sensors

MDPI

Article

Toward a Robust Security Paradigm for Bluetooth Low Energy-Based Smart Objects in the Internet-of-Things

Shi-Cho Cha [1], Kuo-Hui Yeh [2,*] and Jyun-Fu Chen [1]

[1] Department of Information Management, National Taiwan University of Science and Technology, Taipei 10607, Taiwan; csc@cs.ntust.edu.tw (S.-C.C.); D9909302@mail.ntust.edu.tw (J.-F.C.)
[2] Department of Information Management, National Dong Hwa University, Hualien 97401, Taiwan
* Correspondence: khyeh@gms.ndhu.edu.tw; Tel.: +886-3-863-3117

Received: 7 September 2017; Accepted: 11 October 2017; Published: 14 October 2017

Abstract: Bluetooth Low Energy (BLE) has emerged as one of the most promising technologies to enable the Internet-of-Things (IoT) paradigm. In BLE-based IoT applications, e.g., wearables-oriented service applications, the Bluetooth MAC addresses of devices will be swapped for device pairings. The random address technique is adopted to prevent malicious users from tracking the victim's devices with stationary Bluetooth MAC addresses and accordingly the device privacy can be preserved. However, there exists a tradeoff between privacy and security in the random address technique. That is, when device pairing is launched and one device cannot actually identify another one with addresses, it provides an opportunity for malicious users to break the system security via impersonation attacks. Hence, using random addresses may lead to higher security risks. In this study, we point out the potential risk of using random address technique and then present critical security requirements for BLE-based IoT applications. To fulfill the claimed requirements, we present a privacy-aware mechanism, which is based on elliptic curve cryptography, for secure communication and access-control among BLE-based IoT objects. Moreover, to ensure the security of smartphone application associated with BLE-based IoT objects, we construct a Smart Contract-based Investigation Report Management framework (SCIRM) which enables smartphone application users to obtain security inspection reports of BLE-based applications of interest with smart contracts.

Keywords: Bluetooth low energy; internet-of-things; privacy; random address; security

1. Introduction

With the advancement of wireless communications and pervasive computing technologies on smartphones, people can control nearby Internet-of-Things (IoT) devices, e.g., wearable devices or fixed specific-purpose sensors, via their smartphones. Versatile IoT-based service applications have been developed on the smartphone to provide intelligence and convenience to our daily life. Because major smartphone platforms, such as iOS and Android, are equipped with Bluetooth Low Energy (BLE) capabilities and have well-implemented API for BLE communication, more and more IoT devices have adopted BLE technologies to communicate with smartphones. To ensure that users have permissions to access BLE-based IoT devices, the devices usually request users to provide credentials as access tokens for the purpose of authentication and access control. In general, users will request the credentials via an Internet service. In that case, users can obtain credentials with online Internet services, and accordingly the costs for users for installing credentials into their smartphones can be reduced. Alternatively, sellers of the devices will provide physical media, like the PIN code protection envelope provided by banks, to deliver credentials (or key phrases) to users. Nevertheless, there are management issues in protecting the printed out credentials from being stolen by malicious people.

Since privacy has become a critical issue in smartphone applications, current smartphone platforms have started to prohibit smartphone applications from obtaining sensitive information, such as a smartphone's physical identification, which may result in potential individual privacy disclosure. For example, after Android 6.0 [1], if Android applications invoke the BluetoothAdapter.getAddress() method to get the Bluetooth hardware address of a smartphone, the applications will receive a constant value of 02:00:00:00:00:00 instead of the complete address value. Moreover, to prevent user smartphones from being tracked by malicious adversaries using smartphone Bluetooth Media Access Control (MAC) addresses, a useful technique, called BLE random address, is adopted on smartphones [2,3]. That is, the smartphone will adopt different Bluetooth MAC addresses to communicate with nearby BLE-based devices when the smartphone restarts its Bluetooth function. After a device has been paired with another device with the BLE random address scheme, the device will then transfer the original Bluetooth MAC address and associated identification data to another device. After that, when the device picks up a new Bluetooth MAC address during the next session, the linked device can use the information to resolve the new address into the original one. Note that a device may still give random addresses to paired devices rather than its original address for privacy considerations [3]. However, if a device does not provide its original address to paired devices, the device may need to execute the pairing process again.

Although the random address scheme in BLE can prevent BLE-based IoT devices from being tracked by unauthorized parties, once a smartphone has bonded to a device using the random address mechanism, every application installed on this smartphone can track the device. For example, in Figure 1, suppose that a BLE device A adopts the random address scheme to protect its real Bluetooth MAC address. The smartphone X cannot obtain the real Bluetooth MAC address of the device A. For example, assume there exists a smartphone X with the App 1 installed on the smartphone. The user of smartphone X can initiate the pairing process to establish a pairing relationship between the smartphone X and the device A via the App 1. The smartphone X can accordingly obtain the real Bluetooth MAC address of the device A. In order to prevent the App 1 from obtaining the real address of the device A, the smartphone X stores the latest random address of the device A before pairing and provides the stored address to the App 1. Therefore, the App 1 can identify the device A with the stored address. Comparatively, other smartphones aside from X cannot know how to link received Bluetooth MAC addresses to the device A unless there is only one nearby device. Unfortunately, to ensure the effectiveness of random addresses a BLE-based device may not allow simultaneous pairing connections to the device. Consequently, many BLE applications are designed without pairing. This paves a way for malicious adversary to retrieve sensitive data or to launch a user privacy invading attack. As illustrated in Figure 1, the App 2 can also obtain the stored address to identify the device A without any newly invoked pairing process. In our previous work [4], a security weakness based on the above scenario had been identified in a commercial product sold in Taiwan, i.e., a smart motorcycle called Gogoro Smart Scooter [5]. In general, users can control their Gogoro motorcycles through BLE signals sent from the user's smartphone. Based on the reverse-engineering analysis on the application associated with Gogoro motorcycles, in the following we present the communication processes of starting a Gogoro motorcycle. Note that the processes occur between Gogoro motorcycles, smartphones and backend servers.

Each motorcycle has a public Bluetooth address and a secret key stored in its engine control unit. The backend server built by the Gogoro company firstly stores addresses and secret keys of every motorcycle. When a user buys a motorcycle S, the Gogoro company requests the user to provide his/her email and a password as a registration request. Next, the user can install an application on his/her smartphone which will be mainly used for logging into a web server provided by the company. After verifying the user by the provided email and password, the server sends a secret key K_S and a Bluetooth address A_S associated with the motorcycle S to the application. The motorcycles use a customized pairing process to pair with user smartphones: the motorcycles just store the addresses of their owners' smartphones. However, because current smartphones usually use random addresses, the

backend server of the company generates pseudo-addresses for user smartphones. After successfully authenticating against the backend server, the user U can obtains his/her pseudo-address A_U from the backend server and provides the address to his/her motorcycle. When the S wants to use his/her smartphone to control the motorcycle, the smartphone advertises a message with the address of the motorcycle, i.e., A_S, and his/her pseudo-addresse, i.e., A_U. After receiving the message, the motorcycle can detect the presence of the user's smartphone and try to connect to the specific GATT service in the smartphone and send a challenge back to the application. The motorcycle application on the smartphone then retrieves the challenge and generates a response based on the secret key of the motorcycle and the challenge. Finally, the motorcycle judges whether the the response matches the challenge. If the response matches the challenge, the motorcycle executes the previously received command. Nevertheless, the above process has a vulnerability. If malicious people can obtain the pseudo-addresses of motorcycle owners, Bluetooth MAC addresses of motorcycles, and secret keys of the motorcycles, those malicious people can control motorcycles on behalf of motorcycle owners. The pseudo-addresses of smartphones that can control motorcycles and addresses of motorcycle can easily be retrieved because the motorcycle applications advertise the pseudo-addresses of user smartphones and the addresses of motorcycles to notify associated motorcycles to connect to the smartphones. Moreover, based on our analysis, although the application uses SSL to communicate with the Web server of the company, the application does not verify the web server certificate. Therefore, malicious people may intercept the communication between the web server and user applications to obtain secret keys to control associated motorcycles (the company has fixed the issue after being notified by us). Obviously, using pseudo-addresses would not be a good solution to identifying devices using random addresses.

