A Blockchain Connected Gateway for BLE-based Devices in the Internet of Things

Shi-Cho Cha[1], Jyun-Fu Chen[2], Chunhua Su[3], and Kuo-Hui Yeh[4*]

Abstract—Recently, the popularity of the Internet of Things (IoT) has led to a rapid development and significant advancement of ubiquitous applications seamlessly integrated within our daily life. Owing to the accompanying growth of the importance of privacy, a great deal of attention has focused on the issues of secure management and robust access control of IoT devices. In this study, we propose the design of a Blockchain Connected Gateway (BC gateway) which adaptively and securely maintains user privacy preferences for IoT devices in the blockchain network. Individual privacy leakage can be prevented because the gateway effectively protects users' sensitive data from being accessed without their consent. A robust digital signature mechanism is proposed for the purposes of authentication and secure management of privacy preferences. Furthermore, we adopt the blockchain network as the underlying architecture of data processing and maintenance to resolve privacy disputes.

Index Terms—Blockchain, Bluetooth Low Energy, Internet of Things (IoT), Security, Privacy

I. Introduction

An increasing number of IoT objects are equipped with electronics, such as passive Radio Frequency (RF) tags or Bluetooth Low Engergy (BLE) modules, to provide the objects themselves with identification, computing, and communication capabilities to support versatile ubiquitous service applications in the real world. For example, wearables with biometric data retrieval and health management have become more and more popular in our daily life (see Fig. 1). BLE-based wearables can be used to automatically monitor the body condition of individuals and effectively track their health status. However, privacy leakage may occur. From a hardware stanpoint, this could happen at the wearable itself. From a data transmission standpoint, the leakage could be from the communication channel between the wearable and the mobile gateway (or the smartphone). In another instance (e.g., Fig. 2), in a so-called smart factory, every product's identity, history, and specifications are meticulously tracked and documented. Each phase of production is monitored and machines automatically collect data corresponding to the production process. All of the collected data is forwarded to the cloud cluster and analyzed to deliver a set of intelligent functionalities for managers (or workers) to optimize the production processes via appropriate resource-utilization. Nevertheless, it is indispensable to embed an appropriate data management mechanism into the system for organizational privacy protection.

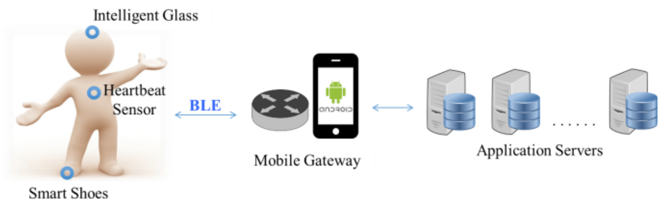


Figure 1: Example of user with wearables

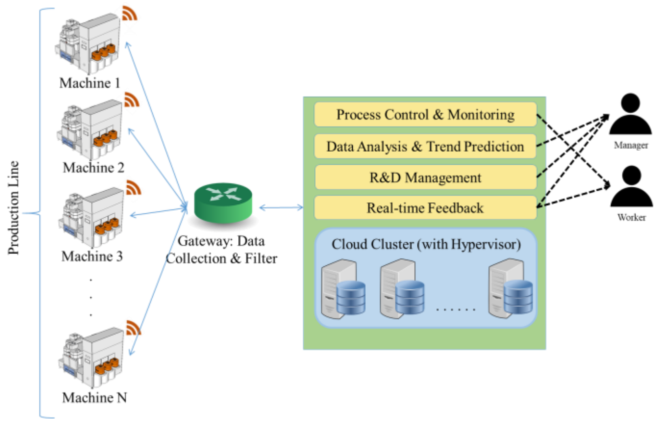

Production Line
Machine 1
Machine 2
Machine 3
Machine N
Gateway: Data Collection & Filter
Process Control & Monitoring
Data Analysis & Trend Prediction
R&D Management
Real-time Feedback
Cloud Cluster (with Hypervisor)
Manager
Worker

Figure 2: Example of a smart factory

While the IoT promises new opportunities for innovative service applications and business models through effective use of next-generation mobile devices, it also brings with it many challenges with respect to privacy issues. In recent years, several governmental agencies, such as the EU Article 29 Working Party and the American FTC (Federal Trade Commission) [1-2], have recommended that IoT application providers notify users of application privacy policies or even obtain user consent before personal data is collected and processed. On the other hand, a move toward a Privacy by

This work was supported in part by 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. In addition, Chunhua Su is supported by JSPS KAKENHI Grant Number 15K16005.

[12]Department of Information Management, National Taiwan University of Science and Technology, Taipei 10607, Taiwan. (e-mail: csc@cs.ntust.edu.tw[1]; attx2000@gmail.com[2])

[3]Chunhua Su is with the Division of Computer Science, University of Aizu, Aizu-Wakamatsu, Fukushima Pref. 965-8580, Japan. (e-mail: chsu@u-aizu.ac.jp[3])

[4]Kuo-Hui Yeh is with the Department of Information Management, National Dong Hwa University, Hualien 97401, Taiwan R.O.C. ([*]corresponding author: Kuo-Hui Yeh; e-mail: khyeh@gms.ndhu.edu.tw[4])

Design (PbD) legal framework has been recommended [3-4], in which seven major principles constitute a fundamental framework for privacy-aware applications (or audit services). These principles are: (1) proactive not reactive, preventative not remedial (2) privacy as the default (3) privacy embedded into design (4) full functionality – positive - sum, not zero - sum (5) end-to-end security - lifecycle protection (6) visibility and transparency, and (7) respect for user privacy. To follow the above 6th and 7th principles, many researchers have dedicated their efforts to allowing application provider to negotiate with users to reach appropriate privacy agreements [5-7], or to the design and implementation of privacy-aware IoT systems [8-10]. Although these studies offer good practical privacy solutions, a fundamental limitation exists in all of them. That is, significant modifition of existing (or legacy) IoT devices may be required to support the operation logic for privacy protection (or policy management) presented by the researchers [5-10]. Modifying (or even replaceing) existing IoT devices to support any newly proposed processes may entail a significant rise in cost.

In light of the foregoing, this study designs and proposes a privacy-aware Blockchain Connected Gateway (BC gateway), where the blockchain network is adopted as the underlying architecture for management of privacy preferences. That is, the proposed BC gateway uses blockchain technology to protect and manage the maintained user preferences from being tampered with. Therefore, the BC gateway enhances user privacy protection while legacy IoT devices are in use. In addition, the blockchain-based user preference management scheme is useful for solving disputes between users and IoT application providers when it comes to privacy practices. The rest of this paper is organized as follows: In Section II, we provide the operation scenario of the proposed BC gateway. Section III illustrates the detailed functions of the proposed BC gateway, including the device binding process, the proposed digital signature scheme and the procedure for preserving privacy preferences. Next, we perform the security analysis and perfromance evaluation in Section IV. Finally, conclusions are drawn in Section V.

