the MAC_{init}
If full verification is SUCCESS, Send $PKE_x(MAC_{init})$
Run full verification by using the MAC_{init}

FIGURE 12. The squential diagram for requesting full verification from an investigator.

To look at this through an example, consider the scenario in which a driver is killed as a result of a car accident. In this case, the insurer and OEM would have to analyze the accident using the vehicle involved. T-Box derives a MAC key list from the bKey which is generated from Key_{init}. This means that if Key_{init} is reconstructed, the keys which have been used for generating MACs for data entries can be recovered again.

As shown in FIGURE 12, the insurer and OEM send their $ssKey_x$ to the investigator and request a restoration of the Key_{init}. At this time, the investigator is able to manage the signature for each party and hash value of all logs. This is an essential requirement to maintain a chain of custody for forensics. After the Key_{init} is reconstructed, the data with the same hash value recorded by the investigator may be included for consideration in digital evidence forensics. When data is modified by attackers in the threat model described in Section IV-B, the manipulated data is not accepted as digital evidence because it differs from the hash value managed by the investigator. The full verification method may prove to have legal ramifications on all data entries by substantiating the recorded data possessed by the investigator. After the OEM and insurer receive the Key_{init} from the investigator, they can verify all data entries with the full verification method using Key_{init}. The algorithm2 is pseudocode for the full verification.

VI. EVALUATION

A. PERFORMANCE MEASUREMENT

T-Box consists of a Client Application (CA) running on Linux in REE and a Trusted Application (TA) running on a secure OS in TEE supported by ARM TrustZone. Linaro, a non-profit organization for ARM-based open source engineers,

Algorithm 2 Full Verification

```
Require: DE and $Key_{init}$
Ensure: Pass or Fail
1: $bKey \leftarrow hash(Key_{init})$
2: if Request is verifying MAC then
3:     $eID \leftarrow DE.eID$
4:     $bID \leftarrow DE.bID$
5:     if $eID$ is the first entry in a block then
6:         /* Generate initial block key */
7:         $bKey \leftarrow Hash(bKey|bID)$
8:         for each key-index in $Key_{list}$ do
9:             /* Generate MAC key-list for each data entry */
10:            $Key_{list}$[key-index] $\leftarrow Hash(bKey|$key-index$)$
11:        end for
12:    end if
13:    $MAC_{eID} \leftarrow HMAC(Key_{list}[eID], DE)$
14:    $Digest \leftarrow hash(Digest, DE)$
15:    if $MAC_{eID} \neq DE.mac$ then
16:        /* Error: failed to verify MAC */
17:    end if
18: else if Request is verifying Signature then
19:    /* Verify Signature */
20:    $temp \leftarrow Verify(Key_{pubKey}, Signature)$
21:    if $Digest \neq temp$ then
22:        /* Error: failed to verify Signature */
23:    end if
24:    $Digest \leftarrow Null$
25: end if
```

has implemented ARM TrustZone features on Quick Emulator (QEMU) in 2014 [45]. ARM Trusted Firmware (ATF) and Open Portable Trusted Execution Environment (OP-TEE) are prerequisite software that runs on ARM TrustZone, which are being maintained as open source projects through GitHub [46], [47]. The ATF provides essential firmware for ARM processors, OP-TEE supports secure operating systems, and TEE libraries standardized by GlobalPlatform.

We have implemented T-Box based on QEMU before evaluating a real target board. If developers intend to implement a security solution based on ARM TrustZone, we recommend that they first implement all solutions on QEMU. Doing so can reduce programming overhead in the embedded software development environment. For example, various side issues occur during overhead, such as the re-flashing of built images to a non-volatile memory of a target board, time spent to reboot a board, and complications involving attachment of a debugger to a target board each time. Using QEMU is advantageous in that it reduces these instances of overheads and the developer can avoid side effects which are induced by a hardware. Thus, the developer is left to fully focus on functional software parts. Implemented CAs and TAs can be quickly verified based on QEMU to ensure that functional requirements of the software are implemented correctly. If there are issues when we evaluate a CA and a TA that are already tested and functional based on QEMU, then the issues are related to a hardware (target board). Thus, a developer can reduce scope of analysis when debugging on a target board. T-Box has been implemented on a symmetric multiprocessor using ARM Cortex-A53 cores provided by QEMU, which is the same ARM core of a real target board. After testing software functional requirements through the QEMU, T-Box migrates to a real target board, which is, specifically, a HiKey board [48]. The specification of the target board using for evaluation is shown in TABLE 2.

TABLE 2. The specification of the evaluation environment.