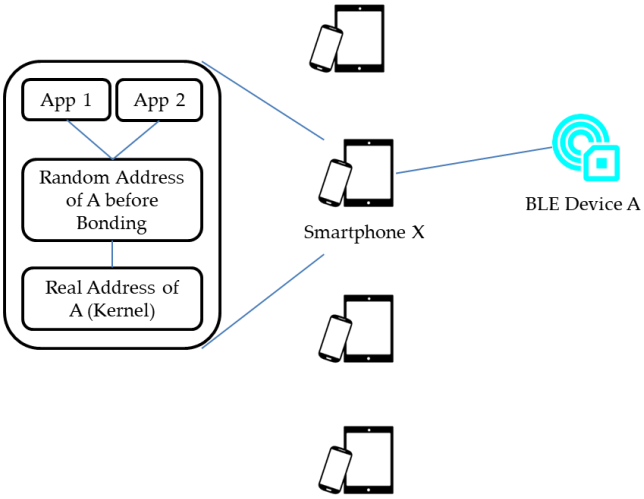

Figure 1. A potential security impact of adopting the random address technique.

Because the pairing process plays an important role in the BLE-based authentication process for preventing Man-In-the-Middle (MITM) attacks and providing communication security, BLE applications without pairing would increase security risks on the smartphone (and to its owner). Therefore, in this study we first present the generalized requirements BLE-based applications associated with IoT devices. From the viewpoint of IoT devices, to pursue the best tradeoff between privacy and security, we demonstrate a privacy-aware access control scheme for BLE-based IoT devices. Since in our scheme BLE-based IoT applications can constrain an authorized person to use a specific smartphone to access IoT devices without revealing the smartphone's physical identification information, the proposed method can contribute to alleviate the tension between security and privacy on BLE-based IoT applications. Moreover, from the viewpoint of a smartphone application associated with the IoT objects, we propose a Smart Contract-based Investigation Report Management (SCIRM) framework

for the security management of BLE-based applications. The SCIRM enables smartphone application users to obtain security inspection reports of BLE-based applications of interest with smart contracts. Benefiting from blockchain technology, users can obtain historical inspection reports of a BLE-based application and verify the integrity of the reports. In addition, SCIRM utilizes smart contract technology to implement the interfaces so that smart contracts will enforce the related actions automatically. The presented framework can enable users to adopt appropriate countermeasures to potential application security risks as users can obtain up-to-date security information about applications in a timely way. In brief, in this study we introduce a robust security paradigm, as shown in Figure 2, for BLE-based smart objects in the IoT in which two viewpoints on IoT device security and smartphone applications security, respectively, are the focus. The contribution of this manuscript is twofold: one is a privacy-aware access control scheme for secure communication between the smartphone and BLE-based IoT devices and the other one is the SCIRM for the security management of BLE-based applications.

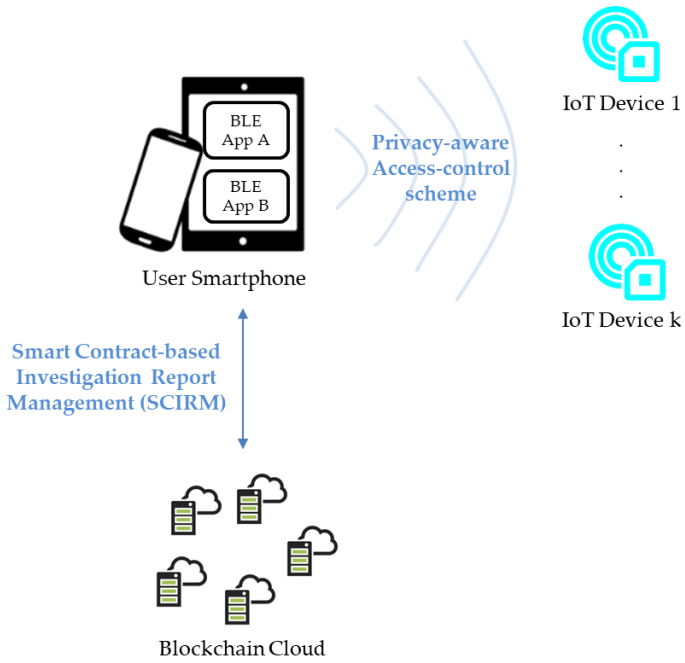

Figure 2. The proposed security paradigm for BLE-based applications associated with IoT devices.

The rest of the paper is organized as follows: Section 2 discusses the related work and Section 3 presents the generalized security requirement of BLE-based applications associated with IoT devices. In Section 4, we introduce a privacy-aware access-control mechanism to fulfill the security criterion we have proposed. After that, the proposed framework, i.e., SCIRM, is presented in Section 5 and, finally, we conclude the paper in Section 6.

2. Related Work

In recent years, researchers have paid a great deal of attention to the development of IoT-based applications with potential security and privacy issues. In 2012, Jara et al. [6] designed a knowledge acquisition and management platform relying on IoT network architecture. The platform focused on the management of personal health, and enabled delivery of healthcare services by virtue of its capabilities to predict health anomalies in real-time and offer feedback to patients. The next year, Berhanu et al. [7] presented an adaptive security process for IoT devices in an e-Health environment, and examined the validation through the study of the impact of antenna orientation on energy consumption. The authors investigated the feasibility of adopting lightweight security solutions

as part of the ASSET infrastructure [8]. In 2014, Torjusen et al. [9] proposed a solution integrating run-time verification enablers in the feedback adaptation loop of the ASSET system, i.e., an adaptive security framework in an IoT based e-Health environment, and implemented the framework with colored Petri Nets. The run-time enablers produce machine-based formal models of a system's status and available context at run-time. Moreover, the authors presented requirements for verification at run-time as formal specifications and introduced dynamic context monitoring and adaptation. Recently, Gope and Hwang [10,11] proposed authentication mechanisms for a distributed IoT-based network architecture and applied the proposed techniques to healthcare management. The protocols are suitable for Body Sensor Networks consisting of health care-oriented sensor nodes. To establish secure communication in an efficient way, lightweight cryptography modules, i.e., one-way hash function and bitwise exclusive-or operation, are adopted to support the authentication process. The authors then investigated the security and performance via BAN logics inference and protocol efficiency analysis.

In view of the security issues of the IoT architecture, Yao et al. [12] presented a lightweight multicast communication scheme on a small scale level of IoT applications. The proposed method is based on the fast one-way accumulator technique [13] which provides high usage flexibility when existing and accumulated items in an application are dynamic. The authors then reconstruct a fast one-way accumulator and designed a lightweight multicast authentication mechanism for small scale IoT applications. Based on the evaluation and analysis, this study showed that the proposed multicast authentication scheme can meet the requirements of resource-constraint applications. In 2015, to achieve security protection among ubiquitous things, Ning et al. [14] proposed an aggregated proof-based hierarchical authentication scheme for layered U2IoT architectures. In the presented scheme, anonymous data transmission, mutual authentication and hierarchical access control are achieved via user authorization, aggregated-proof verifications, homomorphism functions and Chebyshev chaotic maps. The authors claimed that their proposed scheme is suitable for the U2IoT architecture. Later, Hernández-Ramos et al. [15] developed several lightweight authentication and authorization schemes, which are compliant with the Architectural Reference Model (ARM) from the EU FP7 IoT-A project, on constrained smart objects. The proposed schemes can be combined with other standard technologies and form security plans for the lifecycle of IoT devices. The same year, Kawamoto et al. [16] implemented a flexible data collection scheme for retrieving ambient information from intelligent sensors/devices as authentication tokens in a location-based authentication system for industrial IoT applications. In the proposed method, a dynamic parameter adjusting mechanism is automatically performed based on system factors and surrounding environment parameters to pursue higher system accuracy. Next, Cirani et al. [17] presented an integration architecture consisting of HTTP/CoAP services and open authorization (OAuth) services for IoT applications with multiple constrained objects which are limited by their computational power. The authors then conducted a performance evaluation via simulations with Contiki OS-based constrained devices to present the feasibilities and practicability of their proposed architecture. More recently, Cha et al. [4] presented an analysis for examining the security impact of adopting random address technique in BLE-based IoT applications. The authors argued that a one-to-one property must be strictly adopted for credentials and device connection. Afterwards, the authors presented a privacy enhancement solution for BLE-based IoT devices. Fawaz et al. [18] examined more than 200 types of BLE-based devices and discovered these devices cannot fulfill the privacy criteria, i.e., the device's presence may be revealed. Potential threats, such as the revealing of individual sensitive information and even behavior tracking, may thus exist. To conquer the problem, the authors proposed a privacy-aware enhancement system, called BLE-Guardian, which delivers robust privacy protection for users (or environments) equipped with BLE devices. In the proposed system, the users (and administrators) possess the ability to monitor and control those who discover, scan and connect to their devices. Then, Rizzardia et al. [19] added security functionalities to MQTT primitives and accordingly constructed a lightweight data-sharing system. With the support of a policy management scheme, the flow of information in MQTT-enabled