II. OPERATION SCENARIO: THE OVERVIEW OF THE PROPOSED FRAMEWORK

In this section, we demonstrate a broad overview of the functionalities of the proposed BC gateway, as shown in Fig. 3, with a scenario provided to illustrate the functions of the proposed scheme. In general, there are three main types of participants in the scenario involving our proposed BC gateway: (1) the owners or administrators of IoT devices, (2) the BC gateway administrators, and (3) the end users. Before a user can access an IoT device, the administrator of the device can store device information and the privacy policies of the device in the blockchain network. In general, the device information includes a list consisting of: (1) an unique device name (2) manufacture related information (3) features of the device such as device type, device model name and number, serial number and so on, and (4) other attributes for management purposes, such as list of device images, privacy policies, and services provided. On the other hand, privacy policy includes a policy identifier, and preference-related information (i.e. data collector, access, and dispute). In this study, we define the privacy policies in JSON-based machine readable format. The Ethereum blockchain platform was chosen for this study because of its ability to execute and enforce smart contracts[1]. Therefore, the administrator of an IoT device can create a smart contract for the device and use the smart contract to manage the device's information and privacy policies (Step 0a).

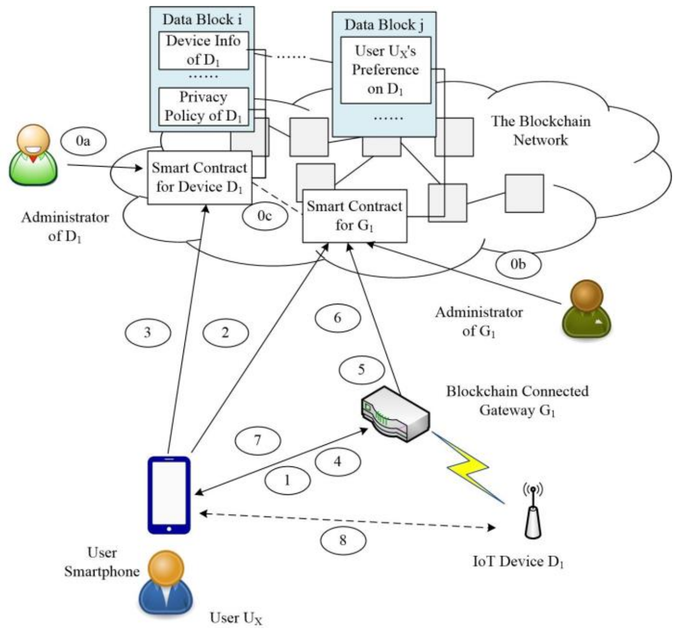


Figure 3: An operation scenario

This study also defines the smart contract for a BC gateway. The administrator of a BC gateway can create a smart contract for the gateway (Step 0b). After connecting the gateway to an IoT device physically, the gateway administrator will link the smart contract of the device to the smart contract of the gateway (Step 0c). When a user uses his/her smartphone to connect to a BC gateway (Step 1), the user can obtain the address of the gateway's smart contract. Then, the user can query the list of devices connected to the gateway from the gateway's smart contract (Step 2). Hereafter, the user can retrieve the address of an IoT device's smart contract and fetch device information and privacy policies via the device's smart contract (Step 3).

After receiving the privacy policies of an IoT device, a user can connect to the associated BC gateway and notify the gateway that he/she accepts or declines the policies (Step 4). In addition to storing the preference data in the gateway (Step 5), the gateway further preserves the preference data in the blockchain network (Step 6). Consequently, when the user accesses the IoT device via the gateway (Step 7 and Step 8), the gateway will process user requests based on the preserved user preferences.

[1] Note that there are other smart contract platforms like Hyperledger. This study leaves implementing the proposed framework on other smart contract platforms to future work.

In the above scenario, the BC gateway will utilize the blockchain technology to protect the user preferences and privacy policy, and thus eliminate disputes involving privacy practices.

III. THE PROPOSED BC GATEWAY

In this section, we depict the architecture of the BC gateway as shown in Figure 4. The BC gateway identifies users and user preferences based on the accounts in the blockchain network. Therefore, a user can use the same account to connect to different BC gateways, rather than registering for each gateway. As major smartphone platforms such as Android and iOS have supported the BLE specification, BLE has become the de facto standard for smartphones to communicate with wearable devices and with nearby IoT devices. This study focuses on the scenario wherein a user accesses nearby BLE-based devices. In the current state of the art, the BC gateways provide a REST-like interface for users to access BLE-based devices with the following commands:

- Lock / unlock a BLE device.
- Discover devices connected to the gateways and services provided by the devices.
- Send read or write requests to a characteristic of a BLE device.
- Request to receive notification or indication of a BLE device's characteristic.

The BC gateways will maintain connectivity to related IoT devices. Therefore, the gateways can play the roles of mediators to forward requests to specified devices and integrate responses from the devices. Finally, the BC gateways provides interfaces for users to manage their privacy preferences and determine whether personal data can be forwarded to an IoT device based on the user's preferences.

