Implementation and Analysis of Fully Homomorphic Encryption in Wearable Devices

Amonrat Prasitsupparote [1] Yohei Watanabe [2] Junji Shikata [1]
[1]Graduate School of Environment and Information Sciences, Yokohama National University, Japan
[2]Security Fundamentals Laboratory, Cybersecurity Research Institute,
National Institute of Information and Communications Technology, Japan
amonrat-prasitsupparote-zp@ynu.jp, yohei.watanabe@nict.go.jp, shikata@ynu.ac.jp

ABSTRACT

Currently, wearable devices, which are known as one of the Internet of things (IoT) devices, have been widely used for healthcare systems. Most of the healthcare systems store users' healthcare data, which is encrypted by ordinary symmetric-key encryption and/or public-key encryption schemes, in a (cloud) server. However, the encrypted data needs to be decrypted for data analysis, and it means that sensitive information is leaked to the server. One promising solution is to use fully homomorphic encryption (FHE), which enables ones to perform any computation among encrypted data while keeping it encrypted. Although FHE generally requires high computational and communication costs in the theoretical sense, several researchers have implemented FHE schemes to measure their practical efficiency. In this paper, we consider a privacy-preserving protocol for healthcare systems employing wearable devices, and implement this protocol over Raspberry Pi, which is a popular single-board computer, to measure the actual efficiency of FHE over wearable devices. Specifically, we implemented the protocol by using two FHE libraries, *HElib* and *SEAL*, on Raspberry Pi and network simulator to measure both computational and communication costs in wireless body area network (WBAN). In terms of the communication overhead, our result shows that the protocol with SEAL is better than that with HElib. In particular, the protocol with SEAL has almost the same communication costs as the *trivial* protocol, which is the same protocol without encryption. On the other hand, HElib is better than SEAL regarding the running time, while SEAL can perform more homomorphic operations than HElib for the almost same plaintext-size. Therefore, HElib is suitable for applications which require small time complexity, and SEAL is suitable for applications which require many homomorphic operations.

KEYWORDS

Implementation, Healthcare system, Fully homomorphic encryption, Wearable devices, WBAN.

1 INTRODUCTION

Recently, the area of Internet of Thing (IoT) has grown significantly supporting wide range applications including medical and healthcare systems, especially wearable devices. In particular, due to the cheap prices, wearable devices have been widely used in our daily lives. Among various wearable devices sold in the market, smartwatches are most popular. Generally, people tend to use wearable devices by placing devices around (i.e., inside or outside) their bodies for monitoring their health conditions. The most popular wearable device applications use symmetric-key encryption (e.g., AES), public-key encryption or both schemes to encrypt the health data, and store it in a third party storage such as a cloud. The cloud needs to decrypt the encrypted data to perform a certain operation when a user requests it. After that the cloud will send back the result to the user in the encrypted form. In the case, the third party storage can know everything about the data since the cloud has capability of decryption. Furthermore, medical sensor devices or wearable devices in healthcare systems usually exchange the personal health record (PHR) between patients, caregivers and physicians through a cloud or a data server. This will result in information leakage issues as mentioned above. Therefore, the privacy

preserving is a critical problem in healthcare systems.
One solution to solve the privacy preserving problem in healthcare systems is the usage of homomorphic encryption (HE), in particular fully homomorphic encryption (FHE). HE allows the operation over the encrypted data (i.e., ciphertext), thereby a third party storage or a data server does not need to decrypt the ciphertext to perform the operation on the data. On the other hand, HE often requires expensive computation and much memory storage in general, and also produces a large size of ciphertexts. Due to the resource limitation and memory constrained in a wearable device, it might be difficult to apply and implement HE in a wearable device. Actually, there are several researchers who applied HE to healthcare systems. The results were not satisfactory because of a high computation time and a high communication cost [1, 2, 3, 4, 5, 6]. Currently, several researchers propose an improved homomorphic encryption algorithm [7, 8], and they claimed that it could be used in a real world. HE can be categorized into three types with respect to the number of allowed operations on the ciphertext (see detail in Section 2). In this work, we focus on the fully homomorphic encryption (FHE) which allows arbitrary operations with unlimited number of times.