IoT systems can be flexibly controlled based on predefined flexible policies. Furthermore, the authors presented the proposed mechanism as an open source feature under an Apache v.2 license.

3. The Generalized Requirements

This section provides generalized requirements for BLE-based applications associated with IoT devices. As depicted in Figure 3, users use their smartphones to send requests to an Internet server to verify their identities. The server then sends credentials to the user smartphones to enable the smartphones to connect to nearby IoT devices and to control those devices. In that case, the following requirements should be achieved:

- The Internet server must ensure that credentials can only be used by the assigned smartphone.
- An IoT device must be controlled by one smartphone at a time.
- Smartphone users can discover the credentials to control IoT devices which have been used by others even though the IoT devices have no Internet connection abilities.

Table 1 lists possible schemes for IoT devices to identify smartphones. First, after verifying the identities of smartphone users, the Internet server can generate credentials based on the identification information of smartphones [20]. Therefore, an IoT device can restrict access so only the smartphone with the specified identification information can use associated credentials to connect to the device. Similarly, smartphone applications can use smartphone identification information to encrypt credentials for IoT device communication. Then, smartphone applications need to use local identification information to obtain the credentials. If people copy the encrypted credentials to other smartphones, the smartphone applications cannot obtain credentials with incorrect identification information. One of the most critical deficiencies of the above schemes is that smartphone applications may have trouble obtaining the real identification information of the located smartphones.

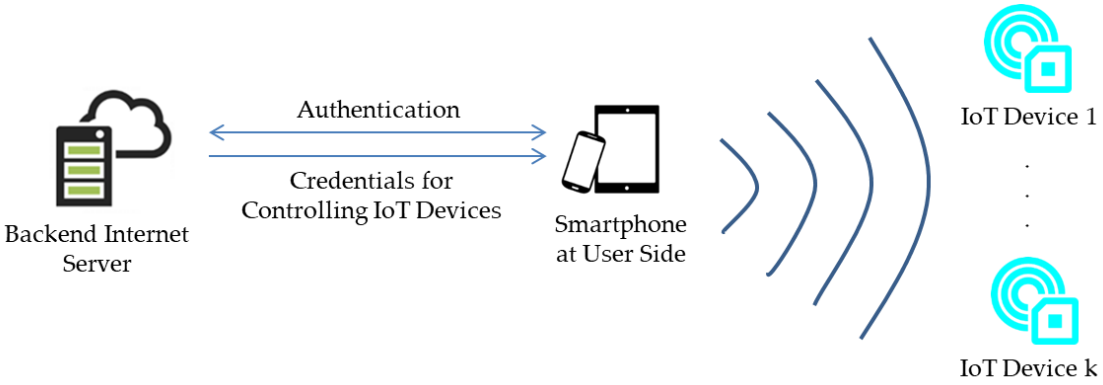

Figure 3. Generalized model of BLE-based applications associated with IoT devices.

Table 1. List of Schemes to Pair Smartphones to Devices.

Scheme	Deficiencies
Using ID-based keys	Difficulty to obtain identity information. Identity stealing and faking.
Encrypting credentials with identities	
Storing credentials in SE	Not every smartphones have built-in SEs
OTP using out-of-band channels	Extra communication costs. Smartphones may not have SMS capability.
Encrypting credentials and pseudo-identities with user pins	Users may forget their pins

Malicious users may obtain identification information and associated credentials if smartphone owners jailbreak their smartphones and accidentally install malware on the smartphones. Then, the malicious users can adopt faked identification information to access IoT devices. To address the issue, smartphone users may store credentials in the security elements (SEs) of their smartphones and only perform operations regarding credentials in the SEs. Obviously, the biggest limitation of the scheme is that not every smartphone has built-in SEs. Moreover, when users wish to communicate with IoT devices, the users can request remote servers to send one-time-passwords (OTPs) to user smartphones via SMS messages. However, users may not be able to receive SMSs if their smartphones do not install SIM cards. Also, users may need to pay extra costs to receive SMS messages. Even when the servers send OTPs to users through e-mails rather than SMS messages, the IoT devices need to synchronize with the servers.

Finally, after servers authenticate the users, the servers can generate pseudo-identities for the users and send the pseudo-identities along with associated credentials to users. The smartphone applications can request the users to input PINs and use the PINs to encrypt the data. Then, users can input PINs into smartphone applications to obtain the original pseudo-identities and credentials to communicate with IoT devices. In this case, if malicious users obtain data encrypted with the PINs of others, the malicious users cannot obtain correct pseudo-identities along with associated credentials to connect to IoT devices. One major challenge of the scheme is to deal with the situations that users forget their PINs. Users may further store encrypted PINs to remote servers to address the issue.

4. A Privacy-Aware Access-Control Mechanism for BLE-Based Smart Objects

In this section, we present a privacy-aware access-control scheme for enabling BLE-based IoT devices. Let the notation E/E_p denotes an elliptic curve E over a prime finite field E_p, defined by an equation: $y^2 = x^3 + ax + b$, where $a,\ b \in F_p$ are constants such that $\Delta = 4a^3 + 27b^2 \neq 0$. All points $P_i = (x_i, y_i)$ on E and the infinity point O form a cyclic group G under the operation of point addition $R = P + Q$ defined based on the chord-and-tangent rule. In addition, we define $t \cdot P = P + P + \ldots + P$ (t times) as scalar multiplication, where P is a generator of G with order n.

- Setup: server X generates a group G of elliptic curve points with prime order n and determines a generator P of G. Then, X chooses a master key $s \in Z_n^*$ and a secure hash function $H : \{0,1\}^* \times G \rightarrow Z_q^*$. Next, X calculates a master public key $PK_X = s \cdot P$, and publishes system parameters, i.e., $(G,\ P, PK_X, H)$. On the other hand, given *params*, the smartphone S picks a random number $x_S \in Z_n^*$ as it's own secret value and computes $PK_S = x_S \cdot P$ as the corresponding public key. During system initialization, a set of system parameters, i.e., $(G,\ P, PK_X, PK_S, H)$, are installed into each IoT device associated with server X.
- Verification (Figure 4): first, an authentication between smartphone S and server X is performed. After authenticating the user, the server generates a pseudo-identity ID_S for the user. Given *params*, s and the identity ID_S of smartphone S, server X generates a random number $r_x \in Z_n^*$, and calculates $R_x = r_x \cdot P$, $h_x = H(R_x, PK_S, ID_S, PK_X)$ and $s_x = r_x \cdot ID_S + h_x \cdot s \bmod n$. Then, server X returns (ID_S, s_x, R_x) to the smartphone S which soon forwards it to the device D. Then, device D checks the validity of (ID_S, s_x, R_x) via whether the equation $s_x \cdot P = R_x \cdot ID_S + h_x \cdot PK_x \bmod n$ holds or not. The correctness of (ID_S, s_x, R_x) is presented as follows: $s_x \cdot P = (r_x \cdot ID_S + h_x \cdot s) \cdot P = r_x \cdot ID_S \cdot P + h_x \cdot s \cdot P = R_x \cdot ID_S + h_x \cdot PK_x$. If the validity of (ID_S, s_x, R_x) holds, device D can be operated and accessed by smartphone S (and server X) and sends a success command as a response to smartphone S.