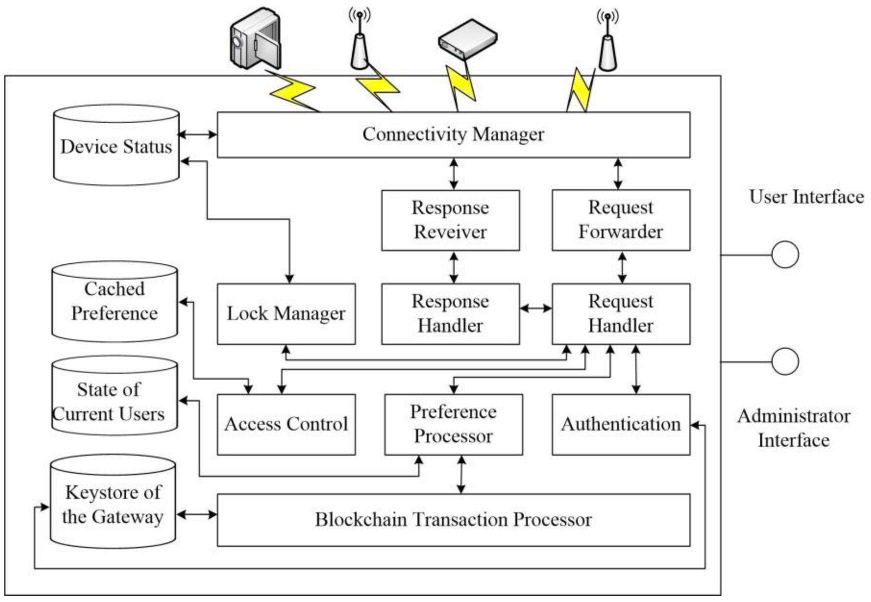


Figure 4: The architecture of a BC gateway

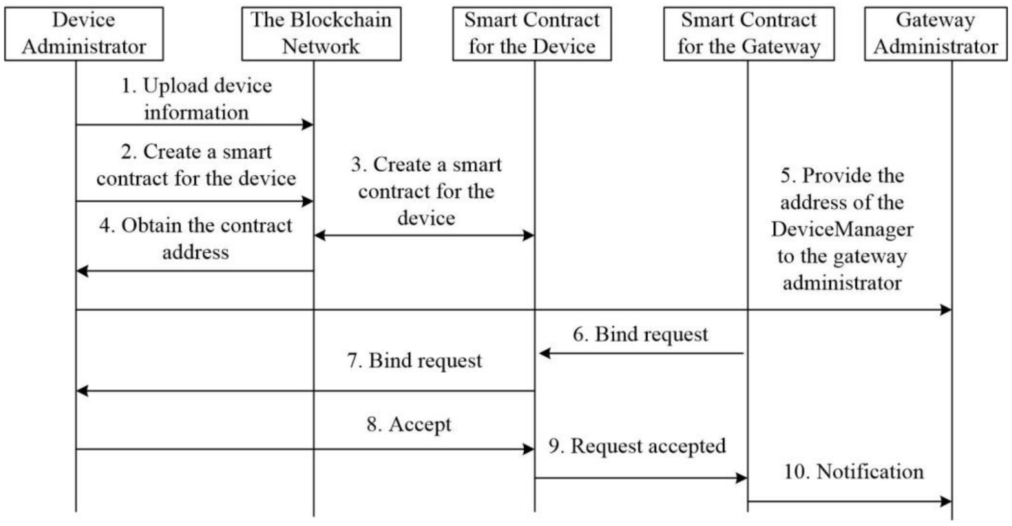
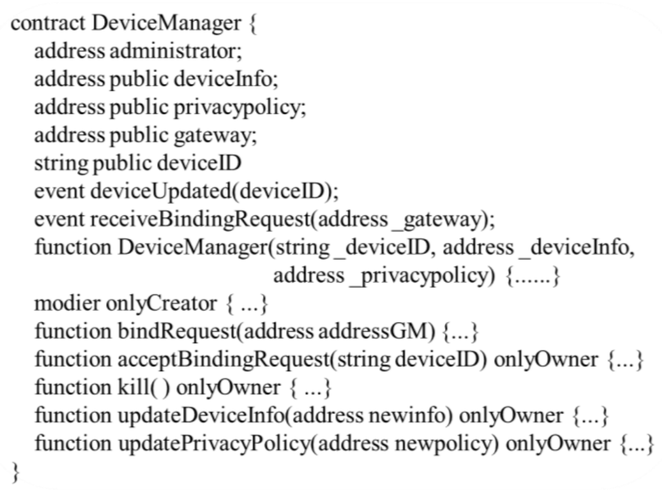

Device Administrator
The Blockchain Network
Smart Contract for the Device
Smart Contract for the Gateway
Gateway Administrator
1. Upload device information
2. Create a smart contract for the device
3. Create a smart contract for the device
4. Obtain the contract address
5. Provide the address of the DeviceManager to the gateway administrator
6. Bind request
7. Bind request
8. Accept
9. Request accepted
10. Notification

Figure 5: The device registration process

A. *Device Binding*

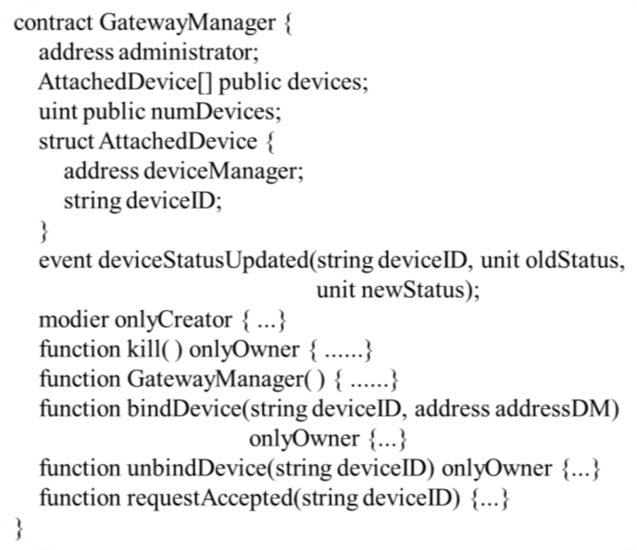
Fig. 5 demonstrates the process for a device administrator to register a device with a BC gateway. Before creating a smart contract of an IoT device, the device administrator invokes transactions to store device information and associated privacy policies. Then, the administrator can create a *DeviceManager* corresponding to the device with the addresses of the device information and privacy policies. As illustrated in Fig. 6, users can obtain device information and privacy policies of a device based on the associated public variables of the device's DeviceManager. In addition, users can listen for events of device updating.

After creating a DeviceManager, the device administrator can obtain the address of the smart contract. Then, the device administrator can provide the address to a gateway administrator. Therefore, the gateway administrator can submit a request to bind the device to the GatewayManager of a BC gateway with the address of the DeviceManager.



```
contract DeviceManager {
  address administrator;
  address public deviceInfo;
  address public privacypolicy;
  address public gateway;
  string public deviceID
  event deviceUpdated(deviceID);
  event receiveBindingRequest(address _gateway);
  function DeviceManager(string _deviceID, address _deviceInfo,
                          address _privacypolicy) {......}
  modier onlyCreator { ...}
  function bindRequest(address addressGM) {...}
  function acceptBindingRequest(string deviceID) onlyOwner {...}
  function kill( ) onlyOwner { ...}
  function updateDeviceInfo(address newinfo) onlyOwner {...}
  function updatePrivacyPolicy(address newpolicy) onlyOwner {...}
}
```

Figure 6: Abstract of the DeviceManager smart contract.



```
contract GatewayManager {
  address administrator;
  AttachedDevice[] public devices;
  uint public numDevices;
  struct AttachedDevice {
    address deviceManager;
    string deviceID;
  }
  event deviceStatusUpdated(string deviceID, unit oldStatus,
                             unit newStatus);
  modier onlyCreator { ...}
  function kill( ) onlyOwner { ......}
  function GatewayManager( ) { ......}
  function bindDevice(string deviceID, address addressDM)
                        onlyOwner {...}
  function unbindDevice(string deviceID) onlyOwner {...}
  function requestAccepted(string deviceID) {...}
}
```

Figure 7: Abstract of the GatewayManager smart contract.

Fig. 7 provides the abstract interface of a GatewayManager. Upon receiving a request to bind a device, a GatewayManager will invoke the *bindRequest* method of the associated DeviceManager. The DeviceManager then notifies its administrator to decide whether to accept the request. If the administrator accepts the request, the DeviceManager smart contract records the address of the GatewayManager. Therefore, a user can ensure that the device administrator has confirmed the binding relationship between the device and the gateway.

B. *The Proposed Digital Signature Scheme (PDSS)*

This section presents our proposed signature scheme (hereinafter PDSS). The security of the scheme is based on the intractability of ECDLP and the robustness of bilinear pairing. In general, the proposed signature scheme consists of six phases, i.e. *Setup*, *Set-Partial-Private-Key*, *Set-Secret-Value*, *Set-Public-Key*, *Sign* and *Verify*.

Setup: Let the notation E/E_p denote 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 $t \cdot P = P + P + \cdots + P$ (t times) as scalar multiplication, where P is a generator of G with order n. The Elliptic Curve Discrete Logarithm Problem (ECDLP) is defined as follows: 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 $x \in Z_n^*$. Moreover, Let G_1 and G_2 be the cyclic group with the same prime order q where G_1 is an additive cyclic group and G_2 is a multiplicative cyclic group. Let $e: G_1 \times G_1 \to G_2$ and $\forall\, a, b \in Z_q^*$, $\forall\, P, Q \in G_1$: $e(aP, bQ) = e(P, Q)^{ab}$.