1.1 Related Work

The privacy issue in healthcare systems is a challenging problem that received wide attention in the past decade. There are several reports trying to solve this issue. Specifically, Riedle et al. [9] proposed a privacy architecture in e-Health by integrating many encryption schemes: attribute-based encryption (ABE), symmetric-key encryption and threshold key sharing. Their architecture conceals patient data by encryption, and divides into two layers: the inner layer for authorized users and the outer layer for unauthorized users. However, this system is insecure with untrusted data storage or third party cloud provider.
The works that utilize HE in healthcare systems are as follows. In 2014, Bos et al. [10] presented a private predictive analysis on encrypted medical data by using homomorphic encryption based on the NTRU scheme [11]. Their scheme used only one ring element and did not use modulus switching for decreasing ciphertext expansion. Their system took sensitive medical data as input and encrypted with HE scheme, which implemented on a laptop computer (Intel Core i7-3520M), and then uploaded its encrypted data to the cloud (hosted on Microsoft's Windows Azure). The cloud ran a prediction algorithm on the encrypted data (i.e., without decryption) and returned the probability of cardiovascular disease for diabetes. This system requires the correct parameters, for example, the size of the function to be computed, the size of the inputs, and the method for encoding real data. It is difficult to choose the suitable parameters, therefore it provides a parameter selection algorithm. In contrast, the inputs of parameter selection algorithm must be defined manually and the system performance heavily relied on the parameter selection. Therefore, this system is not so practical. In addition, this work is developed by Microsoft Cryptography Research Group, which lead to the Simple Encrypted Arithmetic Library (SEAL) [12] later.

In [1, 2, 3, 4], Kocabas and colleagues proposed a general architecture for medical cyber physical system (MCPS), which consists of four layers: data acquisition, data aggregation, cloud processing, and action. This scheme employs two schemes of HE: the Paillier scheme [13] for computing the average heart rate, and the BGV scheme [14] (used in HElib library [7]) for the long QT syndrome detection. They also showed the implementation for the diagnosis of the heart problem on a workstation with dual Xeon E5-2695 Processors and 256 GB RAM. Their result showed that they could retrieve the average heart rate on the cloud close with a real time response. However, they incur high computation time and high communication cost on the user and the server side.

In 2016, Preuveneers and Joosen [5] implemented fully homomorphic encryption by using the HElib library on wearable devices for analyzing diabetics, and sharing data with physicians or other caregivers (e.g. parents of diabetic children). They converted a blood glucose value and a hypoglycemia threshold into 10-bit binary representations, and sequentially encrypted each bit of them, then stored the encrypted data on the cloud server. The caregivers can analyze blood glucose, insulin values, and other parameters while keeping the privacy of data. They implemented their scheme on three platforms: Omate True smartwatch that operates on a dual-core ARM Cortex-A7 CPU running at 1 GHz, Samsung Galaxy S4 smartphone with quad-core ARM Cortex-A7 CPU running at 1.2 GHz, and a server system with Intel Core i7-3770 processor running at 3.40 GHz. Due to the resource limitation and memory constrained in the wearable device, and computational complexity of this solution, it can send blood glucose data to the cloud server every five minutes even though ignoring the three least significant bits, and it is an unacceptable time for monitoring blood glucose in wearable device. They concluded that they demonstrated the practical feasibility of this solution but it was not so practical due to the resource limitation.

In 2017, Sun et al. [6] presented an architecture of mobile healthcare systems, which consists of four sections: wearable device, preprocessing, cloud server, and physician diagnosis. This solution was defined as three secure medical computations: the average heart rate, the long QT syndrome detection, and the chi-square tests. This work is similar to the Kocabas's work [4], however, the difference lies in the usage of the homomorphic encryption library. Kocabas's work employed HElib library, while Sun's work employed Dowlin's scheme. In fact, both libraries are FHE based on BGV scheme. Note that Dowlin's scheme becomes a part of SEAL project later. They evaluated their solution on a PC with Intel Core i5-3470 processor running at 3.20 GHz and 8 GB RAM. Their result shows that it produces only one ciphertext for the average heart rate function, and one multiplication operation for the long QT syndrome detection function. They also compared the implementation time of the long QT syndrome detection in Kocabas's work and their protocol, and this result showed that their protocol was faster than Kocabas's scheme. Finally, they concluded that their scheme was better than the Kocabas's scheme. This was done by comparing efficiency of SEAL library (this work) and HElib library (i.e., Kocabas's work). Furthermore, they first implemented chi-square tests for observing the probability of varicose veins are relevant to overweight.