Category	Specifications
SoC	HiSilicon Kirin 620
CPU	ARM CortexTM-A53 Octa-core 64-bit(ARM v8)
Frequency	1.2GHz
RAM	2GB LPDDR3 SDRAM @ 800MHz
Storage	8GB eMMC
Software	OP-TEE v3.2.0

T-Box uses cryptographic operations, such as key generation, hash, MAC, and signing. Since a verification process is used by service providers and investigators, we do not consider the verification performance when implementing T-Box. We have chosen two methods as performance measurement tools: one is a performance monitor counter value (PMCTR) of an ARM performance monitor unit (PMU), and another is a clock_gettime() function in a standard C library. The PMCTR produces more precise measurements than other tools as it is measured using a per-CPU cycle counter [39]. It is also possible to directly access a hardware counter register in exception level 0 (EL0) mode through a performance control register (PMCR). The EL0 on ARMv8 is the same as the user mode on ARMv7. This helps to reduce measurement overhead of system calls, thus, providing more accurate measurements. Still, there are limitations when using the PMCTR. For instance, we must avoid situations in which a target process is migrated by a scheduler. This is largely because the PMCTR is a counter value for each core and, as such, measurement errors occur during target process migration. To best remedy this, a target process must be fixed to a target core and, accordingly, T-Box uses a sched_setaffinity() function to prevent migration [49]. We were able to read the PMCTR register in QEMU, but it was not deemed to be an accurate clock cycle because of the emulator environment limitations. Furthermore, an instruction to read the PMCTR was not operated in the target board. Therefore, we have used a clock_gettime() function to measure nanoseconds. After measuring 100 times for each item, we report that the result of the measurements is an average of all the items. We provide detailed results on the HiKey board in Table 4 of APPENDIX.

At first, we measured basic performance to determine a key size and appropriate algorithm for MACs and digital signatures, which is a cryptographic operation used in T-Box. The basic performance results were only measured using

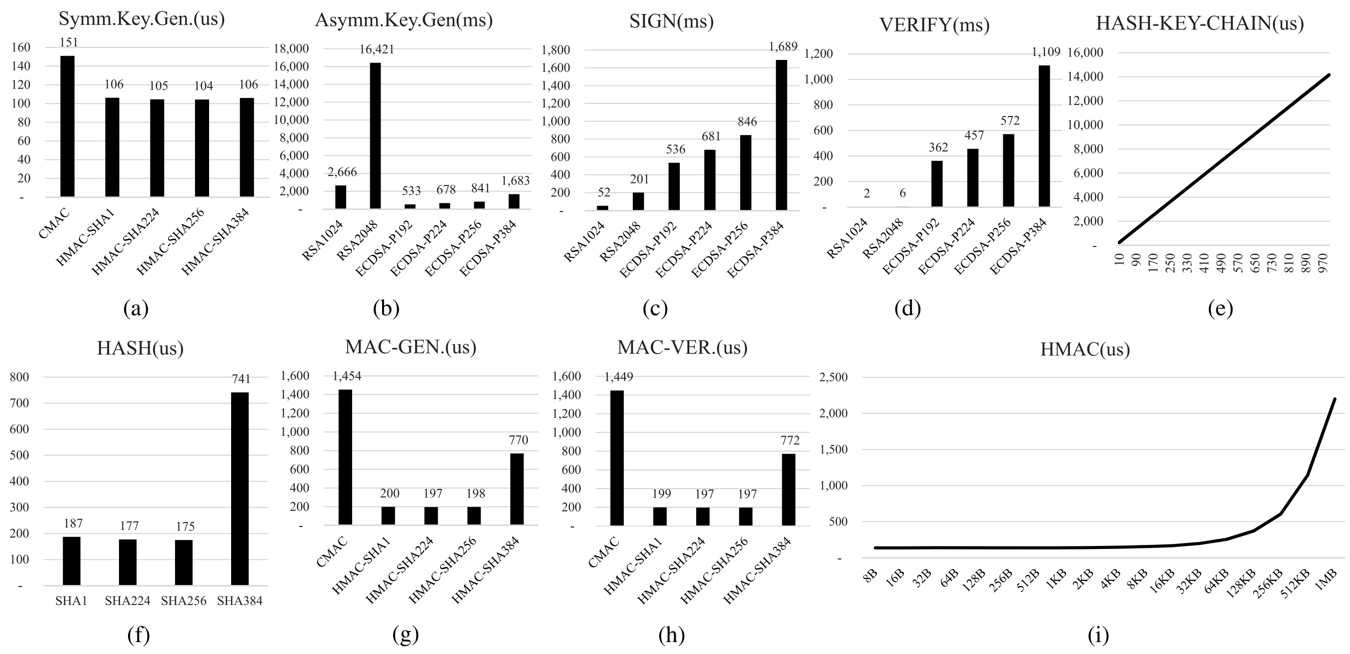


FIGURE 13. The overhead of cryptogrphic operations in OP-TEE (HiKey Board). (a) Symmetric key gen. (b) Asymmetric key gen. (c) Signing. (d) Signature verification. (e) Update key-list. (f) Hash operations. (g) MAC generation. (h) MAC verification. (i) HMAC-SHA256 with various input size.

GlobalPlatform standard libraries, with the exception of the implemented functions in T-Box. FIGURE 13a and FIGURE 13b show the results of key generation times. Since symmetric key generation time is not significantly dependent on varying key sizes, we have selected 256bit, which supports secure key strength. Additionally, we chose RSA2048 and ECDSA-P224 as potential algorithms for signing operations. Not that when considering the public key pair generation time only, we selected ECDSA-P224. The reason is that ECDSA-P224 key generation time is approximately 24 times faster than RSA2048. As a result, we selected RSA2048 instead of ECDSA-P224 due to signature generation time.

