IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 10, 2018, accepted August 10, 2018, date of publication August 22, 2018, date of current version September 7, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2866626

Code-Based Authentication Scheme for Lightweight Integrity Checking of Smart Vehicles

JOONSANG YOO[1] AND JEONG HYUN YI[2]
[1]Department of Software Convergence, Soongsil University, Seoul 06978, South Korea
[2]School of Software, Soongsil University, Seoul 06978, South Korea

Corresponding author: Jeong Hyun Yi (jhyi@ssu.ac.kr)

This work was supported by the Institute for Information and Communications Technology Promotion grant funded by the Korea Government (MSIT), Automatic Deep Malware Analysis Technology for Cyber Threat Intelligence, under Grant 2017-0-00168.

ABSTRACT The stable and seamless connection of the mobile communication system is expected to closely link an unprecedented number of things, including smart cars such as autonomous vehicles, in the near future. These vehicles will operate based on the data transmitted over hundreds of sensors. However, an attacker could remotely control a car by intercepting and tampering with these data, and even threaten the life of the driver. In this paper, we propose a code-based authentication scheme that provides both secure booting and lightweight data integrity checking to prevent unauthorized remote control. First, we split some of the core code involved in booting the car, and divide it into several secret pieces known as a *share polynomial*. One of these is distributed to the driver and is then used to reconstruct the booting code, allowing only the driver with this share polynomial to recover the code and start the car. The remaining share polynomials are distributed to the vehicle control unit and used to generate a lightweight pairwise key. Since the proposed scheme requires only multiplication to generate the pairwise key, it is computationally efficient compared to conventional schemes that require expensive exponential computation. Additional communication is not required to set up the pairwise key, because it needs nothing but the `id` value of the partner. Thus, our scheme is highly suitable for low-performance devices inside the car. We used the Robot OS system, a widely used autonomous vehicle platform, to implement the proposed scheme and evaluated its feasibility on various test cases.

INDEX TERMS Smart device security, data integrity, secure boot, secret sharing, key management.

I. INTRODUCTION

The number of mobile devices and things connected to the Internet is expected to reach billions in the near future [1], [2]. At the forefront of this trend is mobile internet technology, which promises to bring low latency and high reliability. These features will allow completely new mobile services such as connected vehicles, connected healthcare, and connected home appliances, which require a faster and more stable network environment.

One of the representative connected things of the future is smart vehicles, including autonomous vehicles and electric vehicles. According to Gartner [3], in 2020, 80% of cars on the road will be connected to the Internet. Obviously, system software, rather than hardware such as a mechanical engine, will play a key role in automobiles. This means vehicle hacking is not just a theory, but a reality. Hackers could remotely control the infotainment system, Anti-lock Braking System (ABS), or even turn off the engine of the vehicle [4]. Furthermore, they could access the Electronic Control Unit (ECU) of the vehicle from a remote location via the smartphone application. The automaker Fiat Chrysler had to issue a recall after a wireless hack on the Jeep Cherokee [5]. It is also widely known that hackers took remote control of a Tesla Model S from 12 miles away [6]. In recent years, software codes that perform functional safety operations are required to comply with ISO 26262, the functional safety standard for automotive software.

The reason why these hacks are possible is that encryption and integrity checking for the data flowing from hundreds of sensors have not been implemented properly. It means that existing IT security technologies cannot be applied to smart cars that require lightweight cryptography technology. Therefore, this paper proposes a lightweight code-based authentication scheme to provide ultra-fast data encryption and integrity checking functionality in the smart vehicle environment.

The proposed scheme uses *code splitting* and a *bivariate polynomial-based secret sharing* scheme to provide secure booting and data integrity checking that allows only trusted devices to send and receive sensing data. This is achieved by first splitting some of the core code involved in booting the smart car. We define this code as the *secret master*. In addition, the bivariate polynomial-based secret sharing scheme is applied to the secret master code, and then divided into several share polynomials, before distributing these pieces to the devices inside and outside the smart car. At this time, at least one external mobile device must be included. This allows the smart car to boot only if there is a mobile device outside the smart car.

Thus, the proposed scheme makes it impossible for an attacker who does not own a separate share polynomial to control the smart car from the outside. In addition, since the proposed scheme is based on threshold secret sharing, it is possible to self-configurate the share polynomial by collaboration among existing devices even when new internal or external devices are added. These share polynomials are also used to generate pairwise symmetric keys for secure and reliable data exchange between sensors after booting. It helps reducing risks associated with software for safety functions to a tolerable level by providing anomaly detection on sensing data.

The contribution of this paper is as follows.

1) We propose a *secure code splitting* scheme and a *code-based user authentication* scheme using bivariate secret sharing that is secure from malicious code injection and a code tampering attack. Since the separate core code is not uniquely stored on any device, it prevents accidental exposure due to device loss. In addition, since it is based on secret sharing, even if a new device is added, a new share polynomial can be provided by collaboration among existing devices without changing the secret master.
2) It also provides a *lightweight pairwise key establishment* scheme that is also applicable to embedded devices such as sensors. Each of the devices can create the symmetric key with only the own share polynomial and the `id` value of the communication counterpart. Since the proposed scheme requires only multiplication operations, it is much faster than the conventional RSA-based session key distribution [7], Diffie-Hellman key exchange [8], and Shamir's secret sharing [9].
3) Based on the Robot OS (ROS) system [10], which is widely used as an autonomous vehicle platform, *performance evaluation* was conducted according to various test cases. This proved that the proposed scheme is feasible in the real smart car environment.

This paper is organized as follows. In Section II, we discuss the security vulnerabilities of smart vehicles and related work conducted to solve them. Section III introduces the background knowledge needed to understand the proposed scheme. Section IV details the proposed scheme, and Section V discusses the security arguments for the proposed scheme. Section VI presents an evaluation of the implementation and performance of the proposed scheme, and the paper concludes with Section VII.

II. RELATED WORK

Today's automobiles usually contain between 80 and 150 built-in ECUs, which are assigned to almost every function such as engine control, shift control, attitude control, and airbag control. Most of the functions of the vehicle are realized through the cooperation of these ECUs. For example, the Electronic Stability Control (ESC) system is implemented by involving several ECUs such as those controlling the speed of the wheels, the direction of the steering wheel, and the accelerometer. In this case, ECUs communicate via the Controller Area Network (CAN) bus, which is a non-host bus network protocol and uses a publish-subscribe communication model. In this scheme, the packet is broadcasted to all nodes and transmitted. Because of this, packets traversing the CAN bus can be intercepted and tampered with by the malicious component at any time, and are subject to further exploitation.