Given a secure parameter k, $BC\ gateway$ (BCG) chooses two groups G_1 and G_2 with the same prime order q and a bilinear pairing $e: G_1 \times G_1 \to G_2$, where P is a generator of G_1. Next, BCG chooses a random number $s \in Z_q^*$ as its master private key and then computes its corresponding master public key $PK_{BCG} = s \cdot P$. After that, BCG chooses three secure hash functions, i.e. $H_1: \{0,1\}^* \times G_1 \to Z_q^*$, $H_2: \{0,1\}^* \times G_1 \times G_1 \to Z_q^*$ and $H_3: \{0,1\}^* \times G_1 \times G_1 \times G_1 \to Z_q^*$. Finally, BCG publishes $\{G_1, G_2, q, e, P, PK_{BCG}, H_1, H_2, H_3, e(P,P)\}$ as public parameters.

Set-Partial-Private-Key: Given the public parameters, i.e. $Params = \{G_1, G_2, q, e, P, PK_{BCG}, H_1, H_2, H_3, e(P,P)\}$, master private key s, and the user U_i's identity ID_i, BCG generates a random number $r_i \in Z_q^*$ and calculates

$$R_i = r_i \cdot P,\ h_i = H_1(ID_i, R_i, PK_{BCG}),$$
$$s_i = r_i + h_i \cdot s \mod q, \text{ and } \sigma_{i\_1} = s_i^{-1} \cdot P$$

Then, BCG sends the partial private key (R_i, σ_{i_1}) back to the user. Upon receiving (R_i, σ_{i_1}) from BCG, the user U_i verifies the validity of the incoming message via the following equations:

$$\text{Compute } h_i = H_1(ID_i, R_i, PK_{BCG}),$$
$$\text{and Check if } e\left(\sigma_{i\_1}, R_i + h_i \cdot PK_{BCG}\right) = e(P,P)?$$

If the verification holds, the user U_i believes that the partial private key is valid. The correctness of the verification is

$$\begin{aligned} &e\left(\sigma_{i\_1}, R_i + h_i \cdot PK_{BCG}\right) \\ &= e(s_i^{-1} \cdot P, (r_i \cdot P + h_i \cdot s \cdot P)) \\ &= e((r_i + h_i \cdot s)^{-1} \cdot P, (r_i + h_i \cdot s) \cdot P) \\ &= e(P,P)^{(r_i + h_i \cdot s)^{-1} \times (r_i + h_i \cdot s)} \\ &= e(P,P)^{\frac{(r_i + h_i \cdot s)}{(r_i + h_i \cdot s)}} = e(P,P) \end{aligned}$$

Set-Secret-Value: Given $Param$, the user U_i chooses a random number $x_i \in Z_q^*$ as his/her secret value.

Set-Public-Key: Given $Params$ and x_i, the user U_i sets $PK_i = x_i \cdot P$ as his/her public key.

Sign: Given $Params$, (s_i, R_i), h_i, x_i, PK_i and a message m, the user U_i computes

$$k_i = H_2(ID_i, PK_i, R_i, PK_{BCG}, m) \text{ and}$$
$$\sigma_{i\_2} = (k_i \cdot s_i + x_i)^{-1} \cdot P.$$

After that, the user returns (m, R_i, σ_{i_2}) to the verifier as the signature of m.

Verify: Given $Params$, PK_i and (m, R_i, σ_{i_2}), the verifier computes

$$h_i = H_1(ID_i, R_i, PK_{BCG}) \text{ and}$$
$$k_i = H_2(ID_i, PK_i, R_i, PK_{BCG}, m).$$

Next, the verifier verifies the validity of the signature σ_{i_2} via the following equation:

$$\text{Check if } e\left(\sigma_{i\_2}, k_i \cdot (R_i + h_i \cdot PK_{BCG}) + PK_i\right) = e(P,P)?$$

If the aforementioned equation holds, the signature σ_{i_2} is valid. The correctness of the verification is

$$\begin{aligned} &e\left(\sigma_{i\_2}, k_i \cdot (R_i + h_i \cdot PK_{BCG}) + PK_i\right) \\ &= e((k_i \cdot s_i + x_i)^{-1} \cdot P, k_i \cdot (r_i \cdot P + h_i \cdot s \cdot P) + x_i \cdot P) \\ &= e([k_i \cdot (r_i + h_i \cdot s) + x_i]^{-1} \cdot P, [k_i \cdot (r_i + h_i \cdot s) + x_i] \cdot P) \\ &= e(P,P)^{[k_i \cdot (r_i + h_i \cdot s) + x_i]^{-1} \times [k_i \cdot (r_i + h_i \cdot s) + x_i]} \\ &= e(P,P)^{\frac{[k_i \cdot (r_i + h_i \cdot s) + x_i]}{[k_i \cdot (r_i + h_i \cdot s) + x_i]}} \\ &= e(P,P) \end{aligned}$$

C. *Privacy Preference Preserving*

Since user preferences are stored in the blockchain network, user privacy and data confidentiality should be considered. Among different confidentiality and privacy preserving technologies for blockchains [11], this study lets the BC gateway encrypt user preference on the user's behalf and store the encrypted user preference in the blockchain network.

Fig. 7 illustrates the process for a user to express his/her privacy preference on an IoT device. A gateway has a root key K to encrypt user preferences. As shown in Fig. 7, when the gateway receives a user U_i's privacy preference PP_j on a privacy policy P_j, the gateway will generate a nonce N_i and a key $K_{U_{iP_j}}^{N_i} = H(K||N_i||PK_i)$ by passing the values of K, N_i, PK_i to a hash function $H(\,)$. After encrypting the privacy preference along with the key $K_{U_{iP_j}}^{N_i}$, the gateway stores the encrypted message $EM = E_{K_{U_{iP_j}}^{N_i}}\left(PP_j||P_j||T_{ij}\right)$ in the blockchain network, where a timestamp T_{ij} is maintained. Hereafter, the gateway can obtain a transaction ID TID_{ij} for the transaction that stores the message. Therefore, the gateway can provide the transaction ID to its smart contract. Finally, the gateway transfers the transaction ID (i.e. TID_{ij}) and the key (i.e. $K_{U_{iP_j}}^{N_i}$) to the user so that the user can verify that his/her privacy preference is really stored in the blockchain network.