FIGURE 13c and FIGURE 13d show the performance of the functions required for the signing operation. RSA2048 is based on RSASSA-PKCS1-v1.5-SHA256. In the signing performance, RSA2048 is 3.4 times faster than ECDSA-P224. Even though RSA2048 key generation time is approximately 24 times slower than ECDSA-P224 and the signature size of RSA2048 is about four times larger than ECDSA-P224, we chose RSA2048 because the length of the ECDSA signature is double the key size. In fact, T-Box generates the signing key only once during the initializing phase. After initialization, it is able to operate signing processes repetitively without any key generation. If the target board supports hardware accelerators for cryptographic operations, these performance results will be different.

In T-Box, a hash operation is used for the hash key chain and MAC operation. FIGURE 13f, FIGURE 13g, and FIGURE 13h show the performance of such hash operations and MAC operations. Previously, we set the symmetric key size to 256bit, which means that we selected SHA256 and HMAC256–two algorithms which are compatible with the 256bit key size, yet are not slower than other hash operations.

In order to determine the size of a data entry, we have measured processing time according to the data size based on HMAC-SHA256, which was also determined previously. We expected that as the data size increases, the processing time would also increase proportionally. However, as shown in FIGURE 13i, there is no significant change until a data entry size reaches 32KB. This is caused by the OP-TEE overhead itself and the world-switching time. There is no significant difference up to 32KB because overhead is relatively large. Finally, we determined the size of the log entry to be 32KB. The spent time for generating a MAC of 32KB log data is approximately 198us.

B. OPTIMIZATION

To satisfy the second requirement (R-2), for which T-Box requires minimal memory used for data recording, we should find an optimal number of buffers. But prior to specifying the number of buffers, the following challenges must be resolved to execute T-Box:

- The MAC processing time must be faster than data generation time. If the MAC processing time is slower than the data generation period, the data is continuously stacked in the buffer and the buffer full scenario occurs. In this sense, it does not satisfy the requirements. Therefore, if the bandwidth of the packet on an IVN is faster than the MAC generation time, a single data entry should contain several packets that are delivered from the network monitor.

- All data entries in the buffer must be processed before starting a signing process for the next block. The RSA2048 signing operation spends approximately 200ms. In other words, if each data entry is generated every 2ms, one hundred data entries will be stacked in the buffer when a whole block signing is processed. Since the stacked data entries can make a buffer full scenario, they must be processed before the next block begins a signing operation.

To resolve these challenges, we are required to locate two optimal numbers. The first number is for the circular ring buffer and the second is the number of data entries in a block. These can be determined by the following steps.

The generation cycle of a data entry can be calculated by the customer's required bandwidth.

$$F_{GEN} = \frac{Bandwidth}{L_{DE}} \qquad T_{GEN} = \frac{1s}{F_{GEN}}$$

The ring buffer size (N_{RB}) to store subsequently generated data entries during a time in which a current block is being signed can be calculated from T_{GEN} and T_{SIGN} as shown below. The reason we apply the ceiling and alpha value is to avoid the buffer full scenario. Since general-purpose operating systems do not provide real-time property, alpha is set to 10% of the calculated number of ring buffers.

$$N_{RB} = \lceil \frac{T_{SIGN}}{T_{GEN}} + \alpha \rceil$$

If N_{RB} is determined as indicated above, the next parameter to be determined is N_{DE}, which is the number of data entries in a block. In order to calculate N_{DE}, it was necessary to obtain the total processing time for one block. The MAC is generated for entire data entries in a block along with a hash key chain generates a key list at the time when a block is first generated. If a block is full of data entries, then it is signed. This condition can be expressed by the following formula. T-Box requires world-switching once and MAC operations twice to process one data entry. As we described in Subsection V-E, the two MAC operations are for a current data entry and a past data entry.

$$T_{Block} = N_{DE} \times (2 \times T_{MAC} + T_{WS}) + T_{KC} + T_{SIGN}$$

T-Box should process all data entries stacked in the ring buffer before the next block starts a signing operation. The N_{DE} can be calculated such that the time required to process one block (T_{Block}) is faster than the time it takes for the next block to be full.

$$\begin{aligned} N_{DE} \times T_{GEN} &> T_{Block} \\ \rightarrow N_{DE} &> \frac{T_{Block}}{T_{GEN}} \\ \rightarrow N_{DE} &> \left\lceil \left(\frac{T_{KC} + T_{SIGN}}{T_{GEN} - (2 \times T_{MAC} + T_{WS})} \right) \right\rceil \end{aligned}$$

For example, in order to process the bandwidth of 100Mb/s for the Ethernet backbone network, we can calculate all parameters through the formulas above as shown in TABLE 3.