Therefore, in order to complement these vulnerabilities, attempts have been made to distribute the key to each ECU or sensor device, to verify the authenticity and integrity of the vehicle internal packet and sensing data. Existing studies on the integrity checking of the data transmitted on the in-vehicle network are analyzed by using the key exchange method.

A. SYMMETRIC KEY BASED SESSION KEY DISTRIBUTION

1) TRUSTED KEY SERVER BASED APPROACH

This method selects a trusted third party known as a Key Server, which distributes a long-term symmetric key to each ECUs in the vehicle. Then, the transmitting ECU generates a random session key and encrypts this with the long-term symmetric key. The encrypted session key is transmitted to the key server and re-encrypted with the symmetric key, which is pre-shared with the receiving ECU, to share the session key between the sender and receiver. Although the key server can be configured separately, ECUs with high computing power are generally selected, and there may be several depending on the situation. Examples of the application of this method include MaCAN [11], Car2X [12], and Vulcan [13]. These schemes have the disadvantage of a single point of failure such that they are absolutely dependent on the key server.

2) LONG-TERM PRE-SHARD KEY BASED APPROACH

According to this approach, all ECUs in the vehicle share a predefined long-term secret key. In this case, the long-term secret key is set at the install time and stored in the Hardware Security Module (HSM) inside the ECU. Each participant sets session keys with all participants in the communication party or group using the pre-shared long-term key. Examples of these schemes are VatiCAN [14], LCAP [15],

CANAuth [16], and LiBra-CAN [17]. In these schemes, the long-term key is a master key, to which each ECU is required to have physical access whenever a certain ECU is replaced or undergoing maintenance.

B. PUBLIC KEY-BASED KEY DISTRIBUTION

In this case, each ECU in the vehicle owns a public key and a private key, thereby sharing the session key between the sender and the receiver. More specifically, the sender generates a random session key and then encrypts and transmits the random session key using the RSA public key of the receiver. The recipient then decrypts it with its private key and becomes to share the same session key with the sender and the receiver. Alternatively, the Diffie-Hellman algorithm can be used to exchange the public keys between senders and receivers, followed by the use of exponential multiplication with their private keys to calculate the same session key. Examples using this approach include Vatican [14], LCAP [15], [18] and [19]. Since these methods basically require exponential multiplication with a large amount of computation, they are disadvantageous in that they cause severe performance deterioration when applied to an embedded device such as an ECU.

III. BACKGROUND

A. BIVARIATE SECRET SHARING

Since the introduction of the secret sharing scheme [9] by Shamir, various secret sharing schemes have been proposed. In this type of secret sharing scheme, a secret master is divided into n different pieces known as *secret shares*. This secret information can be restored only if $t+1$ of n secret shares are collected again. Conversely, less than $t+1$ secret shares cannot recover secret information. This type of secret sharing scheme is referred to as *Shamir's Secret Sharing (SSS)* in this paper.

In SSS, a polynomial with degree t is generated as follows. In this case, the constant value $f(0) = a_0$ of the polynomial is defined as secret information. Here, $a_i, x < q$ should be satisfied.

$$f(x) = \sum_{i=0}^{t} a_i x^i \pmod q$$

Once the polynomial has been generated, create a secret share S_k by passing the value of id_k, where $k = 0, \ldots, t$, into the polynomial variable x as shown below and distribute it to the user.

$$S_k = f(id_k) \pmod q$$

As mentioned above, secret information is restored only when $t+1$ secret shares are gathered. In order to recover the secret information, the value of $f(0)$ must be obtained, which is achieved in the following manner.

$$f(0) = \sum_{i=0}^{t} S_i \cdot \prod_{\substack{j=0 \\ j\neq i}}^{t} \frac{id_j}{id_j - id_i} \pmod q$$

In this paper, we use the secret sharing scheme based on the bivariate symmetric polynomial introduced in [20], which we refer to as *Bivariate Secret Sharing (BSS)*. The operational principle of BSS is briefly described as follows.

$$f(x, y) = \sum_{i=0}^{t} \sum_{j=0}^{t} a_{ij} x^i y^j \pmod q$$

Similar to the existing SSS scheme, we define $f(0, 0) = a_{00}$, which is a constant term, to be secret information. However, unlike the conventional method, $a_{ij} = a_{ji}$ $(i, j = 0, \ldots, t)$ is set so that the polynomial satisfies the property of symmetry. If the value of user `id`, id_k, is input into variable y of $f(x, y)$, a univariate polynomial is generated, which is termed a *share polynomial*, $S_k(x)$, where $k = 0, \ldots, t$.

$$S_k(x) = f(x, id_k) = \sum_{i=0}^{t} \sum_{j=0}^{t} a_{ij} x^i (id_k)^j \pmod q$$

The secret information $f(0, 0) = a_{00}$ is restored by combining $t+1$ share polynomials in the same manner as in the conventional method. To do this, first define $S_k(0)$ as follows.

$$S_k(0) = f(0, id_k) = \sum_{j=0}^{t} a_{0j} (id_k)^j \pmod q$$

Then, $t+1$ $S_k(0)$ can be expressed as follows.

$$\begin{bmatrix} a_{00} + a_{01} id_0 + \ldots + a_{0t} (id_0)^t = S_0(0) \\ a_{00} + a_{01} id_1 + \ldots + a_{0t} (id_1)^t = S_1(0) \\ \vdots \\ a_{00} + a_{01} id_t + \cdots + a_{0t} (id_t)^t = S_t(0) \end{bmatrix}$$

The above $t+1$ polynomials can be rewritten as the following matrix equation.

$$\begin{bmatrix} \begin{pmatrix} (id_0)^0 & \cdots & (id_0)^t \\ \vdots & \ddots & \vdots \\ (id_t)^0 & \cdots & (id_t)^t \end{pmatrix} \end{bmatrix} \begin{bmatrix} a_{00} \\ \cdots \\ a_{0t} \end{bmatrix} = \begin{bmatrix} S_0(0) \\ \cdots \\ S_t(0) \end{bmatrix}$$