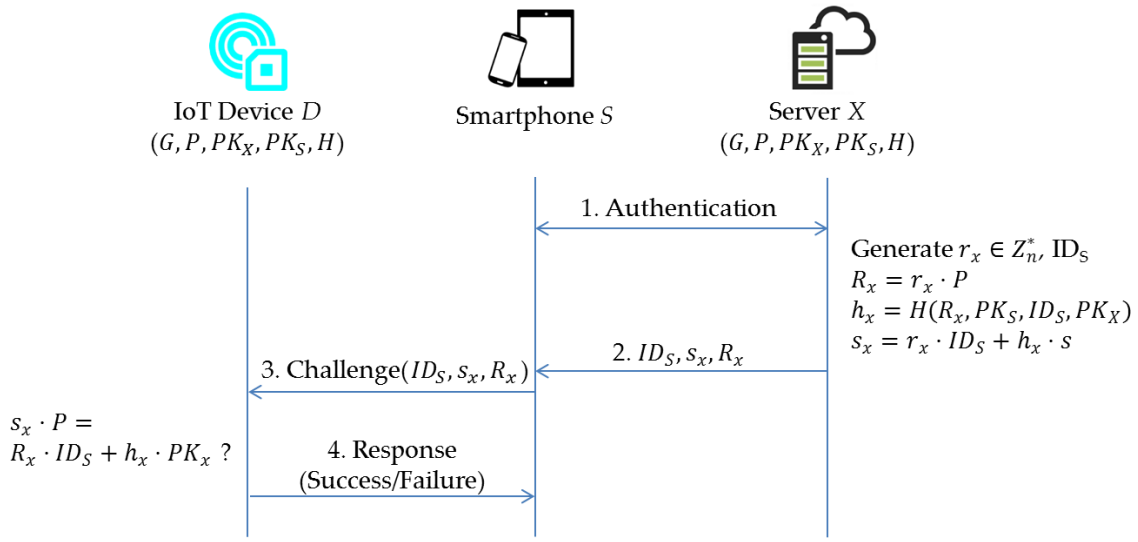

Figure 4. The communication processes of the proposed ECC-based scheme.

4.1. Security Analysis of the Proposed Approach

In this study, we assume that adversary *Adv* models an outside adversary who is able to replace any entity's public key with specific values chosen by the adversary itself; however, the adversary *Adv* does not know the secret value of Server *X*.

Game 1: The following process is performed between a challenger *C* and an adversary *Adv* during the proposed ECC-based approach between IoT device *D* and server *X*. In Initialization phase, *C* generates s, x_S and system parameters, i.e., $(G,\ P, PK_X, PK_S, H)$. Next, *C* sends $(G,\ P, PK_X, PK_S, H)$ to the adversary *Adv*. Next, in Query phase, the adversary *Adv* is able to adaptively issue the following oracle queries, i.e., *RequestPublicKey*(ID_t) and *ReplacePublicKey*(ID_t, PK_t, $PK_t{}^{\#}$) to *C*, where t may be IoT device *D* or server *X*. Finally, in Output phase the adversary *Adv* will output (ID_t, s_t, R_t); and, if $true \leftarrow Verify(params, ID_S, s_x, R_x)$, the adversary *Adv* wins in Game 1.

- *RequestPublicKey*(ID_t): The oracle takes as input a query (ID_t), where ID_t is the party t's identity. It browses the list *L* and returns the party t's public key PK_t.
- *ReplacePublicKey*(ID_t, PK_t, $PK_t{}^{\#}$): The oracle takes as input a query (ID_t, PK_t, $PK_t{}^{\#}$), where ID_t is the party t's identity. This oracle replaces the party t's public key with $PK_t{}^{\#}$ and updates the corresponding information in the list *L*.

Definition 1. *The proposed ECC-based mechanism between IoT device D and server X is secure against malicious adversaries, if for any polynomial adversary Adv, $Succ_j$ is negligible, where $Succ_j$ is the success probability that Adv wins in Game 1.*

Definition 2. *(Elliptic Curve Discrete Logarithm Problem; ECDLP). Given a group G of elliptic curve points with prime order n, a generator P of G and a point $x \cdot P$, it is computationally infeasible to derive x, where $x \in Z_n^*$.*

Theorem 1. *The proposed ECC-based scheme is existentially secure against malicious adversary in the random oracle model, assuming the hardness of solving ECDLP. If there exist a polynomial-time (pt, q_H, q_{pk}, q_{rpk}) adversary α which can submit at most q_H queries to the oracle Hash(.), q_{pk} queries to the oracle RequestPublicKey(ID_t) and q_{rpk} queries to the oracle ReplacePublicKey(ID_t, PK_t, $PK_t{}^{\#}$), and $Succ_\alpha$ is negligible,*

where $Succ_\alpha$ is the success probability that α wins in game 1, then there exists another algorithm β which can solve a random instance of ECDLP in polynomial time with success probability:

$$Succ_\beta \geq \frac{Succ_\alpha}{q_H \times q_{pk} \times q_{rpk}}$$

Proof. Let α be a polynomial-time adversary that breaks the proposed ECC-based scheme with non-negligible advantage $Succ_\alpha$. The goal of this proof is to build a polynomial-time algorithm β which uses α to solve ECDLP. That is, given a random instance $(P,\ Q = x \cdot P)$, it derives the secret x. In Initialization phase, β picks an identity ID^* as the challenged identity in game 1, sets $Q = PK_X$ and sends public system parameters $(G,\ P, PK_X, PK_S, H)$ to α. In Query phase, α adaptively issue the following oracle queries to β, and each query is unique.

- *Hash* query: For each query, β maintains a list $List_H$ storing $<R_t, PK_S, ID_S, PK_t, h_t>$. Upon receiving an *Hash* query for some $< R_t, PK_S, ID_S, PK_t,\ h_t >$ from α, β checks the $List_H$ and returns h_t to α via the following steps:

  (1) If $<R_t, PK_S, ID_S, PK_t,\ h_t>$ exists in $List_H$, β directly returns h_t to α.
  (2) Otherwise, it chooses a random value $h_t \in Z_n^*$, adds $<R_t, PK_S, ID_S, PK_t,\ h_t>$ into $List_H$, and returns h_t to α.

- *RequestPublicKey*(ID_t): Upon receiving a query with an identity ID_t from α, β performs the following steps:

  (1) If $ID_t \neq ID^*$, β generates three random numbers $a_t, b_t, x_t \in Z_n^*$, and performs $R_t \cdot ID_S \leftarrow a_t \cdot P - b_t \cdot PK_t$, $h_t \leftarrow b_t$, $s_t \leftarrow a_t$ and $PK_t = x_t \cdot P$. Then, β adds $<h_t, ID_S,\ R_t>$ and $<ID_S, s_t, R_t, PK_t, x_t>$ to the lists $List_H$ and $List_K$, respectively. Finally, β returns PK_t to α.
  (2) Otherwise, β generates three random numbers $a_t, b_t, x_t \in Z_n^*$, and sets $ID_t \cdot R_t \leftarrow a_t \cdot P$, $h_t \leftarrow b_t$, $s_t \leftarrow \perp$ and $PK_t = x_t \cdot P$. Then, β adds $<h_t, ID_S, R_t>$ and $<ID_S, \perp,\ R_t, PK_t,\ x_t>$ to the lists $List_H$ and $List_K$, respectively. Finally, β returns PK_t to α.