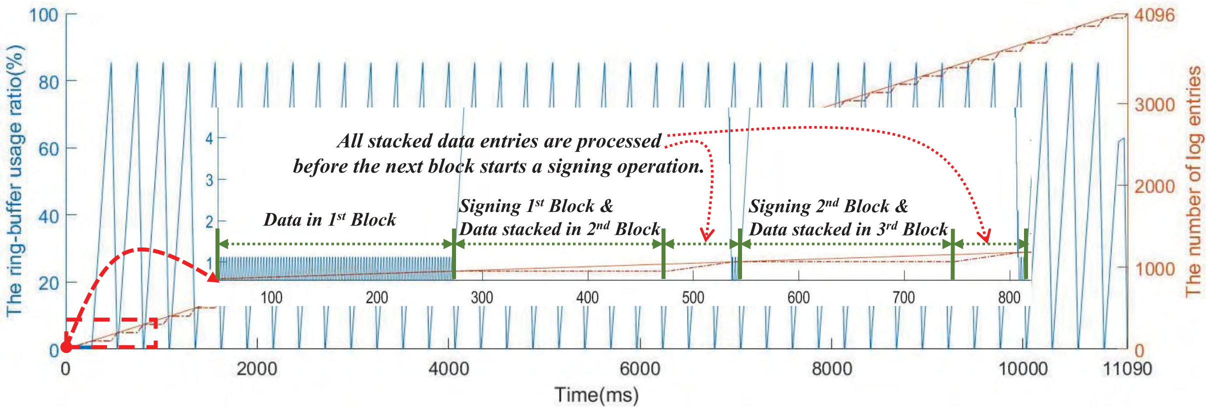
TABLE 3. **The parameters for 100Mb/s data recording.**

Parameter (unit)	Value
nanosecond (ns)	1,000,000,000
req.Banwidth (B/s) $(100 \times 1,024 \times 1,024$ / 8Bit)	13,107,200
Size of log data (Byte) $(32 \times 1,024)$	32,768
Size of meta data (Byte) (predefined by design)	88
Size of data entry (Byte) (L_{DE}: 32,768 + 88)	32,856
Time Cycle of T-Generator (ns) $(T_{GEN} = 1sec/\frac{Bandwidth}{L_{DE}})$	2,500,000
Time Spent by 32KB MAC (ns) (T_{MAC}: predefined in design)	198,408
Time spent by Signing (ns) (T_{SIGN}: predefined in measurement)	200,595,033
Time spent by Key-chain update (ns) (T_{KC}: predefined in measurement)	1,221,092
Time spent by World-Switching (ns) (T_{WS}: predefined in measurement)	89,883
Alpha(α) $(\alpha = \frac{T_{SIGN}}{T_{GEN}} \times 0.1)$	8.0
The Number of ring buffers $(N_{RB} = \lceil \frac{T_{SIGN}}{T_{GEN}} + \alpha \rceil)$	89
The Number of log-entries (N_{DE})	101
Memory usage for buffer (Byte) $(Mem_{usage} = N_{RB} \times L_{DE})$	2,924,184 (2.8MB)

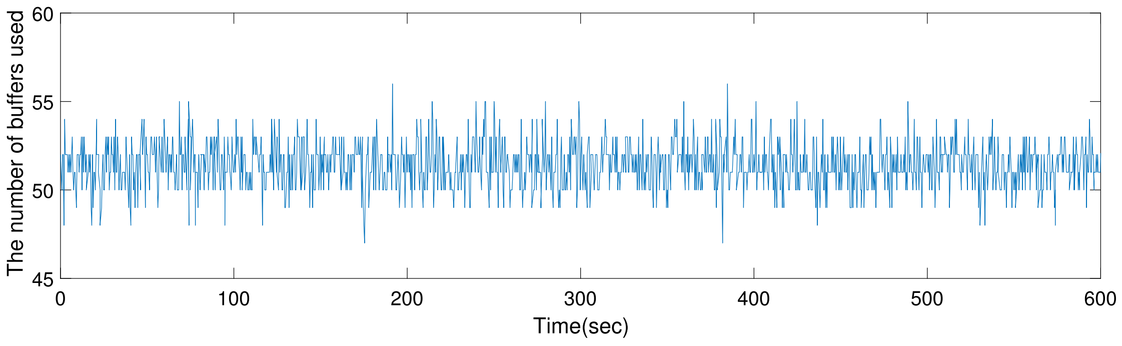
The T_{KC} can be neglected because it is a relatively insignificant overhead compared with T_{SIGN}. However, in the case that the number of data entries in a block is relatively large, the overhead of T_{KC} must be included because a hash function is performed according to the number of data entries (shown in FIGURE 13e).

C. EVALUATION WITH PREDEFINED PARAMETERS

At first, we continuously generate data entry at every 2.5ms for 10 minutes to establish whether or not T-Box is capable of handling up to 100Mb/s. T-Box records 2,200 blocks with signatures, and each block correctly includes 101 data entries with corresponding MACs. Then we concentrate on performance records for 4,096 data entries and inspect the ring buffer usage ratio as shown in FIGURE 14a. While T-Generator sequentially generates 4,096 data entries, the ring buffer consumes approximately up to 85% when a current block is being signed. Meanwhile, remaining data entries in the ring buffer are completely processed before the next block commences a signing operation. This process takes approximately 11.09 seconds for 4,096 data entries. After that, we generate data entries randomly spanning 2.5ms to 5ms time intervals to estimate the real situation of T-Box in a dynamic environment. In this evaluation, T-Box uses an average of 51.4 buffers over 10 minutes and a range of 47 to 56 buffers, as shown in FIGURE 14b. In conclusion, a buffer full condition does not present itself, and it is reasonable to assess that T-Box can theoretically handle bandwidth up to 512Mb/s (64MB/s) (see Appendix).



(a)



(b)

FIGURE 14. The real-time operation status of T-Box. (a) The analysis of consuming ring buffers when T-Box records up to 4096 data entries. (b) The number of used buffers when T-Box records data entries generated at random time for 10 minutes.