Then, the coefficient including a_{00} can be calculated by applying Gaussian elimination to the inverse matrix as follows. As a result, the secret information $f(0, 0) = a_{00}$ can be restored.

$$\begin{bmatrix} a_{00} \\ \cdots \\ a_{0t} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} (id_0)^0 & \cdots & (id_0)^t \\ \vdots & \ddots & \vdots \\ (id_t)^0 & \cdots & (id_t)^t \end{pmatrix} \end{bmatrix}^{-1} \begin{bmatrix} S_0(0) \\ \cdots \\ S_t(0) \end{bmatrix}$$

On the other hand, when the share polynomial $S_k(x)$ is distributed to each user, the following process for checking the validity of the value is additionally required. When generating $f(x, y)$, we generate the following verification values for each coefficient, which is termed the witness in this paper, where g is a generator belonging to Z_p^*.

$$W_{ij} = g^{a_{ij}} \pmod p$$

The generated witness is used to confirm the validity of $S_k(x)$ as follows.

$$g^{S_k(x)} = \prod_{i=0}^{t}\prod_{j=0}^{t}(W_{ij})^{x^i(id_k)^j} \pmod p$$

In the above equation, $\prod_{j=0}^{t}(W_{ij})^{(id_k)^j}$ is pre-computable because W_{ij} and id_k are known in advance.

B. CODE SPLITTING

Schemes for protecting software core logic include obfuscation [21] and packing [22]. However, these methods, which were previously operating in a PC-level computing environment, cannot be applied to embedded devices such as sensors due to the performance overhead they generate.

In this paper, we propose a scheme to protect the kernel code of smart cars by physically splitting some of the core logic and storing it in external media such as mobile devices. The concept of a code-splitting scheme is as follows.

1) Split part of the binary code.
2) Reconfigure the binary code such that the remainder of the original code is self-contained without the separated part.
3) At the time of execution, combine the separated code and the remainder to restore the original code.

This code-splitting scheme can be applied to securely boot smart cars as follows. First, split a portion of the binary code associated with the partial boot-related code (e.g., 20-64 bytes) and set the split core code to the secret master of the BSS described above.

Subsequently, the share polynomial is calculated and distributed to n devices. At this time, not only the internal sensors but also external mobile devices must necessarily be included in these n devices. Then, $t+1$ ($t < n$) devices, including the mobile device, cooperate to restore the key code of the secret master so that booting is performed. In other words, smart cars can only be booted if they own a trusted mobile device so that hackers can prevent them from starting or controlling the car remotely.

Unlike data-based user authentication schemes using passwords or fingerprints, the proposed scheme does not require a separate cryptographic key or algorithm for authentication such that the authentication is provided based on separate program codes.

C. ROBOT OS

ROS is adopted as the target system software to which the proposed scheme is applied. ROS is an open-source meta-operating system for software development for robots and has become the de-facto standard for robot application development. Unlike the existing OS, distributed data processing between various sensors is very important function in a robot system, and this function can be applied to the sensors of smart cars as well. Thus, most autonomous vehicle prototypes are now equipped with ROS, and this trend is expected to continue in the future.

In ROS, a *node* is the basic unit of execution, and a node replaces the concept of a *process* on the UNIX system. In a typical ROS, one node performs one task. For example, in the case of an automobile, operations such as motor driving, driving control, and routing are implemented using the respective subdivided node groups. In this case, the node comprising the ROS is required to use the API provided by the system in order to perform each task as in the conventional process. Here, ROS client libraries such as `roscpp` and `rospy`, which a node mainly uses, are provided by *ROS Master*.

ROS Master is one of the roscore components, and it provides various infrastructures that are necessary to execute nodes such as a naming service and registration service. Once the node is running, it registers its name, topic name, message type, URI address, and port on the ROS Master. In addition, when requested, it notifies other nodes of the node information registered in the ROS Master to enable communication between nodes. In order for the node to operate properly, the ROS Master must be executed in advance. In this paper, we apply some of this code to achieve secure booting.

IV. PROPOSED SCHEME

A. OVERVIEW

Although various data and user authentication schemes have been proposed, the availability of lightweight and secure authentication technology suitable for smart cars remains insufficient. In this paper, we propose code-based authentication schemes for an ROS-based system using BSS and code splitting. The basic operational principle of the proposed method is illustrated in Fig. 1.

First, the proposed scheme splits a part of the ROS Master code. The ROS Master is a core component that affects the execution of all nodes running in the ROS system. If the ROS Master does not run properly, none of the functions of the ROS system can operate. It is possible to control the entire ROS system by removing only some part of the ROS Master code.

When the code splitting is complete, the split code is divided into share polynomials by using the BSS scheme and distributed to the driver's mobile device and the internal sensors or ECUs of the vehicle. At this step, a plurality of drivers or a plurality of mobile devices may be included.

The share polynomial is used for the authentication of trusted operators and the ROS Master code recovery process. Before the car is booted, a secret share is generated from the share polynomial in the driver device and transmitted to the car.

Once the threshold $t+1$ secret shares have been gathered, the ROS Master code is restored and the system is booted securely. If less than $t+1$ secret shares have been gathered, the ROS system will not boot because the ROS Master code cannot be recovered.

After secure booting with the use of driver authentication, the symmetric key for authentication of the data transmitted between the internal sensor or the mobile device and the

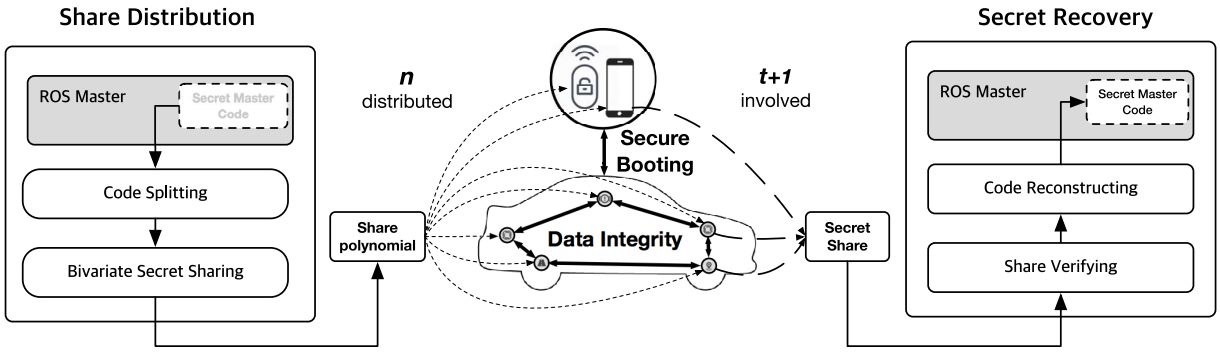

FIGURE 1. Conceptual overview of the proposed scheme.

internal device can be generated by setting the partner device `id` on the share polynomial without exchanging the public key as in the existing Diffie-Hellman exchange. Message authentication is performed based on the generated symmetric key to provide data integrity.