- *ReplacePublicKey*(ID_t, PK_t, $PK_t^{\#}$): Once β receives a query for some (ID_t, PK_t, $PK_t^{\#}$) from α, β performs the following steps:

  (1) β looks for $< ID_S, s_t, R_t, PK_t, x_t >$ in the list $List_K$. If there exists such a record, β sets PK_t = $PK_t^{\#}$ and $x_t = \perp$.
  (2) Otherwise, β simulate the *RequestPublicKey*(ID_t) query for the identity ID_t and sets PK_t=$PK_t^{\#}$ and $x_t = \perp$.

In the final phase, α successfully outputs (ID_S, s_t, R_t) for the target ID^* with non-negligible advantage $Succ_\alpha$. Based on the forking lemma [21], if we have the polynomial replay of β with the same random tape and different choices of hash oracle, α is able to output another valid signature. Eventually, we will have two valid verification, i.e., $\left(ID_S, s_t, {R_t}^{(j)}\right)$, satisfying the following equations ${s_t}^{(j)} = r_t \cdot {ID_S}^{(j)} + {h_t}^{(j)} \cdot s \bmod n$, where j = 1 and 2. Now, β can derive the two unknown values r_t and s, and outputs s as the solution of the random instance $(P,\ Q = s \cdot P)$ of ECDLP. Next, we analyze β's success probability $Succ_\beta$ of winning in game 1. Note that Event 1 (E_1) denotes that α can forge a valid verification message, i.e., (ID_S, s_t, R_t), while Event 2 (E_2) represents that the output (ID_S, s_t, R_t) satisfies $ID_t = ID^*$. It is calculated that $\Pr[E_1] \geq Succ_\alpha$, $\Pr[E_2|E_1] \geq \frac{1}{q_H \times q_{pk} \times q_{rpk}}$ and $Succ_\beta = \Pr[E_1 \wedge E_2] = \Pr[E_1]\Pr[E_2|E_1] \geq \frac{Succ_\alpha}{q_H \times q_{pk} \times q_{rpk}}$, where q_H is the number of *Hash* query, q_{pk} is the number of *RequestPublicKey* query and q_{rpk} is the number of *ReplacePublicKey* query. Hence, the algorithm β can solve ECDLP with, at minimum, the advantage $\frac{Succ_\alpha}{q_H \times q_{pk} \times q_{rpk}}$, where

q_H denotes the maximum number of queries to *Hash*, q_{pk} denotes the maximum number of queries to *RequestPublicKey*(ID_t) and q_{rpk} denotes the maximum number of queries to *ReplacePublicKey*(ID_t, PK_t, $PK_t^{\#}$). That contradicts the hardness of solving ECDLP. □

4.2. Performance Analysis of the Proposed Approach

To evaluate the performance of the proposed approach, we adopt an IoT-based testbed, i.e., Raspberry PI 2, as the implementation platform in the experiment. The Raspberry PI platform is a card-sized single-board computer which offers an ARM GNU/Linux kernel. In general, the Raspberry PI platform is simulated as an IoT-based smart object. The implementation environment is outlined in Table 2. In addition, in the experiment the elliptic curve is with a 384-bit prime n and the random number generator is with 96-bit output sequence, while SHA-3 (512-bit) is used as the one-way hash function. From Table 3, we can see that around 1301 ms is required in terms of the execution time for the required computations at BLE-based IoT devices. Note that in the experiment, the smartphone performs only the job of message forwarding, while the server is responsible for the signature generation. Obviously, both of them will not be performance bottlenecks when performing our proposed privacy-aware access-control scheme. Therefore, we only investigate the performance at the IoT device end in which the signature verification is invoked.

Table 2. Implementation Environment.

Environment	Description
IoT Platform	Raspberry PI 2: Broadcom BCM2836 @ 1 GHz Quad-Core ARM Cortex-A7 Architecture, 1 GB DDR2 RAM, SanDisk 16 GB Class 10 SD Card
Programming Language	Eclipse 3.8 with Oracle Java 8
Crypto API	The Bouncy Castle Crypto APIs [22]

Table 3. Experiment Results at BLE-based IoT Devices.

Computation Cost	Execution Time
Compute $H(R_x, PK_x, ID_x, PK_x)$	1.017 ms
Verify $s_x \cdot P = R_x \cdot ID_S + h_x \cdot PK_x$	0.284 ms

5. Smart Contract-Based Investigation Report Management Framework (SCIRM)

To help application developers develop secure smartphone applications, several countries and organizations such as the European Union Agency for Network and Information Security (ENISA) [23] and the Taiwan Industrial Development Bureau (IDB) [24], have developed guidelines for smartphone application security. However, when users download smartphone applications from marketplaces, users usually cannot know whether the developers of the applications have followed the guidelines to develop the applications. Therefore, organizations such as US National Institute of Standards and Technology (NIST) [25] and the Open Web Application Security Project (OWASP) [26] have developed smartphone application verification guidelines as well as security requirements for smartphone applications. Hence, smartphone application developers can delegate third parties to follow the verification guidelines to check whether their applications satisfy security requirements of the verification guidelines. Furthermore, organization or government agencies may develop certification programs for smartphone applications. For example, the Taiwan IDB has announced the self-regulatory mobile application security certification program [27]. Smartphone application developers can appoint accredited inspection laboratories to inspect whether their applications satisfy applicable security requirements. Consequently, users can decide to only install inspected applications to reduce security risks of using smartphone applications. In light of this, we propose a Smart Contract-based Investigation Report Management (SCIRM) framework for security evaluation of

BLE-based applications associated with IoT devices. The framework provides not only ontologies to store verification reports about smartphone applications in a bitcoin-like blockchain, but also standard interfaces based on smart contracts for application verifiers to upload inspection reports to the blockchain. The users are notified once the inspection results of a smartphone application have changed. As a result, the proposed framework can enable users to obtain the latest security information of an application timely. Moreover, users can also obtain historical inspection results on applications by versions, developers, and other application profiles. The proposed framework can also contribute to providing users more information to evaluate application security risks.

Figure 5 provides an overview of the proposed framework. As depicted in Figure 5, the framework contains two views: the conceptual view and the implementation view. The conceptual view is composed of four major standards:

- The Specification of Inspection Report defines the components of an inspection report as well as the format of each component. In general, an inspection report includes a list of inspection results. Each inspection result is for a security objective. An inspector can describe findings about whether an application comply with a security objective in an inspection result.
- The Specification of Report Storage clarifies how to store an inspection report in the blockchain.
- The Trusted Verifier Schema Specification provides properties required to describe a trusted verifier. In general, users may not know the trustworthy of an inspector and the inspection reports it issued. This study assumes that a notary can provide identities and associated public keys for well-known inspectors. For example, Taiwan has the Accreditation Program for Mobile Application Basic Security Evaluation Laboratories [28]. The Taiwan Accreditation Foundation lists information about the accredited laboratories on its Web pages. Therefore, we can simply append blockchain account identities and associated public keys of the laboratories.
- The standard API and the associated smart contract for report management.

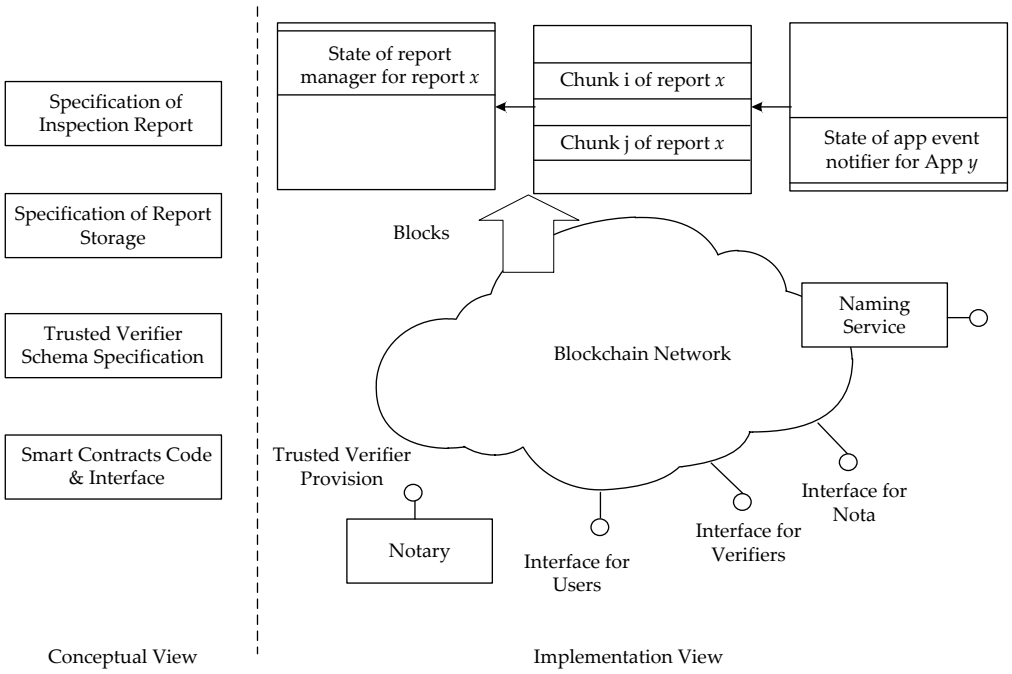

Figure 5. The overview of the proposed SCIRM framework.