VII. DISCUSSIONS

A. DATA COMPRESSION

Next, T-Box must also consider the size of the recording data. In Section VI-C, the size of recorded data is roughly 7.3GB if T-Box fully records 100 Mb/s bandwidth for 10 minutes. We experimentally collected 270 MB of CAN packets over a one-hour time period through the OBD-II port from our research vehicle. When the collected data is compressed via a zip compression algorithm, the size shrinks to 47MB. The result shows that compression algorithms can reduce the recorded data approximately 17.4%. Applying compression algorithms is more efficient in reducing storage space and network overhead, but further study is needed to understand precise application times. When compressing algorithms are positioned in a recording period in T-Box, additional parameter settings also become necessary. If they are positioned after the recording period, T-Box requires additional operations to manage the compressed data carefully. If even one bit of the compressed data is corrupted, the original data cannot be recovered.

B. BLOCKCHAIN

In recent years, the hash chain scheme has become widely known to the public as more blockchain networks use it. This scheme is similar in that it guarantees the correctness of sequentially ordered data, but there remain differences between the previous works and blockchain networks. Existing methods use hash operations with a one-way forward secrecy ensures the integrity and sequential connectivity of the data being recorded by pre-generating consecutive key arrays for the MAC, or by including the data in the hash operation. In particular, a two-dimensional key array has the capacity to provide rapid verification of arbitrary data. The hash chain used in blockchain networks requires extremely heavy computation resources called a proof of work (PoW) to find a hash value that is less than the specified target value to prevent a 51% attack. Therefore, the internal data, transactions, or contracts included in blocks is determined by miners that generate the block regardless of the generation time [50], [51]. However, in the case of the recording system proposed by T-Box, the data entries included in a block must be stored in the order of generation time, and they must be recorded in real time without missing any data entries. Additionally, they should also satisfy non-repudiation against recorded data. To do this, T-Box generates a two-dimensional key array from a seed key through a hash operation, uses it for MAC operation, and generates a digital signature for each block. The keys used for cryptographic operations are securely stored in the space allocated to the hardware-based isolated execution environment and perform the

TABLE 4. Measurement results on HiKey Board.

Symmetric Key Generation			Asymmetric Key Generation		
Type	Mean (us)	Std. Dev (us)	Type	Mean (ms)	Std. Dev (ms)
CMAC	151.02	5.55	RSA1024	2666.39	1004.55
HMAC-SHA1	106.27	6.04	RSA2048	16421.29	6149.95
HMAC-SHA224	104.52	4.39	ECDSA-P192	532.52	0.76
HMAC-SHA256	104.33	3.93	ECDSA-P224	677.78	0.73
HMAC-SHA384	106.05	2.90	ECDSA-P256	841.48	0.66
-	-	-	ECDSA-P384	1683.40	1.63

MAC Generation (32KB)			MAC Verification (32KB)		
Type	Mean (us)	Std. Dev (us)	Type	Mean (us)	Std. Dev (us)
CMAC	1453.68	11.00	CMAC	1449.38	10.52
HMAC-SHA1	199.98	3.43	HMAC-SHA1	199.48	4.12
HMAC-SHA224	197.39	3.62	HMAC-SHA224	197.40	4.20
HMAC-SHA256	197.87	4.77	HMAC-SHA256	197.17	3.22
HMAC-SHA384	770.23	7.70	HMAC-SHA384	771.63	7.97

HMAC Operation			World-Switching		
Type	Mean (us)	Std. Dev (us)	Type	Mean (us)	Std. Dev (us)
8B	137.71	4.91	8B	89.22	4.17
64B	138.90	4.82	64B	89.05	3.77
512B	139.05	3.70	512B	89.25	3.55
4KB	145.93	4.80	4KB	89.17	3.73
32KB	198.41	6.32	32KB	89.88	2.56
256KB	607.90	13.74	256KB	97.62	4.70
512KB	1146.13	41.41	512KB	106.14	7.61
1MB	2201.23	107.22	1MB	123.80	13.56

Message Digest Generation (32KB)		
Type	Mean (us)	Std. Dev (us)
SHA1	187.15	5.63
SHA224	177.48	4.98
SHA256	174.96	3.92
SHA384	741.48	8.10

Signing		
Type	Mean (ms)	Std. Dev (ms)
RSA1024	52.43	0.21
RSA2048	200.60	0.41
ECDSA-P192	535.74	0.69
ECDSA-P224	681.42	0.87
ECDSA-P256	845.62	0.74
ECDSA-P384	1689.15	1.35

Verifying		
Type	Mean (ms)	Std. Dev (ms)
RSA1024	2.45	0.01
RSA2048	5.82	0.01
ECDSA-P192	362.49	4.53
ECDSA-P224	456.81	5.13
ECDSA-P256	571.55	5.68
ECDSA-P384	1109.32	10.23

Hash Key Chain Generation		
Num. of keys	Mean (us)	Std.Dev (us)
10	236.64	5.55
20	378.37	12.12
30	519.36	9.77
40	659.36	13.06
50	800.18	10.41
60	940.86	8.91
70	1080.63	8.80
80	1221.09	10.34
90	1363.85	10.40
100	1503.29	10.18
200	2910.31	11.72
300	4314.81	8.39
400	5724.86	13.76
500	7128.77	12.64
600	8536.23	12.53
700	9940.23	19.34
800	11346.87	17.61
900	12750.20	17.57
1000	14159.59	25.04