B. DESIGN DETAILS

The proposed scheme consists of initialization, secure booting, and data integrity checking phases.

1) INITIALIZATION

In this phase, the Trusted Dealer (TD) intervenes to initialize the system. TD first generates the system parameters to be used throughout the protocol. For any prime q, a sufficiently large prime number p that satisfies $q \mid p-1$ is generated. We then randomly select a generator g whose order is q and which is an element of Z_p^*.

When the system parameter generation is completed, the TD requests the car for permission to perform the initialization, and accordingly, the car splits a part of the ROS Master code and sends it to the TD. The split code is referred to as the *secret master*. When the secret master is transmitted, the TD generates the following bivariate symmetric polynomial based on this.

$$f(x,y) = \sum_{i=0}^{t}\sum_{j=0}^{t} a_{ij}x^i y^j \pmod q$$

At this time, $f(0,0) = a_{00} = secret\ master$ is set and $a_{ij} = a_{ji}$ is satisfied.

The TD then computes the witness for validation of this polynomial. This witness is distributed to the public repository (e.g., the directory server) or distributed to each device with the shared polynomial

$$W_{ij} = g^{a_{ij}} \pmod p$$

Now, the TD issues a share polynomial to each device by computing the following share polynomial and by passing the device identifier id_k $(k = 0, \ldots, n)$ into the y variable of $f(x,y)$.

$$S_k(x) = f(x, id_k) = \sum_{i=0}^{t}\sum_{j=0}^{t} a_{ij}x^i (id_k)^j \pmod q$$

At this time, the user id, id_0, is assigned to an external device (e.g., a mobile device or physical car key).

2) SECURE BOOTING

The secret master code needs to be restored to ensure that the system boots after initialization. This is achieved by securing the secret share value from $t+1$ or more devices. Furthermore, the boot process should only operate in the presence of the external medium (i.e., id_0) that owns the share polynomial (see Fig. 1). The reason for this is to prevent the car from being started remotely when the car owner is not present.

Algorithm 1 Secure Booting

Input: *secret share Set* $S = S_0(0), \ldots, S_t(0)$
Output: *secret master code*

```
if s_0(0) exists then
    for i := 1 to t do
        receive S_i(0)
    end for
    for i := 1 to t do
        verify S_i(0) with witness
        if s_i(0) is not valid then
            terminate
        end if
    end for
    reconstruct (secret master)
    launch (ROS master)
end if
```

Specifically, suppose the ROS boot system receives the secret share $S_0(0)$ of the external device with id_0 and the secret share $S_1(0), \ldots, S_t(0)$ of the remaining t devices. Then, by using witness, the validity of received secret shares is

verified as follows, where $k = 0, \ldots, t$.

$$g^{S_k(0)} = \prod_{j=0}^{t} (W_{0j})^{(id_k)^j} \pmod{p}$$

When the validation is completed, the secret master a_{00} is reconstructed from $t+1$ secret shares and identifiers, respectively, as shown in the following equation.

$$\begin{bmatrix} a_{00} \\ \cdots \\ a_{0t} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} (id_0)^0 & \cdots & (id_0)^t \\ \vdots & \ddots & \vdots \\ (id_t)^0 & \cdots & (id_t)^t \end{pmatrix} \end{bmatrix}^{-1} \begin{bmatrix} S_0(0) \\ \cdots \\ S_t(0) \end{bmatrix}$$

That is, the secret master can be restored as follows: $a_{00} = [(id_0)^0]^{-1} S_0(0) + [(id_1)^1]^{-1} S_1(0) + \ldots + [(id_t)^t]^{-1} S_t(0)$.

3) DATA INTEGRITY CHECKING

After booting, the car performs all functions entirely depending on the data coming from the various sensors. Thus, the integrity of the data must be guaranteed. This makes it necessary to authenticate the data that are exchanged between internal sensors. For this purpose, existing authentication methods can be utilized. In this paper, we assume that a message authentication code based on a symmetric key is used, as shown in Fig. 2. Here, message (M) means sensing data.

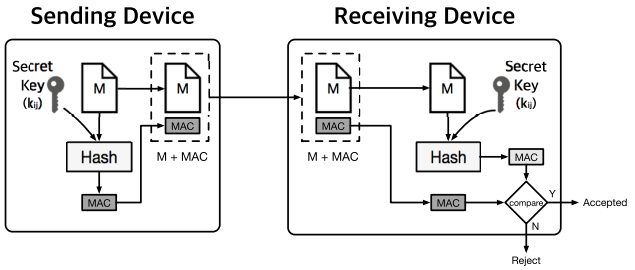

FIGURE 2. Symmetric key-based message authentication mechanism.