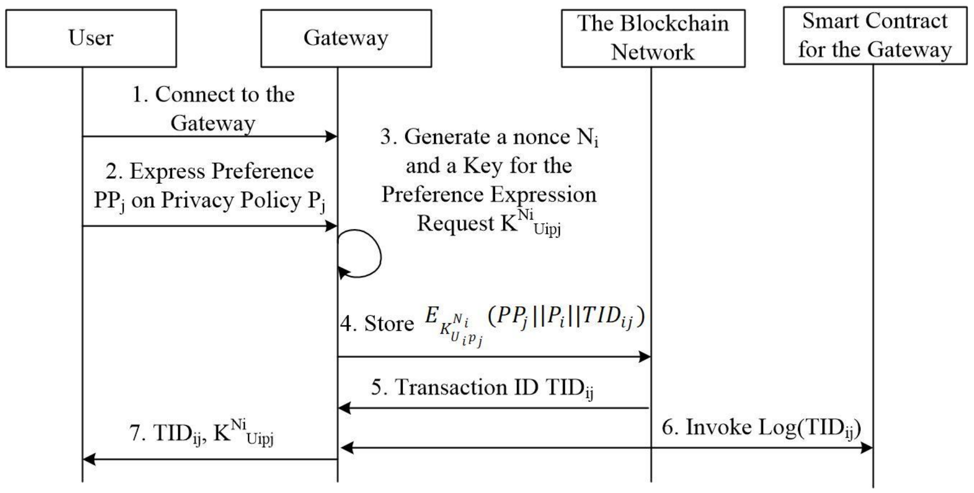

User
Gateway
The Blockchain Network
Smart Contract for the Gateway
1. Connect to the Gateway
2. Express Preference PP$_j$ on Privacy Policy P$_j$
3. Generate a nonce N$_i$ and a Key for the Preference Expression Request K$^{Ni}_{Uipj}$
4. Store $E_{K_{U_{iPj}}^{N_i}}(PP_j||P_i||TID_{ij})$
5. Transaction ID TID$_{ij}$
6. Invoke Log(TID$_{ij}$)
7. TID$_{ij}$, K$^{Ni}_{Uipj}$

Figure 7: The process for a user to express privacy preference

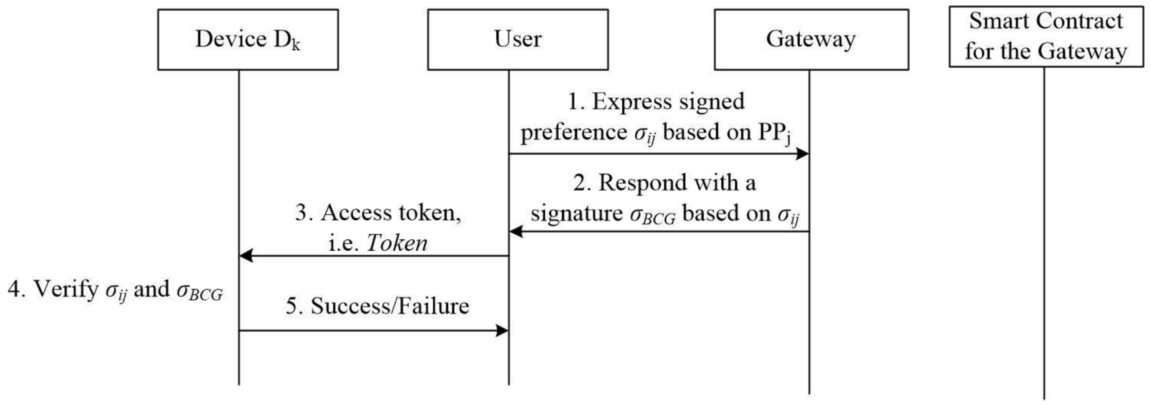

Device D_k
User
Gateway
Smart Contract for the Gateway
1. Express signed preference σ_{ij} based on PP_j
2. Respond with a signature σ_{BCG} based on σ_{ij}
3. Access token, i.e. *Token*
4. Verify σ_{ij} and σ_{BCG}
5. Success/Failure

Figure 8: Intelligent access control on IoT devices

D. *Intelligent Access Control on IoT Devices (e.g., Fig. 8)*

Once the user U_i would like to perform a service corresponding to a set of IoT devices, U_i produces a signature on the privacy preference PP_j. That is, given $Params$, (s_i, R_i), h_i, x_i, PK_i and PP_j, the user U_i computes $k_{ij} = H_2(ID_i, PK_i, R_i, PK_{BCG}, PP_j)$ and $\sigma_{ij} = (k_{ij} \cdot s_i + x_i)^{-1} \cdot P$. Note that $R_i = r_i \cdot P$, $h_i = H_1(ID_i, R_i, PK_{BCG})$, and $s_i = r_i + h_i \cdot s$. After that, the user sends (PP_j, R_i, σ_{ij}) to the BCG as the signature of PP_j. After that, BCG computes $h_i = H_1(ID_i, R_i, PK_{BCG})$ and $k_{ij} = H_2(ID_i, PK_i, R_i, PK_{BCG}, PP_j)$ with a given dataset including $Params$, PK_i and (PP_j, R_i, σ_{ij}). Next, BCG verifies the validity of the signature σ_{ij} via the following equation: if $e(\sigma_{ij}, k_{ij} \cdot (R_i + h_i \cdot PK_{BCG}) + PK_i) = e(P, P)$? If the aforementioned equation holds, the signature σ_{ij} is valid.

$$\begin{aligned} & e(\sigma_{ij}, k_{ij} \cdot (R_i + h_i \cdot PK_{BCG}) + PK_i) \\ & = e((k_{ij} \cdot s_i + x_i)^{-1} \cdot P, k_{ij} \cdot (r_i \cdot P + h_i \cdot s \cdot P) + x_i \cdot P) \\ & = e(P, P)^{[k_{ij} \cdot (r_i + h_i \cdot s) + x_i]^{-1} \times [k_{ij} \cdot (r_i + h_i \cdot s) + x_i]} = e(P, P) \end{aligned}$$

Upon successfully verifying σ_{ij}, BCG generates a random number and computes $r_{ij} \in Z_q^*$ and calculates $R_{ij} = r_{ij} \cdot P$, $l_{ij} = H_1(ID_i, R_{ij}, \sigma_{ij}, PK_{BCG})$, $s_{ij} = r_{ij} + l_{ij} \cdot s$, and $\sigma_{BCG} = s_{ij}^{-1} \cdot P$. Then, BCG sends (R_{ij}, σ_{BCG}) back to the user. Upon receiving (R_{ij}, σ_{BCG}) from BCG, the user U_i verifies the validity of the incoming message via the following computations: (1) calculate $l_{ij} = H_1(ID_i, R_{ij}, \sigma_{ij}, PK_{BCG})$, and (2) check if $e(\sigma_{BCG}, R_{ij} + l_{ij} \cdot PK_{BCG}) = e(P, P)$? If these two verifications hold, the user U_i believes that σ_{BCG} is valid.

$$\begin{aligned} & e(\sigma_{BCG}, R_{ij} + l_{ij} \cdot PK_{BCG}) \\ & = e\left(s_{ij}^{-1} \cdot P, (r_{ij} \cdot P + l_{ij} \cdot s \cdot P)\right) \\ & = e(P, P)^{(r_{ij} + l_{ij} \cdot s)^{-1} \times (r_{ij} + l_{ij} \cdot s)} = e(P, P) \end{aligned}$$

Next, the user U_i sends an access token, i.e. $Token = (PP_j, R_i, \sigma_{ij}, R_{ij}, \sigma_{BCG})$ to the corresponding smart devices D_k. At each IoT-based device D_k side, D_k will verify σ_{ij} and σ_{BCG} to ensure whether the access behavior is valid or not. A success acknowledgement will be sent to the user U_i if the verifications of σ_{ij} and σ_{BCG} are passed. Otherwaise, a Failure result will be sent out by D_k.