The implementation view illustrates the physical components of the framework. We deploy the framework on the Ethereum blockchain system [29]. In the Ethereum blockchain system, people can enclose data in transactions. The blockchain system encapsulates the transactions in blocks. Due to the size limits of a transaction, this study requests a verifier to split an inspection report into several chunks and enclose the chunks into different transactions. The report manager is to manage the inspection reports. When a verifier submits an inspection report to the blockchain, the verifier obtains the compiled byte code of the report manager and creates a new instance of the report manager smart contract. The verifier can then bind the contract to the transactions of the inspection report. As depicted in Figure 5, the Ethereum blockchain system stores the state of each contract in blocks. After a verifier has created a report manager smart contract for an inspection report, people can obtain the state of the contract from a block in the blockchain system and use the state information to find associated transactions storing the chunks of the report. In addition, we define the application event notifier smart contract. A notary can create an application event notifier smart contract for an application. Verifiers can then link the report manager smart contract of the application to the application event notifier smart contract. Therefore, users only need to listen to the application event notifier smart contract to obtain events that an inspection report of the application has been created or obsoleted.

In addition to the constructor, a report manager smart contract provides interfaces for users and verifiers to access information of the associated report. Simply speaking, the report manager smart contract of an inspection report enables users to obtain the contents and status (e.g., active, obsolete, or revoked) of the report. Moreover, report manager smart contracts can trigger events to notify users the status changes of reports. On the other hand, a verifier can update the status of a report and append other reports to the report through the related smart contract. Although users can obtain all inspection reports and related information of smartphone applications by traversing through every block in the blockchain, finding transactions and contract states of a specific smartphone application could be very time-consuming. Therefore, this study proposes to deploy some supernodes for building indexes on transaction attributes and contract states. The supernodes can provide naming services for users to lookup inspection re-ports on a specific smartphone application or reports on applications developed by a specific developer. Finally, in current Ethereum blockchain systems, people only know a transaction or a smart contract is submitted by an identity. Consequently, users cannot determine whether they should trust inspection reports issued by an identity. To address the issue, this study assumes that notaries can provide a list of trusted verifiers as well as information of the verifiers. The notaries may also process complaints of application developers and mark an inspection report as revoked. However, enabling notaries to revoke inspection reports may induce disputes between notaries and verifiers. We leave the report revocation mechanism as our future work.

Before introducing the report management smart contract, this study will first introduce the application event notifier smart contract. Figure 6 illustrates the abstract interface of an application event notifier smart contract. An application event notifier smart contract contains profiles of a smartphone application. In addition, it provides the *reportStatusChanged* interface for the associated report manger smart contract to inform the application event notifier smart contract that the creator of the report manger has just updated the state of the related inspection report. Consequently, interested people can obtain update events for an inspection report. In this case, SCIRM provides the following types of update events for an inspection report:

SCIRM provides the following types of update events for an inspection report:

- Creation of the inspection report.
- Obsoleting of the inspection report.
- Being extended by another report.

```
contract AppEventNotifier {
    address public creator;
    string public appID;
    string public version;
    string public platform;
    string public developerInfo;
    uint public status;
    event statusChangeNotification(string appID, string version, string platform, uint type,
        address rmAddr)
    function AppEventNotifier(string _appID, string _version, string _platform,
            string _developerInfo) {
        ...
    }
    function reportStatusChanged(uint type) {
        ...
    }
}
```

Figure 6. Abstract of the application event notifier smart contract.

Note that because contents of transactions are immutable in blockchains, update operations to an inspection report are not provided. The abstract interface of a report manager smart contract is depicted in Figure 7. In addition to the profiles of the associated application, a report manager smart contract contains the status of an inspection report, total chunks of the report, and the addresses of the chunks. As depicted in Figure 8, when a verifier wishes to upload an inspection report for an application, the verifier will first split the report into several chunks and submit transactions containing the chunks to the blockchain network. The verifier then generates a report manager smart contract with application profiles, total chunk number of the report, and the address of the applications event notifier smart contract.

```
contract ReportManager {
    address public verifier;
    string public appID;
    string public version;
    string public platform;
    string public developer;
    uint public status;
    uint public totalChunks;
    AppEventNotifier appNotifer;
    struct crunkIndex {
        uint seq;
        string txAddr;
    }
    mapping(uint=>crunkIndex) content;
    mapping(uint=>address) extendedBy;
    function ReportManager(string _appID, string _version, string _platform,
            string _developer, unit _totalChunks, AppEventNotifier _notifer) {
        ...
    }
    modifier onlyCreator {if (msg.sender == verifier) _ }
    function setChunkInfo(uint _seq, address _txAddress) onlyCreator {
        ...
    }
    function extended(address _reportManager) onlyCreator {
        ...
    }
    function obsoleteReport() onlyCreator {
        ...
    }
    function queryChunkAddress(uint _seqNumber)
        returns (address chunkAddress) {
        ...
    }
}
```

Figure 7. Abstract of the report manager smart contract.

Because the current Ethereum platform only allows using fixed size array as function parameters, the verifier needs to use the *setChunkInfo* function to provide associated transaction addresses one by one. The report manger smart contract will store the addresses in a mapping object. After receiving addresses to access every chunk of the associated report, the report manger smart contract will invoke the *reportStatusChanged* function of related application event notifier. The notifier can then trigger an event to notify interested users of the creation of the report. The verifier can also use the *obsoleteReport* function to obsolete a report, or the *extended* function to represent the situation that the report is extended by another report. The report manger smart contract will also request the associated application event notifier smart contract to notify users when these two functions are called.

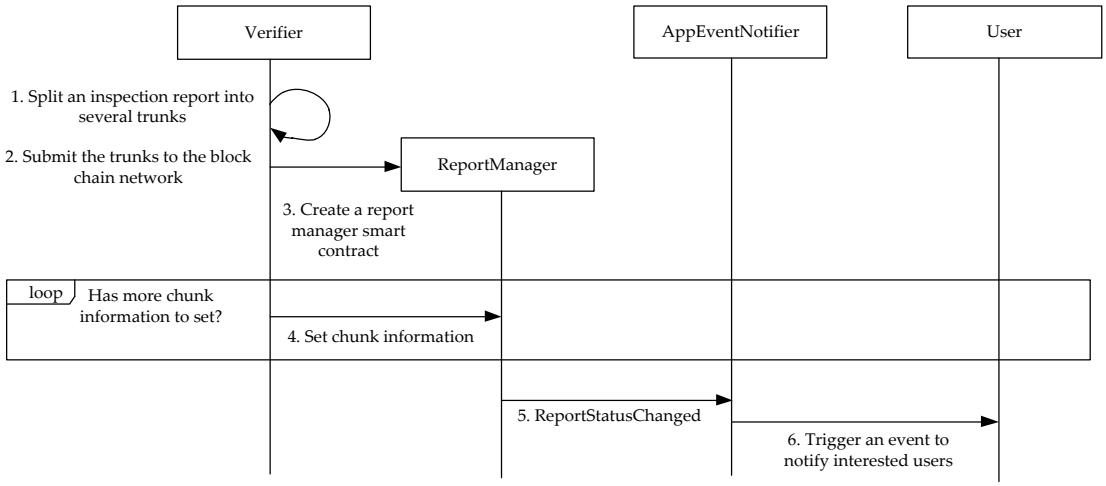

Figure 8. The report creation process.