The integrity checking of the sensing data requires a pairwise symmetric key between two devices to be present. Generally, an RSA-based session key distribution or Diffie-Hellman key exchange is used for symmetric key sharing. However, the proposed scheme enables the pairwise symmetric key to be calculated immediately with only its own share polynomial and the `id` of the communication counterpart as shown in the following equation where no exponentiation arithmetic and public key exchange is required. For convenience, suppose that the identifier of the sending device is id_i, and the identifier of the receiving device is id_j. The sending device calculates the secret key (K_{ij}) as follows.

$$K_{ij} = S_i(id_j) = f(id_j, id_i) \pmod{q}$$

Similarly, the receiving device calculates the secret key (K_{ji}) as follows.

$$K_{ji} = S_j(id_i) = f(id_i, id_j) \pmod{q}$$

As mentioned earlier $f(x, y) = f(y, x)$ be satisfied. Therefore, the following formula always holds.

$$\begin{aligned} K_{ij} = S_i(id_j) &= f(id_j, id_i) \\ &= f(id_i, id_j) = S_j(id_i) = K_{ji} \pmod{q} \end{aligned}$$

4) DATA CONFIDENTIALITY

CAN offers fast communication speed, low communication cost, and high resistance to noise. However, because it uses bus communication and, as a result of its packet structure, CAN is not secure at all. An attacker can easily intercept a packet and unpack it by participating in the bus communication. The problem can be overcome by using data encryption. This requires the secret key between the transmitter and the receiver to be set first by using the same method as the session key setting method in Section IV. Upon completion of the secret key setting, the sender encrypts the message using this key and sends it to the recipient, who also decrypts the message encrypted with its secret key. At this time, symmetric key encryption such as AES can be used for encrypting the message. AES is known as a very secure encryption algorithm, and some microcontrollers already incorporate high-speed hardware capable of processing this algorithm.

5) DYNAMIC DEVICE ADDITION

The concept of automobiles has been extended to a connected car, which has the characteristics of a living space, and which is able to communicate with its external surroundings. As a result, various devices connected to the automobile are participating, and are joining and leaving the network frequently. This also means that share polynomials must be distributed to new devices participating in the automotive network. The initial nodes can receive the share polynomial via TD, but after the initialization process, TD will no longer store the secret master and safely discard it. In this situation, the new device creates its own share polynomial through the collaboration of existing devices. The principle is similar to the initial generation method of TD. First, $t+1$ existing devices calculate their partial share polynomial with their new device identifier id_{new} in their share poloynomial, $S_k(x)$ and deliver it.

$$S_k(id_{new}) = f(id_{new}, id_k) \pmod{q}$$

Since $S_k(id_{new}) = S_{id_{new}}(k)$, the following expression is established.

$$\begin{bmatrix} \begin{pmatrix} (id_0)^0 & \cdots & (id_0)^t \\ \vdots & \ddots & \vdots \\ (id_t)^0 & \cdots & (id_t)^t \end{pmatrix} \end{bmatrix} \begin{bmatrix} a_{00} \\ \cdots \\ a_{0t} \end{bmatrix} = \begin{bmatrix} S_0(id_{new}) \\ \cdots \\ S_t(id_{new}) \end{bmatrix}$$

Therefore, we can calculate all the coefficients including a_{00} by applying Gaussian elimination, as shown in the background of Section III, and calculate the share polynomial $S_{id_{new}}(k)$ for the new device.

$$S_{id_{new}}(k) = a_{00} + a_{01} id_k + \ldots + a_{0t} (id_k)^t$$

TABLE 1. Feature comparison.

Key Features	Symmetric Key Based Schemes	Public Key Based Schemes		Secret Sharing Based Schemes	
Representative cypto algorithm	AES	RSA	DH	SSS	BSS
Approach	Data-based	Data-based	Data-based	Data-based	Code-based
Authentication medium	Shared key	Digital certificate	Digital certificate	Secret share	Secret share
Secure boot provided	No	No	No	Yes	Yes
Extra communication required	Yes	Yes	Yes	Yes	No
Dynamic device addition provided	No	No	No	Yes	Yes

V. SECURITY ARGUMENTS

A. CODE-BASED USER AUTHENTICATION

An attacker can inject malicious code or packets into an automobile by using an IoT device or a removable medium to disable driver authentication routines such as an immobilizer. The data-based authentication schemes using passwords or specific conditions are vulnerable to authentication-credential duplication [23] or code injection attacks [4], [5]. On the other hand, the proposed scheme splits the core code involved in booting the vehicle. Hence, the system is operated by assembling a pairwise code between the automobile and the external medium at booting time. That is, the correct code pair must be assembled before the driver is authenticated. Moreover, since the authentication data does not exist in the storage medium, the above-mentioned attacks are completely invalid.

B. STEALING SECRET SHARE

An attacker may attempt to steal the share polynomial stored in the car by exploiting the vulnerability of the car. Even if all of the share polynomials in the car are successfully compromised, because only t share polynomials exist in the car, the attacker cannot restore the original secret master. Thus, this type of attack has no impact on the overall system security.

On the other hand, it can be assumed that an attacker would attempt to steal a share polynomial stored in a user device. However, in mobile operating systems such as Android and iOS, various code and data protection technologies effectively protect application resources and functions from unauthorized access. Thus, an attacker aiming to steal a share polynomial stored in a user's device would need to execute a task such as privilege escalation by exploiting vulnerability in the system. Even if such an attack was to succeed, updating the share polynomial using the proactive secret sharing scheme [24] could prevent the theft of the share polynomial or disable the already stolen share polynomial immediately.

C. EAVESDROPPING

The secret shares and sensing data must be transmitted over secure channels. To do this, a pre-procedure is usually required to share the session keys between senders and receivers. In general, RSA-based session key distribution or DH key exchange is applied. In contrast, the proposed scheme generates the symmetric key immediately without the extra key distribution or exchange procedure. This symmetric key enables the secret share and sensing data to be encrypted and protected against eavesdropping.

VI. PERFORMANCE EVALUATION

This section compares the features and performance of the proposed scheme with those of the conventional schemes.

A. FEATURE COMPARISON

The proposed BSS scheme is an authentication scheme that provides code-based user authentication and data integrity. The features of the proposed scheme are analyzed and compared with those of the conventional schemes as shown in Table 1. The comparison covers RSA-based session key distribution (RSA in short), DH key exchange (DH in short), and Shamir Secret Sharing (SSS).

Unlike the existing schemes, the proposed scheme performs user authentication by using a code-based approach based on split code. As mentioned above, it also provides a secure boot functionality, which RSA and DH do not support.

More specifically, in the proposed scheme and SSS, the split core code is distributed to multiple devices such that no single device can know the original contents of the split core code alone. However, since split codes cannot be distributed when the AES or RSA scheme is applied, each device has a split core code in its entirety. Thus, this approach does not meet the goal of this paper to provide secure booting through secure protection of the core code.