Theoretical-practical bandwidth with a 32KB data entry					
Bandwidth (MB)	T_{GEN}(ms)	Alpha(α)	N_{RB}	N_{DE}	Mem. Usage (MB)
12	2.60	8	36	85	2.66
13	2.40	9	36	92	2.88
-	-	-	-	-	-
62	0.50	40	438	11,644	13.72
63	0.50	41	445	21,627	13.94
64	0.49	42	452	127,634	14.16
65	0.48	42	459	**-34,029**	14.38
-	-	-	-	-	-

The table illustrates crypto operation performance on a HiSilicon Kirin 620 board, version OP-TEE 3.2.0. The predefined parameters in TABLE 3 are highlighted in red and underscored. The last row indicates that T-Box theoretically records 64MB/s of bandwidth. If it exceeds this amount, a buffer full situation will present, as the data entry is generated faster than the time it takes to process the data entries accumulated in the ring buffer.

cryptographic operation there without key exposure. Although we do not consider blockchains in this paper, the specific information of the blocks recorded in T-Box can be extended to blockchain networks in the future, suggesting various applications of data recording methods.

C. ABNORMAL EVENT DETECTION

When it comes to security, automotive data would also require additional verification. T-Box enables the results of event detection to be verified by ensuring the integrity of source data. For example, an intrusion detection system (IDS), which is one promising process for securing vehicles from cyber attacks, analyzes automotive data to detect automotive intrusions [52]. Accordingly, the IDS should leverage the data recorded by T-Box for integrity verification. In addition, dash cameras have been installed recently in many vehicles for safety purposes such as emergency brake assist protocols. We believe that this type of automotive application should operate exclusively with non-modified data [53]. For secure automotive applications, we are planning to design T-Box to operate with an event detection method.

D. PRIVACY ISSUES

In this paper, we have solved challenges regarding a trusted automotive data recording system that operates in real time without significant consideration for privacy issues. Although a simple method is able to encrypt all of the data with a MAC key in [15], we thought that this was not suitable to manage keys securely between all of the service providers in an automotive environment in the context of this paper. As such, we have focused solely on solving the presented limitations of automotive data recording systems. Based on the results of this study, we discovered that an ARM TrustZone-based solution can be installed as a real-time data recording system in a vehicle. In the future, we will expand this research to locate ways in which to solve the privacy issues surrounding T-Box. Privacy issues must be addressed, in particular, in light of the introduction of the General Data Protection Regulation (GDPR) in the European Union, which was enacted and promulgated on May 25, 2018.

VIII. CONCLUSION

Automotive data is becoming integrated with big data to maximize utility of information, and it is becoming evermore imperative to protect this data from malicious agents. Thus, in this paper we defined the three security requirements for automotive data recording systems. To satisfy the first requirement (R-1), T-Box utilizes a two-dimensional hash chain using a sequential monotonic counter to support forward integrity. Thus, T-Box detects malicious data manipulations such as data deletion, replacement, and replaying. Moreover, we have implemented a new scheme which detects data truncation by handling two data entries concurrently. For the second requirement (R-2), T-Box uses the three-indexed circular ring buffer which is shared with T-Generator, T-Recorder, and T-Writer. As a result, T-Box consumes minimal memory, which is obtained through the formalized method for recording automotive real-time data. T-Box has shown that it only requires 2.8MB of memory for handling 100Mb/s of bandwidth. Lastly, T-Box makes the investigator an important party to manage the chain of custody by using Shamir's Secret Sharing Scheme, enabling the use of recorded data as digital forensic evidence and, thus, satisfying the third requirement (R-3).

We have implemented the components of T-Box to meet the described three security requirements, but more importantly, the automotive data recorded by T-Box satisfies all security properties defined at the outset of this paper, namely data integrity, continuity, and non-repudiation. The evaluation results on the HiKey board provide a foundation on which an automotive data recording system can feasibly be developed. Finally, our presentation of T-Box demonstrates its successful recording of 100Mb/s of bandwidth and its ability to theoretically handle up to 512Mb/s (64MB/s) (see Appendix), which can serve as a launching board for future innovative systems research.

APPENDIX
MEASUREMENT RESULTS ON HIKEY BOARD

See table 4.

ACKNOWLEDGEMENT

The authors would like to thank the editors and anonymous reviewers who gave us valuable feedback to improve the quality of this paper.

REFERENCES

[1] *Automotive Gateway: A Key Component to Securing the Connected Car*, NXP Ltd, 2018.

[2] VEMOCO Telematics Ltd. *Vehicle Mobile Communication*. Accessed: Sep. 8, 2018. [Online]. Available: https://vemoco.com/en/insurance

[3] C. Miller and C. Valasek, *Remote Exploitation of an Unaltered Passenger Vehicle*. New York, NY, USA: Black Hat, 2015.

[4] K. T. Cho and K. G. Shin, "Viden: Attacker identification on in-vehicle networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA: ACM, 2017, pp. 1109–1123.