Figure 9 illustrates the process for a user to retrieve inspection reports of an application from a blockchain. When a user downloads a smartphone application from a marketplace, the user can first query the naming service to obtain the address of the *AppEventNotifier* smart contract of the application as well as the addresses of associated *ReportManager* smart contracts. Then, for each *ReportManager* smart contract, the user can obtain the status of the associated inspection report and the verifier to determine whether or not to download the report. If the user decides to download the report, the user can reassemble the report by collecting all the chunks stored in the transactions. Finally, the user can listen to the *AppEventNotifier* smart contract of the application for status changing events of the application inspection reports.

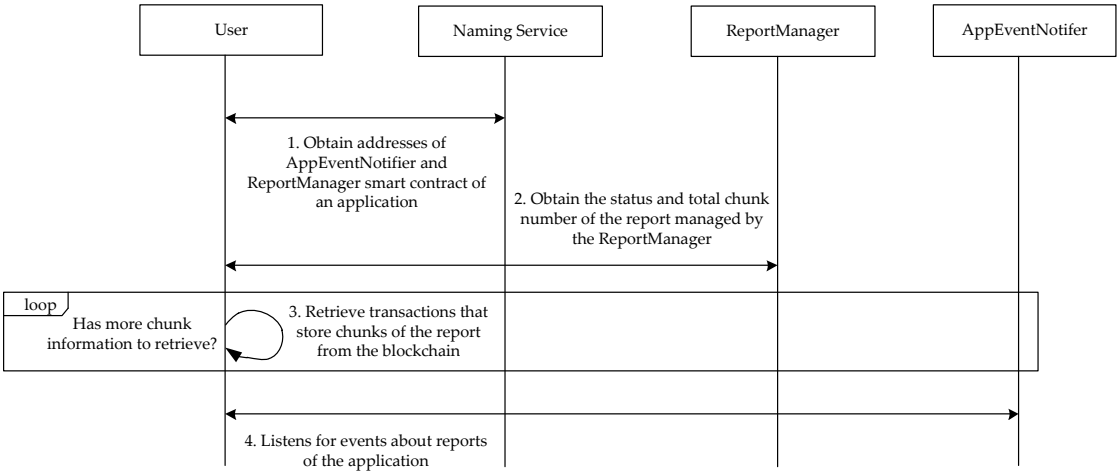

Figure 9. The report query process.

5.1. Performance Analysis for SCIRM

To prove the concept of the proposed framework, we have implemented a prototype system and performed simulation experiment. As shown in Figure 10, we use Node.js to implement a Report Upload Server to handle inspection report uploads. In addition, this study launches a private Ethereum network with a single service node. The report manager communicates with the node via the Ethereum JavaScript API. After receiving an inspection report, the report upload server will split received reports into chucks (Step 1), store the chunks in transactions (Step 2), and send the transaction to the

node (Step 3). Next, the report upload server creates a report manager smart contract and sends the addresses to the contract (Step 4).

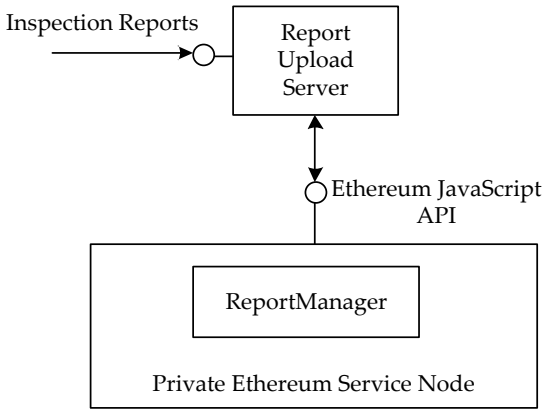

Figure 10. The experimental scenario.

In the experiment, the report manager and the Ethereum service node are both executed on a desktop equipped with an Intel Core i7-4790 CPU 3.60 GHz CPU and 12 GB RAM running Ubuntu 16.04 LTS. To simplify the experimental environment, this study fixes the *mining difficulty* to 0x4000. Note that the *mining difficulty* is a parameter of the Ethereum and other blockchain platforms. In the blockchain platforms, each participant competes to win the right to generate a data block and store the block in associated blockchains. In this case, a participant tries to find a nonce that the participant can use the nonce to generate a hash value matching specific patterns with the nonce and the data block to be stored. The higher the *mining difficulty*, the more calculating efforts a participant need to perform. In addition, the competition winner can usually receive a specified amount of digital currency as incentives to maintain the platforms continuously. As advances of information technologies, the blockchain platforms usually increase the *mining difficulty* to control the total amount of associated digital currencies. However, increasing the *mining difficulty* may influence the experimental results. Therefore, we fix the *mining difficulty* in our experiment to discuss the differences of using different blocksize.

An inspection report is about 22 Kbytes (based on the inspection items of the Taiwan self-regulatory mobile application security certification program). This study calculates the average time for the report update server to perform the tasks (from Step 1 to Step 4) in the above paragraph with different maximum chunk size. For each maximum chunk size, this study performs the experiment 20 times. Table 4 summarizes the experimental results in which the experimental result illustrates the time for an inspector to store an inspection report in the blockchain. As listed in the table, the report update server can usually finish the above tasks in reasonable amount of time. On the aspect of report inquiry, a user can usually obtain the inspection report less than a second because the user can just read data in a blockchain directly without any mining behavior.

Table 4. Experimental results of storing an inspection report in the blockchain.

Maximum Chunk Size (Bytes)	1 K	2 K	4 K	8 K	16 K	32 K
Average time (seconds)	79.6	42.0	24.9	14.0	10.9	7.41

5.2. Security Analysis for SCIRM

The SCIRM framework has the following major security requirements: (a) a malicious person cannot spoof the identity of a verifier to issue an inspection report; (b) unauthorized people cannot tamper with the content of an inspection report; and (c) only the creator of an inspection report

can change the status of the report. To fulfill requirement (a), a user can obtain the verifier identity from a report manager smart contract and use the identity to retrieve information of the verifier from the notaries. This study assumes that notaries can check the authenticity of the identities and the related verifier information before the notaries register the data to their databases. Therefore, unless a malicious person can tamper databases of notaries or steal the credential information of a verifier, malicious people cannot impersonate the verifier to issue a report. For the requirement (b), if a malicious person wishes to tamper the content of an inspection report, the person would need to modify the transactions storing the chunks of the report. Moreover, the person may also try to change the variables of the associated report manager smart contract that keep addresses of the transactions to other addresses to link the report manager to a faked report. However, the Ethereum platform has built in security mechanisms to prevent people from tampering existing transactions or contract states in blocks unless the malicious person can control the majority of computing power in the Ethereum network. Therefore, unauthorized person cannot tamper the content of an inspection report while the underlying Ethereum network security mechanism are in place. To meet the requirement (c), this study uses the function modifier of the Ethereum platform to impose constraints so that only the creator of a report manager smart contract can link an inspection report to the smart contract and change the status of the report. Malicious people cannot change the status of the report unless they can obtain the credentials of the creator.