In addition, the proposed scheme and the SSS scheme have the advantage that, when a new device is added, the new device can be added to the existing devices without affecting these devices. On the other hand, in the case of RSA and DH, addition of a new device requires a new key pair to be generated for this device and a public key certificate procedure for existing devices. In fact, it would be necessary to update the information of all devices. As with RSA and DH, the SSS scheme requires additional communication such as random shuffling [24] to set up the pairwise key for integrity checking, whereas the proposed scheme does not require such overhead.

B. PERFORMANCE ANALYSIS

We analyze the proposed scheme for complexity in terms of performance. As shown in Table 2, various computations are

TABLE 2. Computational complexity (No. of exponentiation).

Key Features	Symmetric Key Based Schemes	Public Key Based Schemes		Secret Sharing Based Schemes	
Representative cypto algorithm	AES	RSA	DH	SSS	BSS
Secure boot	N/A	N/A	N/A	$(t+1)(t+2)$	$(t+1)(t+2)$
Secure boot (w/ random shuffling)	N/A	N/A	N/A	$(t+1)(t+2)+(t+1)^2$	$(t+1)(t+2)$
Secure boot (w/ random shuffling & precomp.)	N/A	N/A	N/A	$(t+1)+(t+1)^2$	$(t+1)$
Pairwise key setup	0	2	2	$(t+1)$	0

included in the calculation, but the exponentiation operation, which is the most computationally intensive, is mainly analyzed. On the other hand, in the case of secure boot, the RSA and DH-based schemes cannot provide this functionality. Hence, only the SSS and BSS are compared.

The performance of the simple restoration of the secret master code is similar in both schemes. However, in the case of SSS, if the identifiers of $t+1$ devices participating in the Lagrange interpolation are exposed during the partial share calculation, the attacker can learn the share of the participating devices. Thus, random shuffling must be applied to compensate for this. As a result, the amount of computation in the SSS increases as a result of the overhead generated by the random shuffling procedure. Since both the SSS and BSS schemes can pre-calculate a witness, it is also possible to reduce the amount of computation by applying pre-computation.

Next, when comparing the operations of the pairwise key setup process necessary for data integrity between the devices, both RSA and DH require two exponentiation operations. In the case of SSS, the exponentiation operation of $t+1$ is required because the public key of the communication partner is calculated by using the witness. On the other hand, the proposed scheme requires only $t+1$ multiplication operations without exponentiation. Thus, the processing speed is expected to be much faster than all the other schemes. In smart car security, secure boot is a one-time operation in the initial booting process and requires a single amount of computation, whereas the sensing data integrity checking is routinely required during normal operation. Thus, a reduction in the amount of computation to calculate the pairwise key is expected to have a decisive influence on the overall system performance.

C. EXPERIMENTAL SETUP

In the proposed scheme, the smart car prototype was implemented using the Gazebo emulator [25] (see Fig.3) provided by ROS Indigo. The mobile device uses Android (SDK version 23) and the BSS implementation was implemented using OpenSSL library version 1.0.1f [26] in Linux (Ubuntu 14.04).

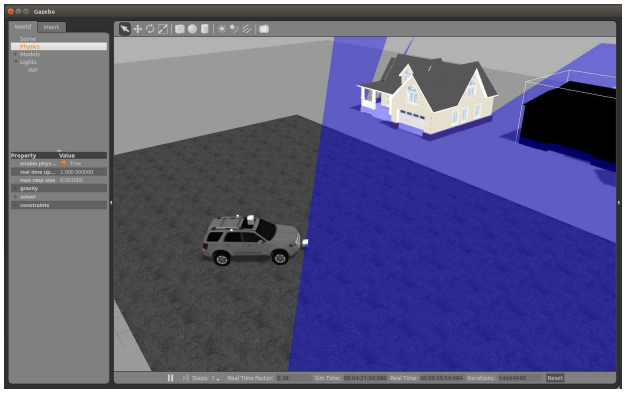

FIGURE 3. Screenshot of Gazebo emulator.

D. EXPERIMENTAL RESULTS

Experiments were conducted to evaluate the computation cost of the secure boot and pairwise key operations.

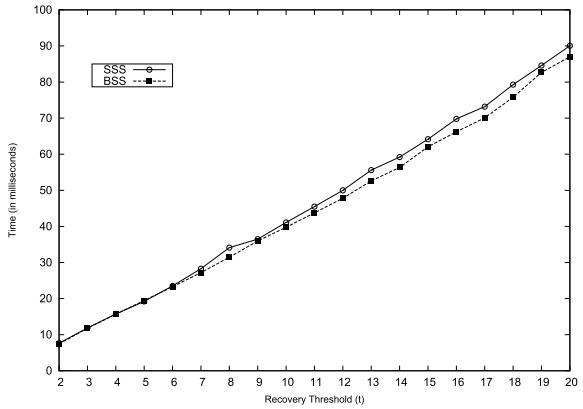

FIGURE 4. Computation time for secure booting.

1) SECURE BOOT

The following experiment implemented three cases of recovery of the secret master code using the BSS and SSS schemes. Fig. 4 shows the result of the experiments on the restoration of the simple secret master code. The time required to recover the secret master code was used as experimental data by varying the threshold (the number of devices required to recover the secret master) from 2 to 20.

In this experiment, neither random shuffling nor witness pre-computation was applied. The experimental results of the two schemes are almost the same for secret master code recovery or the BSS scheme is slightly faster.

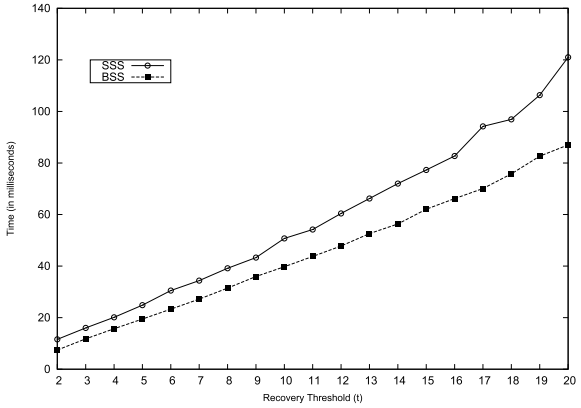

FIGURE 5. Computation time for secure booting with random shuffling.