IV. Performance Evaluation

In this section, we present the performance evaluation of PDSS and smart contract management on the Ethereum blockchain platform to examine the practicability of our proposed BC gateway.

A. *Performance Evaluation of PDSS*

We implement each security component of PDSS on a popular IoT-based testbed, i.e. Raspberry PI III Model B, and calculate the corresponding computation cost. Table 1 presents the implementation environment, where Raspberry PI III is simulated as a smart device and all of the security components are programmed via Oracle Java 8 and Eclipse 3.8. Based on the experiment results, Table 2 shows the computation cost of each security module adopted in PDSS.

Table 1. Implementation environment

Raspberry PI III	Broadcom BCM2837@1GHz Quad-Core ARM Cortex-A53 64Bit Processor
	1GB DDR3 RAM
	SanDisk 8GB Class 10 SD Card
Software Platform	Debian 8 (Raspbian 2016/10)
	Oracle Java 8 for ARM
	Eclipse 3.8

Table 2. Computation cost of each security module in PDSS

Security Module	Cost
Random number generator (96 bits)	0.5ms
Hash function (SHA-512 with input 1000 bits)	7ms
ECC Pairing (384 bits)	240ms
ECC point multiplication (384 bits)	4ms
ECC point addition (384 bits)	2ms

For each signature signing and verification in PDSS, we require a one-way hash function (SHA-512 with input 1000 bits) to be performed three times, ECC pairing (384 bits) to be performed once, ECC point multiplication (384 bits) to be performed four times, and ECC point addition (384 bits) to be performed three times. The total computation cost of PDSS is nearly 283 ms, which is practical and reasonable for application development in real world.

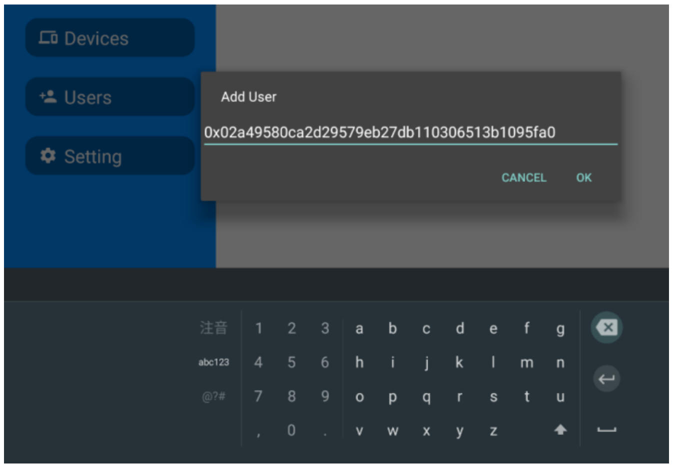
B. Implementation of the Proposed BC Gateway

In this section, we implement a prototype system to verify the practical potential of the proposed BC gateway. The experimental environment contains three entities, i.e. the Ethereum network, the BC gateway and the client application. First, we construct a private Ethereum network, which is executed on a desktop with Intel Core i7-3770 3.4GHz CPU and 16GB RAM running Windows 10. In the experiment, we fix the mining difficulty to 0x4000. Second, the BC Gateway is simulated on the NVIDIA Shield TV with NVIDIA® Tegra® X1 processor and 3G RAM. The NVIDIA Shield TV is not only a simple game console (or streaming device) but acts as a hub for the home which can handle the requests and the responses of the IoT devices. Third, the client application is implemented on users' smartphone, where an LG Nexus 5X with Android 8.0 (Oreo) is adopted. The interface of the client application can be utilized for users to express (or input) their privacy preferences. We summary all of the above detailed specifications in Table 3. In the following, we demonstrate our implementation and the corresponding screenshot as shown in Figures 9-15. Note that in our experiment, the access to the IoT device is strictly controlled based on the privacy policy. Without the user's consent, access is not allowed to the IoT device.

TABLE 3. Experimental evironment

System	Operating System	Processor	Storage
Desktop (Ethereum Network)	Windows 10	Intel Core i7-3770 3.4GHz	500 GB
NVIDIA Shield TV (Gateway)	Android 7.0 (Nougat)	NVIDIA® Tegra® X1 processor with a 256-core GPU and 3GB RAM	16GB
LG Nexus 5X (Client Application)	Android 8.0 (Oreo)	1.8GHz hexa-core with 2GB RAM	16GB

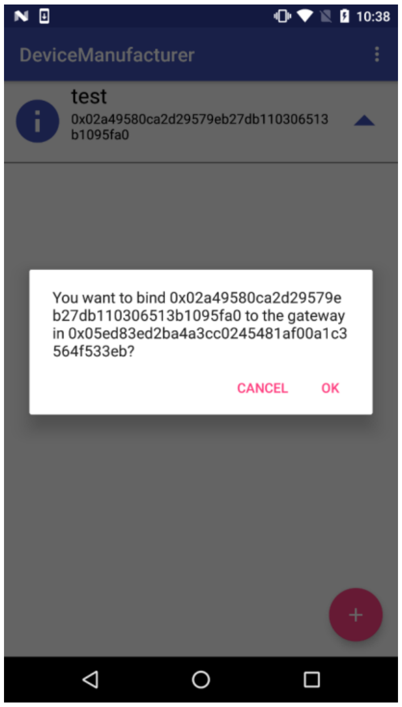
The device administrator will first invoke a transaction to store device information and the associated privacy policies. Then the administrator can create a DeviceManager smart contract of the device with the addresses of the device information and privacy policies. After getting the address of the smart contract, the device administrator can provide the address to a gateway administrator. Next, the gateway administrator keys in the address of the DeviceManager smart contract and invokes a request to bind the device to the GatewayManager smart contract of a BC gateway, which is showed in Figure 9. Then the device administrator will receive this binding request and decide whether to accept, which is illustrated in Figure 10. If the device administrator accepts the request, the DeviceManager smart contract will record the address of the GatewayManager and finish the binding process. A successful binding will be launched as shown in Figure 11. Now the users is able to ensure that the device administrator has confirmed the binding relationship between the device and the gateway.



Add User

0x02a49580ca2d29579eb27db110306513b1095fa0

Figure 9. The step for the device administrator to add the address of the smart contract for the device

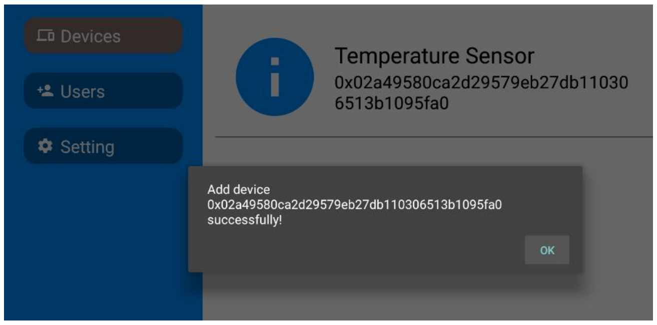


test

0x02a49580ca2d29579eb27db110306513b1095fa0

You want to bind 0x02a49580ca2d29579eb27db110306513b1095fa0 to the gateway in 0x05ed83ed2ba4a3cc0245481af00a1c3564f533eb?