[5] ARM Ltd. *Driving Into the Future With Automotive Technology*. Accessed: Jul. 10, 2018. [Online]. Available: https://www.arm.com/solutions/automotive

[6] Samsung. (Jan. 19, 2017). *Samsung's Exynos Processors Selected to Revolutionize Audi's Next-Generation In-Vehicle Infotainment*. Accessed: Nov. 8, 2017. [Online]. Available: https://www.samsung.com/semiconductor/insights/

[7] C. W. Dukes, "Committee on national security systems (CNSS) glossary," CNSSI, Fort Meade, MD, USA, Tech. Rep. 4009, 2015.

[8] J. Kelsey, B. Schneier, and C. Hall, "An authenticated camera," in *Proc. 12th Annu. Comput. Secur. Appl. Conf.*, Dec. 1996, pp. 24–30.

[9] B. Schneier and J. Kelsey, "Automatic event-stream notarization using digital signatures," in *Proc. Int. Workshop Secur. Protocols*. Berlin, Germany: Springer, 1996, pp. 155–169.

[10] M. Bellare and B. Yee, "Forward integrity for secure audit logs," Dept. Comput. Sci. Eng., Univ. California San Diego, San Diego, CA, USA, Tech. Rep., vol. 184, 1997.

[11] M. Bellare and B. Yee, "Forward-security in private-key cryptography," in *Proc. Cryptographers Track RSA Conf.* Berlin, Germany: Springer, 2003, pp. 1–18.

[12] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines.," in *Proc. USENIX Security Symp.*, vol. 98, 1998, pp. 53–62.

[13] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Trans. Inf. Syst. Secur.*, vol. 2, no. 2, pp. 159–176, 1999.

[14] J. Kelsey and B. Schneier, "Minimizing bandwidth for remote access to cryptographically protected audit logs," in *Proc. 2nd Int. Workshop Recent Adv. Intrusion Detection (RAID)*, Sep. 1999.

[15] J. E. Holt, "Logcrypt: Forward security and public verification for secure audit logs," in *Proc. Australas. Workshops Grid Comput. E-Res.*, Hobart, TAS, Australia, vol. 54, 2006, pp. 203–211.

[16] D. Ma and G. Tsudik, "Extended abstract: Forward-secure sequential aggregate authentication," in *Proc. IEEE Symp. Secur. Privacy*, May 2007, pp. 86–91.

[17] D. Ma, "Practical forward secure sequential aggregate signatures," in *Proc. ACM Symp. Inf., Comput. Commun. Secur.*, 2008, pp. 341–352.

[18] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Trans. Storage*, vol. 5, no. 1, p. 2, 2009.

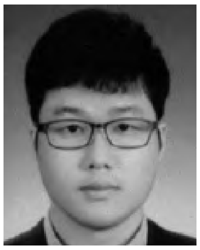
[19] A. A. Yavuz and P. Ning, "BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2009, pp. 219–228.

[20] A. A. Yavuz, P. Ning, and M. K. Reiter, "BAF and FI-BAF: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems," in *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 2, p. 9, 2012.

[21] C. N. Chong, Z. Peng, and P. H. Hartel, "Secure audit logging with tamper-resistant hardware," in *Proc. IFIP Int. Inf. Secur. Conf.* Boston, MA, USA: Springer, 2003, pp. 73–84.