Fig. 5 shows the result of the secret master recovery experiment where random shuffling was applied to the SSS. The experimental results show that the SSS scheme takes longer to boot, and the gap increases as the value of t increases. This is due to the additional random shuffling cost arising from the process of calculating partial shares in the SSS scheme. On the other hand, because this process is not required in the BSS scheme, fewer operations are required. Thus, the performance improved as shown by the experimental result.

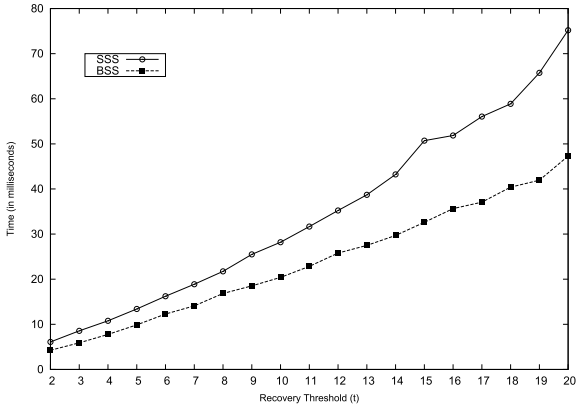

FIGURE 6. Computation time for secure booting with random shuffling and precomputation.

Fig. 6 shows the most realistic scenario in which precomputation of the witness is applied to both the SSS and BSS. The experimental results show that the SSS scheme takes a longer time to boot, and the gap between the two schemes increases as the value of t increases. However, compared with previous experiments, both the SSS and BSS schemes shortened the time required for booting by approximately 30-50 milliseconds.

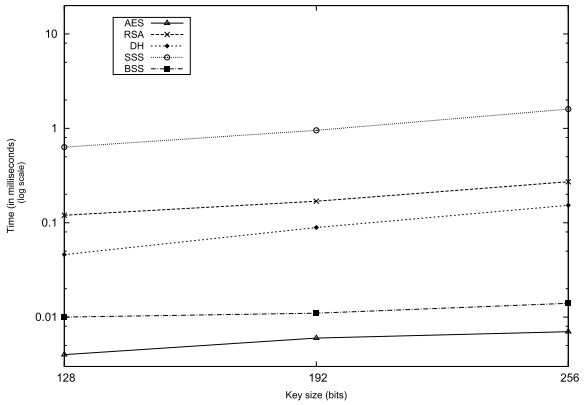

FIGURE 7. Computation time for session key establishment.

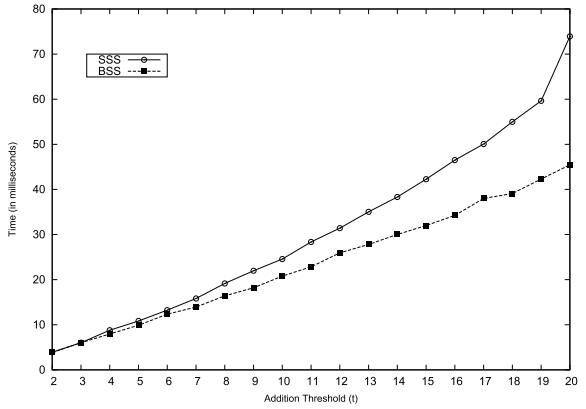

FIGURE 8. Computational time for the dynamic devices addition.

2) SESSION KEY COMPUTATION

Fig. 7 shows the result of calculating the session key using the AES, RSA, DH, SSS, and BSS scheme used in the proposed method. The experimental data used the time required to set the session key when the key size is 128 bytes, 192 bytes, and 256 bytes, respectively. The experimental results show that the SSS scheme takes the longest time to set the session key, followed by RSA, DH, BSS, and AES. This is due to the difference in the exponentiation amount in the process of calculating the session key. In fact, the exponential computation requires $O(t)$ exponentiation in SSS, $O(1)$ in RSA and DH, and 0 exponentiation in BSS and AES schemes. Instead, $O(t)$ multiplication and addition operations were required in the BSS scheme, which resulted in an overwhelming performance advantage over other schemes, although it was somewhat inferior to the AES technique. In fact, when the 128-bit session key is set, the RSA scheme needs approximately 12 times more time than the BSS scheme. In the case of the DH technique, this is approximately 5 times, whereas the SSS technique requires 63 times more time. Conversely, the AES scheme was approximately twice as fast as the BSS scheme.

3) DYNAMIC DEVICES ADDITION

A smart car environment allows new mobile devices or drivers to be dynamically introduced. In this case, the SSS and BSS

schemes enable a new device to be added by itself without the need to update existing devices. Both the SSS and BSS schemes require the coordination of existing devices to add new devices. The following are the results of an experiment to evaluate the time spent when adding new devices in the BSS and SSS schemes, respectively. In this experiment, a threshold of 2-20 was used to assess the time required to issue the secret share and share polynomial to the new device. The experimental results show that the SSS takes longer to add devices, and the gap increases as t increases. These results contain a contribution by the Random Shuffling process, which forms part of the process of adding new devices in SSS. However, BSS was able to omit these processes, thereby requiring less time, as shown in Fig. 8.

VII. CONCLUSION

The hacking of smart vehicles can mainly be attributed to the lack of appropriate integrity checking and encryption scheme for data originating from hundreds of sensors. This is because existing IT security technology cannot be adopted for use in smart cars, which require high-speed and lightweight crypto algorithm. Therefore, in this paper, we proposed a lightweight code-based authentication scheme to provide data encryption and integrity for the smart vehicle environment. The proposed scheme provides secure booting by using bivariate polynomial-based secret sharing and code splitting schemes, and allows only reliable devices to send and receive sensing data.

We showed the applicability of the proposed scheme to the real driving environment by implementing the scheme directly in a vehicle equipped with the ROS platform and conducted the performance evaluation and security analysis. The use of various test cases enabled us to verify that the proposed scheme is superior to existing methods in respect of both security and performance, and it can be confirmed that the proposed scheme can be effectively applied to smart vehicles.

On the other hand, since the proposed scheme was developed for ROS, it can be applied not only to ROS based platforms such as Apollo [27], Voyage [28], and Autoware [29] but also to various mobile platforms such as PolySync [30], MAX [31], and EB robinos [32]. In the future, we expect that the code splitting method will contribute to improving smart car security if applied to core automotive software modules such as ECM (Engine Control Module) and BCM (Body Control Module).