5.3. Limitations for SCIRM

We have proposed a SCIRM framework for BLE-based applications to enable smartphone application users to obtain security inspection reports of interested applications using smart contracts. Based on the blockchain technology, users can get historical inspection reports of an application and ensure the integrity of the reports. In addition, application users and other volunteers can collaborate to provide resources needed to host the framework. Therefore, as the framework does not rely on a single party, the framework does not need to consider the business interests of a business company. However, there exist certain limitations that point the way toward future research. First, this study does not address the dispute between application developers and verifiers over contents in inspection reports. In this case, an application developer and a verifier may delegate a mutually agreed third party arbitrator to decide whether or not to invalidate an inspection report or a particular part of a report. Designing and implementing a smart contract to automate the above process would be interesting future work. Second, an application developer may develop an application for different application platforms. Therefore, an application may have different identities in different platforms. To address the issue, we may assign a unique identity to link the same application developed for different platforms. Last but not least, this study currently only provides operations for a whole report. A verifier may wish to invalidate an inspection result item in a report or append a set of new inspection result items to an existing report. However, the finer the granularity, the more complex it is for users to collect inspection result items in different transactions in blockchains. Designing a fine-grained system would be a complicated and interesting problem which is worthy of further studies.

6. Conclusions

In this study, we have introduced a robust security paradigm for BLE-based smart objects in the IoT in which two viewpoints on IoT device security and smartphone applications security, respectively, are the focus. First, we addressed the security impact of adopting BLE random address technique on smartphones to prevent malicious adversary from retrieving smartphones' physical identification codes. Three critical security requirements for designing privacy-aware access-control schemes on BLE-based smart objects are thus derived. Then, a privacy-aware access-control scheme adopting a robust ECC-based crypto-module is presented to fulfill the claimed requirements. Second, we demonstrate the SCIRM framework for efficiently managing security inspection reports for the

BLE-based applications on smartphones associated with IoT objects. Based on our analysis, the proposed SCIRM framework is secure and practical for application scenarios in real world.

Acknowledgments: This work was supported in part by the Academia Sinica, in part by the Taiwan Information Security Center, and in part by the Ministry of Science and Technology, Taiwan under Grant MOST 105-2221-E-259-014-MY3, Grant MOST 105-2221-E-011-070-MY3, Grant MOST 105-2221-E-011-079-MY3, and Grant MOST 106-3114-E-011-003.

Author Contributions: Shi-Cho Cha and Kuo-Hui Yeh wrote the paper; Shi-Cho Cha and Jyun-Fu Chen conceived, designed and performed the experiments, and analyzed the data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Android Developer Website. Android 6.0 Changes. Available online: https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html (accessed on 12 October 2017).
2. Bluetooth SIG. Specification of Bluetooth System Core Package 4.2. Available online: https://www.bluetooth.com/specifications/adopted-specifications (accessed on 12 October 2017).
3. Gupta, N. *Inside Bluetooth Low Energy*; Artech House: Norwood, MA, USA, 2013.
4. Cha, S.-C.; Dai, C.-Y.; Chen, J.-F. Is there a tradeoff between privacy and security in BLE-based IoT applications: Using a smart vehicle of a major Taiwanese brand as example? In Proceedings of the 2016 IEEE 5th Global Conference on Consumer Electronics (GCCE 2016), Kyoto, Japan, 11–14 October 2016.
5. Gogoro Smart Scooters. Available online: https://www.gogoro.com/tw/ (accessed on 5 October 2017).
6. Jara, A.J.; Zamora, M.A.; Skarmeta, A.F. Knowledge acquisition and management architecture for mobile and personal Health environments based on the Internet of Things. In Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Liverpool, UK, 25–27 June 2012; pp. 1811–1818.
7. Berhanu, Y.; Abie, H.; Hamdi, M. A Test bed for Adaptive Security for IoT in eHealth. In Proceedings of the International Workshop on Adaptive Security, Zurich, Switzerland, 8–12 September 2013.
8. ASSET Project. Available online: http://asset.nr.no (accessed on 12 October 2017).
9. Torjusen, A.B.; Abie, H.; Paintsil, E.; Trcek, D.; Skomedal, Å. Towards Run-Time Verification of Adaptive Security for IOT in eHealth. In Proceedings of the 2014 European Conference on Software Architecture Workshops, Vienna, Austria, 25–29 August 2014.
10. Gope, P.; Hwang, T. BSN-Care: A Secure IoT-Based Modern Healthcare System Using Body Sensor Network. *IEEE Sens. J.* **2016**, *16*, 1368–1376. [CrossRef]
11. Gope, P.; Hwang, T. Untraceable Sensor Movement in Distributed IoT Infrastructure. *IEEE Sens. J.* **2015**, *15*, 5340–5348. [CrossRef]
12. Yao, X.; Han, X.; Du, X.; Zhou, X. A Lightweight Multicast Authentication Mechanism for Small Scale IoT Applications. *IEEE Sens. J.* **2013**, *13*, 3693–3701. [CrossRef]
13. Nyberg, K. Fast accumulated hashing. In Proceedings of the 3rdFast Software Encryption Workshop, Cambridge, UK, 21–23 February 1996; pp. 83–87.
14. Ning, H.; Liu, H.; Yang, L.T. Aggregated-proof Based Hierarchical Authentication Scheme for the Internet of Things. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 657–667. [CrossRef]
15. Hernández-Ramos, J.L.; Pawlowski, M.P.; Jara, A.J.; Skarmeta, A.F.; Ladid, L. Toward a Lightweight Authentication and Authorization Framework for Smart Objects. *IEEE J. Sel. Areas Commun.* **2015**, *33*, 690–702. [CrossRef]
16. Kawamoto, Y.; Nishiyama, H.; Kato, N.; Shimizu, Y.; Takahara, A.; Jiang, T. Effectively Collecting Data for the Location-Based Authentication in Internet of Things. *IEEE Sys. J.* **2017**, *11*, 1403–1411. [CrossRef]
17. Cirani, S.; Picone, M.; Gonizzi, P.; Veltri, L.; Ferrari, G. IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sens. J.* **2015**, *15*, 1224–1234. [CrossRef]
18. Fawaz, K.; Kim, K.-H.; Shin, K.G. Protecting Privacy of BLE Device Users. In Proceedings of the 25th USENIX Security Symposium, Austin, TX, USA, 10–12 August 2016.
19. Rizzardi, A.; Sicari, S.; Miorandi, D.; Coen-Porisini, A. AUPS: An Open Source AUthenticated Publish/Subscribe system for the Internet of Things. *Inf. Syst.* **2016**, *62*, 29–41. [CrossRef]

20. Markmann, T.; Schmidt, T.C.; Wählisch, M. Federated end-to-end authentication for the constrained internet of things using ibc and ecc. *SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 603–604. [CrossRef]
21. Pointcheval, D.; Stern, J. Security Proofs for Signature Schemes. In Proceedings of the EUROCRYPT '96 (LNCS 1070), Zaragoza, Spain, 12–16 May 1996; pp. 387–398.
22. The Bouncy Castle Crypto APIs. 2016. Available online: https://www.bouncycastle.org/ (accessed on 12 May 2017).
23. European Union Agency for Network and Information Security (ENISA). *Smartphone Secure Development Guidelines*. Available online: https://www.enisa.europa.eu/publications/smartphonesecuredevelopment-guidelines2016 (accessed on 30 June 2017).
24. Taiwan IDB (Industrial Development Bureau). Available online: https://www.moeaidb.gov.tw/external/ctlr?PRO=index&lang=1 (accessed on 12 October 2017).
25. Quirolgico, S.; Voas, J.; Karygiannis, T.; Michael, C.; Scarfone, K. Vetting the Security of Mobile Applications; (NIST) SP 800-163. Available online: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-163.pdf (accessed on 12 October 2017).
26. Mueller, B. Mobile Application Security Verification Standard (MASVS) 0.9.4. Available online: https://www.owasp.org/images/6/61/MASVS_v0.9.4.pdf (accessed on 12 October 2017).
27. Taiwan Industrial Development Bureau (IDB). Self-Regulatory Mobile App Functional Security Certification v3.0. Available online: https://www.moeaidb.gov.tw/external/en.html (accessed on 12 October 2017).
28. Accreditation Program for Mobile Application Basic Security Evaluation Laboratories 0263, 2918, 3016, 3102, 3302, 3325, 3334. Available online: http://www.taftw.org.tw/wSite/taf/list_expansion_special.jsp (accessed on 10 October 2017).
29. Ethereum Blockchain System. Available online: https://www.ethereum.org/ (accessed on 30 June 2017).