[22] A. Sinha, L. Jia, P. England, and J. R. Lorch, "Continuous tamper-proof logging using TPM 2.0," in *Proc. Int. Conf. Trust Trustworthy Comput.* Cham, Switzerland: Springer, 2014, pp. 19–36.
[23] V. Karande, E. Bauman, Z. Lin, and L. Khan, "SGX-log: Securing system logs with SGX," in *Proc. ACM Asia Conf. Comput. Commun. Secur.* New York, NY, USA: ACM, 2017, pp. 19–30.
[24] *IEEE Standard for Motor Vehicle Event Data Recorders (MVEDRs) Amendment 1: MVEDR Connector Lockout Apparatus (MVEDRCLA)*, IEEE Standard 1616-2004, 2004.
[25] H. C. Gabler, D. J. Gabauer, H. L. Newell, and M. E. O'Neill, "Use of event data recorder (EDR) technology for highway crash data analysis," NCHRP Project, Transp. Res. Board, Washington, DC, USA, Dec. 2004, pp. 17–24.
[26] H. Mansor, K. Markantonakis, R. N. Akram, K. Mayes, and I. Gurulian, "Log your car: The non-invasive vehicle forensics," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 974–982.
[27] C. Patsakis and A. Solanas, "Privacy-aware event data recorders: Cryptography meets the automotive industry again," *IEEE Commun. Mag.*, vol. 51, no. 12, pp. 122–128, Dec. 2013.
[28] The New York Times Company. *How A Self-Driving UBER Killed a Pedestrian in Arizona*. Accessed: Mar. 20, 2018. [Online]. Available: https://www.nytimes.com
[29] Tesla Blog. *An Update on Last Week's Accident*. Accessed: Mar. 30, 2018. [Online]. Available: https://www.tesla.com/blog/update-last-week's-accidentl.
[30] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proc. IEEE*, vol. 93, no. 6, pp. 1204–1223, Jun. 2005.
[31] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin, "Intra-vehicle networks: A review," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 534–545, Apr. 2015.
[32] Y. Kim and M. Nakamura, "Automotive ethernet network requirements," in *Proc. IEEE 802.1 AVB Task Force Meeting*, Mar. 2011, pp. 1–10.
[33] A. Tomlinson, "Introduction to the TPM," in *Smart Cards, Tokens, Security and Applications*. Springer, 2017, pp. 173–191.
[34] *Intel Software Guard Extensions Developer Guide*, Intel Corporation, Santa Clara, CA, USA, 2017.
[35] *Intel Software Guard Extensions Programming Reference*, Intel Corporation, Santa Clara, CA, USA, 2014
[36] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.
[37] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel software guard extensions: Epid provisioning and attestation services," Intel Corp., Santa Clara, CA, USA, White Paper, vol. 1, 2016, pp. 1–10.
[38] *Security Technology Building a Secure System Using Trustzone Technology (White Paper)*, ARM Ltd, Cambridge, U.K., 2009.
[39] *ARM Cortex-A53 MPCore Processor*, ARM Ltd, Cambridge, U.K., 2015, pp. 1–577.
[40] GlobalPlatform. *Globalplatform Specification Library*. Accessed: Sep. 8, 2018. [Online]. Available: https://globalplatform.org/specs-library
[41] "The trusted execution environment: Delivering enhanced security at a lower cost to the mobile market," Global Platform, White Paper, 2011, pp. 1–26. Accessed: May 12, 2018. [Online]. Available: http://globalplatform.org/wp-content/uploads/2018/04/GlobalPlatform_TEE_Whitepaper_2015.pdf
[42] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.
[43] G. Irazoqui and X. Guo, *Cache Side Channel Attack: Exploitability and Countermeasures*, vol. 2017. Singapore: Black Hat Asia, 2017.
[44] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "Armageddon: Cache attacks on mobile devices," in *Proc. USENIX Security Symp.*, 2016, pp. 549–564.
[45] Linaro. (Sep. 26, 2014). *Arm TrustZone in QEMU*. [Online]. Available: https://www.linaro.org/blog/arm-trustzone-qemu
[46] Arm Ltd. and Contributors. *Trusted Firmware-A*. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware
[47] OP-TEE. *Open Portable Trusted Execution Environment*. Accessed: Mar. 30, 2018. [Online]. Available: https://www.optee.org
[48] 96Boards. *HiKey Board Which is Available From LeMaker*. Accessed: Feb. 12, 2018. [Online]. Available: https://www.96boards.org/product/hikey
[49] GNU. *Limiting Execution to Certain CPUs*. Accessed: Sep. 8, 2018. [Online]. Available: http://www.gnu.org/software/libc
[50] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Feb. 10, 2019. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[51] V. Buterin, "A next generation smart contract & decentralized application platform," White Paper, 2014. Accessed: Feb. 10, 2019. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper
[52] W. Choi, K. Joo, H. J. Jo, M. C. Park, and D. H. Lee, "VoltageIDS: Low-level communication characteristics for automotive intrusion detection system," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 8, pp. 2114–2129, Aug. 2018.
[53] X. Chang, Y.-L. Yu, Y. Yang, and E. P. Xing, "Semantic pooling for complex event analysis in untrimmed videos," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 8, pp. 1617–1632, Aug. 2017.

SEUNGHO LEE received the B.S. and M.S. degrees in computer science from Pukyong National University, Busan, South Korea, in 2006 and 2010, respectively. He is currently pursuing the Ph.D. degree in information security with the Graduate School of Information Security, Korea University, Seoul, South Korea. Since 2006, he has been with the Software Research and Development Center, Samsung Electronics Corp., Giheung, South Korea. In the past, he has worked on embedded system security projects for the IoT and automobiles. His research interests are embedded system security, audit logs, and automotive security.



WONSUK CHOI received the B.S. degree in mathematics from the University of Seoul, Seoul, South Korea, in 2008, and the M.S. and Ph.D. degrees in information security from Korea University, Seoul, in 2013 and 2018, respectively, where he is currently a Postdoctoral Researcher with the Graduate School of Information Security. His research interests include security for body area networks, usable security, applied cryptography, and smart car security.

HYO JIN JO received the B.S. degree in industrial engineering and the Ph.D. degree in information security from Korea University, Seoul, South Korea, in 2009 and 2016, respectively. From 2008 to 2010, he was a Postdoctoral Researcher with the Department of Computer and Information Systems, University of Pennsylvania, Philadelphia, PA, USA. In 2018, he joined the Department of Software Convergence, Hallym University, Chuncheon, South Korea, as an Associate Professor. His research interests include security for cyber-physical systems, applied cryptography, and privacy-preserving methods.

DONG HOON LEE received the B.S. degree from the Department of Economics, Korea University, Seoul, South Korea, in 1985, and the M.S. and Ph.D. degrees in computer science from The University of Oklahoma, Norman, OK, USA, in 1988 and 1992, respectively. He has been with the Faculty of Computer Science and Information Security, Korea University, since 1993. He is currently a Professor and the Director of the Graduate School of Information Security, Korea University. His research interests include cryptographic protocol, applied cryptography, functional encryption, software protection, mobile security, vehicle security, and ubiquitous sensor network (USN) security.