Figure 10. The step for the device administrator to confirm the binding relationship

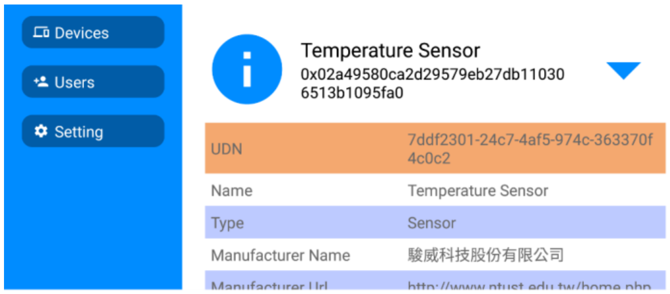


Temperature Sensor

0x02a49580ca2d29579eb27db110306513b1095fa0

Add device 0x02a49580ca2d29579eb27db110306513b1095fa0 successfully!

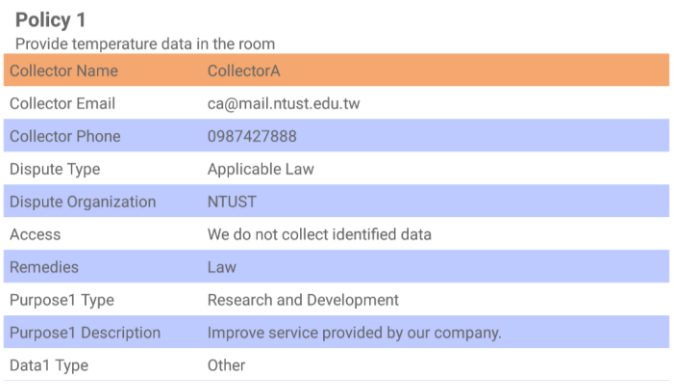
Figure 11. The screenshot of the successful binding popup message



Temperature Sensor
0x02a49580ca2d29579eb27db11030
6513b1095fa0

UDN	7ddf2301-24c7-4af5-974c-363370f4c0c2
Name	Temperature Sensor
Type	Sensor
Manufacturer Name	駿威科技股份有限公司
Manufacturer Url	http://www.ntust.edu.tw/home.php

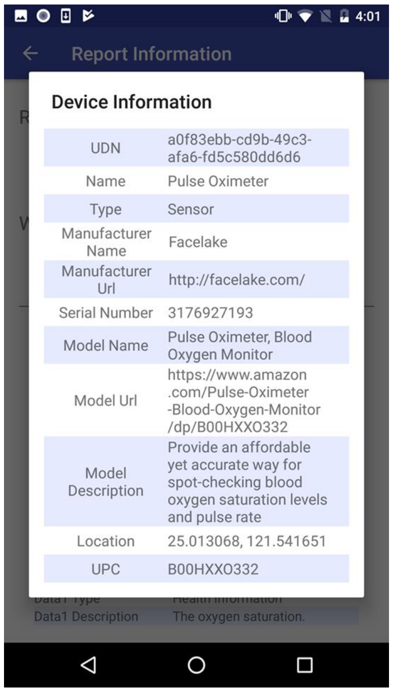
Figure 12. The device information provided by the BC gateway



Policy 1
Provide temperature data in the room

Collector Name	CollectorA
Collector Email	ca@mail.ntust.edu.tw
Collector Phone	0987427888
Dispute Type	Applicable Law
Dispute Organization	NTUST
Access	We do not collect identified data
Remedies	Law
Purpose1 Type	Research and Development
Purpose1 Description	Improve service provided by our company.
Data1 Type	Other

Figure 13. The privacy policy of the device provided by the BC gateway

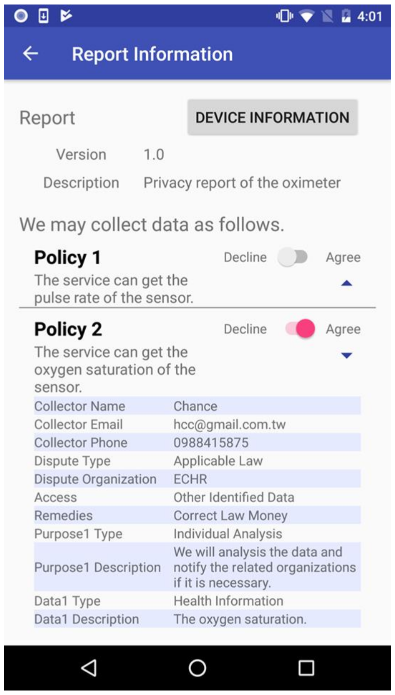


Device Information

UDN	a0f83ebb-cd9b-49c3-afa6-fd5c580dd6d6
Name	Pulse Oximeter
Type	Sensor
Manufacturer Name	Facelake
Manufacturer Url	http://facelake.com/
Serial Number	3176927193
Model Name	Pulse Oximeter, Blood Oxygen Monitor
Model Url	https://www.amazon.com/Pulse-Oximeter-Blood-Oxygen-Monitor/dp/B00HXXO332
Model Description	Provide an affordable yet accurate way for spot-checking blood oxygen saturation levels and pulse rate
Location	25.013068, 121.541651
UPC	B00HXXO332

Figure 14. Device information on user’s smartphone.

Afterwards, the administrators can view the information and privacy policies of the device connected to the BC gateway, which are displayed in Figure 12 and Figure 13. The format of the privacy policy is represented by Platform for Privacy Preferences Project (P3P), which provides the description for the collection and use of data. On the other hand, the user can use his/her smartphone to connect to the BC gateway and also view the information and privacy policies of the device, which are depicted in Figure 14 and Figure 15. Further, the user will utilize the application of the smartphone showed in Figure 15 to decide whether or not to agree with the privacy policies of an IoT device.



Report

Version	1.0
Description	Privacy report of the oximeter

We may collect data as follows.

Policy 1 — Decline / Agree
The service can get the pulse rate of the sensor.

Policy 2 — Decline / Agree
The service can get the oxygen saturation of the sensor.

Collector Name	Chance
Collector Email	hcc@gmail.com.tw
Collector Phone	0988415875
Dispute Type	Applicable Law
Dispute Organization	ECHR
Access	Other Identified Data
Remedies	Correct Law Money
Purpose1 Type	Individual Analysis
Purpose1 Description	We will analysis the data and notify the related organizations if it is necessary.
Data1 Type	Health Information
Data1 Description	The oxygen saturation.

Figure 15. The content and selection bars of the Privacy policies on user’s smartphone.

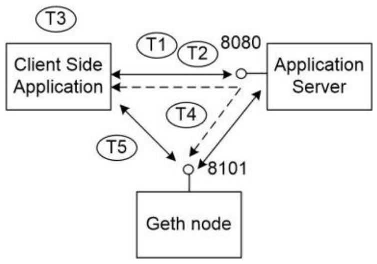


Figure 16: The experiment conducted for performance evaluation of the smart contract.