REFERENCES

[1] *Gartner Says 8.4 Billion Connected 'Things' Will Be in Use in 2017, Up 31 Percent From 2016*. Accessed: Nov. 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3598917

[2] *Worldwide Internet of Things Forecast, 2017–2021*. Accessed: Nov. 2017. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US43087717

[3] *Master the Four Stages of Connected-Vehicle Evolution to Lead the Renaissance of the Automobile*. Accessed: Nov. 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3598917

[4] K. Koscher *et al.*, "Experimental security analysis of a modern automobile," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 447–462.

[5] S. Checkoway *et al.*, "Comprehensive experimental analyses of automotive attack surfaces," in *Proc. 20th USENIX Conf. Secur.*, 2011, p. 6.

[6] T. Ring, "Connected cars—The next target for hackers," *Netw. Secur.*, vol. 2015, no. 11, pp. 11–16, 2015.

[7] M. Tatebayashi, N. Matsuzaki, and D. B. Newman, Jr., "Key distribution protocol for digital mobile communication systems," in *Advances in Cryptology—CRYPTO*. New York, NY, USA: Springer, 1989, pp. 324–334.

[8] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[9] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[10] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, vol. 3, no. 2, 2009, p. 5.

[11] O. Hartkopp, C. Reuber, and R. Schilling, "MaCAN-message authenticated CAN," in *Proc. 10th Int. Conf. Embedded Secur. Cars (ESCAR)*, 2012.

[12] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann, "Car2X communication: Securing the last meter—A cost-effective approach for ensuring trust in Car2X applications using in-vehicle symmetric cryptography," in *Proc. IEEE Veh. Technol. Conf.*, Sep. 2011, pp. 1–5.

[13] J. Van Bulck, J. T. Mühlberg, and F. Piessens, "VulCAN: Efficient component authentication and software isolation for automotive control networks," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2017, pp. 225–237.

[14] S. Nürnberger and C. Rossow, "VatiCAN–vetted, authenticated CAN bus," in *Cryptographic Hardware and Embedded Systems*. Berlin, Germany: Springer, 2016, pp. 106–124.

[15] A. Hazem and H. A. H. Fahmy, "LCAP—A lightweight CAN authentication protocol for securing in-vehicle networks," in *Proc. Embedded Secur. Cars Conf.*, vol. 6, 2012, pp. 1–10.

[16] A. Van Herrewege, D. Singelee, and I. Verbauwhede, "CANAuth—A simple, backward compatible broadcast authentication protocol for CAN bus," in *Proc. ECRYPT Workshop Lightweight Cryptogr.*, 2011, pp. 1–7.

[17] B. Groza, S. Murvay, A. Van Herrewege, and I. Verbauwhede, "LiBrA-CAN: A lightweight broadcast authentication protocol for controller area networks," in *Cryptology and Network Security*, vol. 16. Berlin, Germany: Springer, 2012, pp. 185–200.

[18] C. Szilagyi and P. Koopman, "Low cost multicast authentication via validity voting in time-triggered embedded control networks," in *Proc. 5th Workshop Embedded Syst. Secur.*, 2010, Art. no. 10.

[19] C. Szilagy and P. Koopman, "A flexible approach to embedded network multicast authentication," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2008, pp. 165–174.

[20] N. Saxena, G. Tsudik, and J. H. Yi, "Efficient node admission and certificateless secure communication in short-lived MANETs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 2, pp. 158–170, Feb. 2009.

[21] A. Cabutto, P. Falcarin, B. Abrath, B. Coppens, and B. De Sutter, "Software protection with code mobility," in *Proc. ACM Workshop Moving Target Defense*, 2015, pp. 95–103.

[22] X. Ugarte-Pedrero, D. Balzarotti, I. Santos, and P. G. Bringas, "SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers," in *Proc. IEEE Symp. Secur. Privacy*, May 2015, pp. 659–673.

[23] J. Choi, G. Na, and J. H. Yi, "Hardware-assisted credential management scheme for preventing private data analysis from cloning attacks," *Multimedia Tools Appl.*, vol. 75, no. 22, pp. 14833–14848, 2016.

[24] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *Advances in Cryptology—CRYPT*. Berlin, Germany: Springer, 1995, pp. 339–352.

[25] *Gazebo*. Accessed: Nov. 2017. [Online]. Available: http://gazebosim.org

[26] *OpenSSL Cryptography and SSL/TLS Toolkit*. Accessed: Nov. 2017. [Online]. Available: https://www.openssl.org

[27] *Apollo Open Vehicle Certificate Platform*. Accessed: Jul. 2018. [Online]. Available: http://apollo.auto

[28] *Voyage Self-Driving Taxi*. Accessed: Jul. 2018. [Online]. Available: https://voyage.auto

[29] *Autoware Open Source to Self-Driving*. Accessed: Jul. 2018. [Online]. Available: https://autoware.ai

[30] *Polysync Autonomous Vehicle*. Accessed: Jul. 2018. [Online]. Available: https://polysync.io

[31] *Max Platform for Driverless Cars*. Accessed: Jul. 2018. [Online]. Available: https://polysync.io
[32] *Eb Robinos–Automated Driving System*. Accessed: Jul. 2018. [Online]. Available: https://www.elektrobit.com

JOONSANG YOO received the B.S. and M.S. degrees in computer science and engineering from Soongsil University, Seoul, South Korea, in 2016 and 2018, respectively. His research interests include mobile device security, automotive software security, and applied cryptography.

JEONG HYUN YI received the B.S. and M.S. degrees in computer science from Soongsil University, Seoul, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree in information and computer science from the University of California at Irvine, Irvine, CA, USA, in 2005. He was a Principal Researcher at the Samsung Advanced Institute of Technology, South Korea, from 2005 to 2008, and a member of research staff at the Electronics and Telecommunications Research Institute, South Korea, from 1995 to 2001. From 2000 to 2001, he was a Guest Researcher with the National Institute of Standards and Technology, Gaithersburg, MD, USA. He is currently an Associate Professor with the School of Software and the Director of the Cyber Security Research Center at Soongsil University. His research interests include mobile security and privacy, IoT security, and applied cryptography.