C. Performance Evaluation of Smart Contract Management

To evaluate the performance of the operations of the smart contract, we execute our experiment on a Asus ZenBook UX430UQ with Intel Core™ i7-7500U processor, 16GB 2133MHz DDR4 RAM, and 1T SATA3 M.2 SSD. The experiment is conducted as illustrated in Fig. 16, in which three major components, i.e. client-side application, geth node and backend application server, are designed. First, this study implements a client-side application with JDK 8u151, and a server side Web application using JDK 8u151 and Java EE 7. In addition, we deploy the server side application on a Tomcat 8.0.27 Web application server (listening on port 8080). Finally, this study adopts Geth v1.7 to launch a single node Ethereum private chain. The node listens to JSON-RPC requests on the 8101 port. This study embeds a Geth wallet file in the client side application. Therefore, we can obtain the public/private key pair and the associated wallet address from the file.

Similarly, this study embeds an another Geth wallet file in the server side application. The performance is measured via the following metrics T1 to T5. Note that for each of the metrics, we perform the experiment 10 times and retrieve an average value as the experiment results.

- Criterion T1 models the following processes. First, the client sends its id, public key and a random number r1 to the application server. The server then exploits the elliptic-curve Diffie-Hellman (ECDH) algorithm to generate a shared key *K* with the client's public key and the server's private key. Next, the server generates a random number r2, and encrypts the random numbers r1 and r2 with *K*. Note that the encryption algorithm is the Advanced Encryption Standard (AES). After that, the server sends the encrypted message E_K(r1, r2) with the server's public key to the client. The above processes are represented as criterion T1. Based on our experiment, we obtain an average time of 28 ms for T1.

```
contract Agreement {
   address owner;
   string public datasig = "";
   string public sigR = "";
   string public sigS = "";
   function Agreement(string _ds, string _sigR, string _sigS) public {
      owner = msg.sender;
      datasig = _ds;
      sigR = _sigR;
      sigS = _sigS;
   }
   modifier onlyOwner {
      require(msg.sender==owner);
      _;
   }
   function kill( ) onlyOwner public {
      selfdestruct(owner);
   }
   function getDataSig( ) public constant returns (string) {
      return datasig;
   }
   function getR( ) public constant returns (string) {
      return sigR;
   }
   function getS( ) public constant returns (string) {
      return sigS;
   }
}
```

Figure 17: The agreement adopted in our experiments.

- For criterion T2, the client uses its private key and the server's public key to generate the shared key *K* via the ECDH algorithm. Then, the client decrypts E_K(r1, r2) and retrieves r2. Finally, the client computes E_K(r2) and sends E_K(r2) to the server. Once the server decrypts E_K(r2) and verifies r2, the server will sends a success (or failure) acknowledgement to the client based on the verification result. The above processes are represented as criterion T2. According to our experiment results, we get an average time of 15 ms for T2.

- Criterion T3 models the client producing a digest and a digital signature of an agreement, as shown in Fig. 17, with a size of 2K bytes. In our experiment, the hash function is the SHA-256 algorithm, and the digital signature scheme is the Elliptic Curve Digital Signature Algorithm (ECDSA). An average time of 8 ms is calculated for T3.

- Criterion T4 models the processes of uploading a smart contract to the server, and then the storing of this smart contract in the blockchain by the server. The computation time required for T4 is 15068 ms (around 15 secs) on average.

- Criterion T5 represents the data query process for a smart contract at the server side. The experiment results show that retrieving a smart contract from the blockchain requires 1399 ms, on average.

V. Conclusion

To enable IoT service providers to obtain user consent on privacy policies without modifying (or replacing) legacy IoT devices immediately, this study has proposed the Blockchain Connected Gateway. The BC gateway plays the role of a mediator between users and IoT devices: users can obtain the device information and privacy policies of an IoT device connected to a BC gateway and access the device via the BC gateway rather than accessing the device directly. Therefore, the BC gateway can prevent the device from obtaining sensitive personal data unless users accept the privacy policies of the device. Moreover, the BC gateway will store a user's preference regarding privacy policies in the blockchain network. Because data stored in the blockchain network are tamper-resistant, user preference data stored in the blockchain network can be utilized to resolve disputes between users and IoT service providers. Therfore, this paper can contribute to improving user privacy and trust in IoT applications while legacy IoT devices are still in use.

References

[1]. EU Article 29 Data Protection Working Party, "Opinion 8/2014 on the on recent developments on the internet of things," European Commission, 2014.
[2]. US Federal Trade Commission, The Internet of Things: Privacy and Security in a Connected World, ser. Federal Trade Commission staff reports. DIANE Publishing Company, 2015.
[3]. Noria Foukia, David Billard, and Eduardo Solana, "PISCES: A framework for privacy by design in IoT," in Proceedings of the 14th Annual Conference on Privacy, Security and Trust, Auckland, New Zealand, 12-14 Dec. 2016.
[4]. Ann Cavoukian, Privacy by Design: The 7 Foundational Principles, https://iab.org/wp-content/IAB-uploads/2011/03/fred_carter.pdf, 2011. (Accessed on 26th November 2017)
[5]. N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, and B. Amos, "Privacy mediators: Helping IoT cross the chasm," in Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications (HotMobile'16), New York, NY, USA: ACM, 2016, pp. 39–44.
[6]. Ukil, S. Bandyopadhyay, J. Joseph, V. Banahatti, and S. Lodha, "Negotiation-based privacy preservation scheme in internet of things

platform," in Proceedings of the First International Conference on Security of Internet of Things (SecurIT'12), New York, NY, USA: ACM, 2012, pp. 75–84.

[7]. S.-C. Cha, C.-M. Shiung, T.-C. Huang, T.-Y. Tsai, and T.-Y. Hsu, "A user-friendly privacy framework for users to achieve consents with nearby BLE devices," No. 001, Tech. Rep., Department of Information Management, National Taiwan University of Science and Technology, May 2017.

[8]. Ana Nieto, Ruben Rios, and Javier Lopez, "Digital Witness and Privacy in IoT: Anonymous Witnessing Approach," in Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, NSW, Australia, 1-4 August 2017.

[9]. Ted H. Szymanski, "Security and Privacy for a Green Internet of Things," IT Professional, Volume: 19, Issue: 5, 2017, pp. 34-41.

[10]. Dimitris Geneiatakis, Ioannis Kounelis, Ricardo Neisse, Igor Nai-Fovino, Gary Steri, and Gianmarco Baldini, "Security and privacy issues for an IoT based smart home," in Proceedings of the 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2017), Opatija, Croatia, 22-26 May 2017.

[11]. D. Yang, J. Gavigan, and Z. Wilcox-O' Hearn, "Survey of confidentiality and privacy preserving technologies for blockchains," R3, Tech. Rep., https://z.cash/static/R3_Confidentiality_and_Privacy_Report.pdf, 2016. (Accessed on 26th November 2017)

[12]. FL-100 Fingertip Pulse Oximeter - Blood Oxygen Monitor with Alarm & Case, Facelake, http://facelake.com/fl-100.html. (Accessed on 27th November 2017)