Costache and Smart [15] extended the work [16] and showed a comparison result of four HE schemes, FV, YASHE, BGV and NTRU. They applied the four schemes to the same API and the same optimization. They also investigated applying modulus switching to the scale invariant schemes, and considered the plaintext modulus, level bound and security parameters. They presented the results of all schemes in various parameter sets, while fixing the security level of $k = 80$ bits, a Hamming weight of $h = 64$ and a ring constant of $c_m = 1.3$. They also presented the relation between the large plaintext-space $p \approx 2^{32}$, the number of levels L, and the ciphertext-size, and concluded that the BGV scheme appears to be more efficient for large plaintext moduli, while the YASHE scheme seems more efficient for small plaintext moduli. Note that the BGV scheme means HElib library, and the YASHE scheme means SEAL library.

1.2 Our Contribution

As can be seen in Kocabas's work [1, 2, 3, 4], FHE produces large ciphertexts, which lead to high communication costs on both the user and server sides in healthcare systems. Our work has been motivated by this problem. Note that all related works were done by estimating only the time complexity, even though the real implementation in Kocabas's work shows significance of the overhead in communication in FHE. Therefore, estimating the communication overhead over the wearable-device network is important as well as time complexity. Note that a group of short-range wearable devices on, in, or around human bodies is based on IEEE 802.15.6 standard [17], which is called the wireless body area network (WBAN). Consequently, the first part of our work investigates the communication overhead of healthcare systems using FHE over WBAN. Furthermore, several researchers presented the practical FHE library [7, 8], and they claimed that it could be used in a real world. [1, 2, 3, 4, 5] showed implementations of healthcare systems using the HElib library over a PC and a wearable device, and [10, 6] implemented a healthcare system using the SEAL library on a PC. [5] implemented a healthcare system using the HElib library over a smartwatch. In addition, Costache and Smart's work [15] dealt with implementations of various FHE schemes for general systems on PCs. However, there is no research on estimating the efficiency of HElib and SEAL libraries on restricted resource devices in a general setting, which is the second part of our work.

Specifically, the contribution of this paper is as follows:

- We investigate the communication overhead of a certain privacy-preserving protocol using FHE for healthcare system over WBAN. We implement the protocol by using two FHE libraries, *HElib* and *SEAL*, with a network simulator to measure communication costs over WBAN. Our result shows that the protocol with SEAL is better than that with HElib. In particular, the former has almost the same communication costs as the *trivial* protocol, which is the same protocol without considering privacy (i.e., a protocol without FHE).

- We evaluate efficiency of two FHE libraries, *HElib* and *SEAL* on a restricted resource device. We implement them on a PC supposed to be a cloud server, and Raspberry Pi supposed to be a wearable device such as a smartphone or any restricted resource device over WBAN. Our result shows that HElib is better in terms of running time than SEAL, while SEAL can perform more homomorphic operations than HElib for the almost same plaintext size. Therefore, HElib is suitable for applications which require small time complexity, and SEAL is suitable for applications which require a lot of homomorphic operations.

2 FULLY HOMOMORPHIC ENCRYPTION (FHE)

Homomorphic encryption (HE) is a public-key encryption that enables us to perform an arithmetic or logical operation on ciphertexts without decrypting it. Generally, a HE scheme consists of four polynomial time algorithms: $KeyGen, Enc, Dec,$ and $Eval$. $KeyGen$ is a probabilistic algorithm for generating a key-pair, a public key pk and a secret key sk. Enc is an algorithm to encrypt a plaintext m and to output a ciphertext c. Dec is an algorithm to decrypt a ciphertext c and to output the plaintext m. In fact, $KeyGen, Enc, Dec$ are the same as those in the traditional public key encryption scheme, however $Eval$ is a special algorithm included in a HE scheme. An evaluation algorithm $Eval$ takes two ciphertexts of two plaintexts m_1 and m_2 respectively, and an operation $\star$ as input, and it outputs an evaluated ciphertext $\tilde{c} := \mathit{Eval}_{pk}(\mathit{Enc}_{pk}(m_1), \mathit{Enc}_{pk}(m_2), \star)$, which satisfies $\mathit{Dec}_{sk}(\tilde{c}) = m_1 \star m_2$. HE can be categorized into three types with respect to the number of allowed operations on the ciphertext as follows:

- In Partially Homomorphic Encryption (PHE), $Eval$ can perform only one type of operation (e.g., either addition or multiplication), though the number of operations performed is unlimited. For instance, the Paillier scheme [13] based on composite residuosity problem [18] allows only addition, and was used in [4] to calculate the average heart rate on the cloud.

- In Somewhat Homomorphic Encryption (SHE), $Eval$ can perform two kinds of operations such as both addition and multiplication, though the number of one of the two operations is limited. SHE schemes include BGN [19] based on a subgroup decision problem [20]. In the BGN scheme, the number of allowed addition-operations is unlimited, while the multiplication operation is allowed only one time.

- Fully Homomorphic Encryption (FHE) was proposed by Gentry [21] in 2009 where $Eval$ can perform any operations (i.e., both addition and multiplication), and the number of performed operations is unlimited. It was constructed based on ideal lattices, and has a massive overhead in computation and memory. FHE has a lot of attractive applications, especially in cloud environments, and therefore a variety of FHE schemes have been proposed after Gentry's work. There are four main categories in FHE after Gentry's work: Ideal lattice-based FHE [21], FHE over integers [22, 23], FHE from the learning with errors (LWE) assumption [24, 25, 26], and NTRU-like FHE [11, 27]. In FHE, whenever a homomorphic operation is applied, some noise will be added into the ciphertext and the ciphertext-size increases. When the noise grows over the limitation, the decryption algorithm cannot correctly decrypt ciphertexts. To resolve this problem, there is a bootstrapping technique to reduce the noise, and get a fresh ciphertext from the noisy ciphertext corresponding to the same plaintext. After applying bootstrapping, a homomorphic operation can be applied to the fresh ciphertext as long as the noise is within the limitation.

Table 1. The property of FHE libraries

Name	**HElib [7]**	**SEAL [12]**
Base Scheme	BGV [14]	BFV [28]
Language	C/C++	C++/.NET
Required Libraries	GMP [29], NTL [30]	No
Bootstrapping	Yes	No

In this paper, we focus on two well-known FHE libraries: HElib and SEAL, which we summarize in Table 1. The first is *HElib* [7], an open source library implemented by Halevi and Shoup in 2014, which is based on the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [14]. Halevi and Shoup also applied several techniques: a ciphertext packing proposed by Smart and Vercauteren [31], an optimization for homomorphic evaluation proposed by Gentry, Halevi, and Smart [32], and a noise management by bootstrapping [33]. This library is written in C++ and has several parameters which effect to the performance and security level, thus it is difficult to choose the suitable parameters for non-experts. Moreover, it requires two prerequisite libraries: GNU Multiple Precision Arithmetic (GMP) library [29] and NTL mathematical library [30] (version 10.0.0 or higher). However, it is a low-level implementation, thereby it was applied in various areas. The current version is 1.3, which is available at github [34].
The second one, *Simple Encrypted Arithmetic Library (SEAL)* [12], was developed by Cryptography Research Group at Microsoft Research in 2015. It is based on the Brakerski/Fan-Vercauteren scheme (BFV) [28], and it is a SHE scheme since bootstrapping is not yet supported. The goal of this library is to be easily used by both crypto experts and non-experts like in bioinformatics. Accordingly, there are automatic parameter selection and noise estimator tools for non-experts, and this library does not require any external dependencies. It is written in C++, and contains .NET wrappers for the public API, thereby it can be compiled on various platforms. The current version of SEAL is 2.3.1. Although this version does not provide bootstrapping, the developer encourages to use parameter selection and noise estimator tools instead. It needs the Microsoft Research License Agreement to use and be free for the research purpose.

3 PRIVACY PRESERVING PROTOCOL FOR WEARABLE DEVICES IN HEALTHCARE SYSTEMS

We assume that there are a user (e.g., a patient), a user's smartphone, a cloud server, and

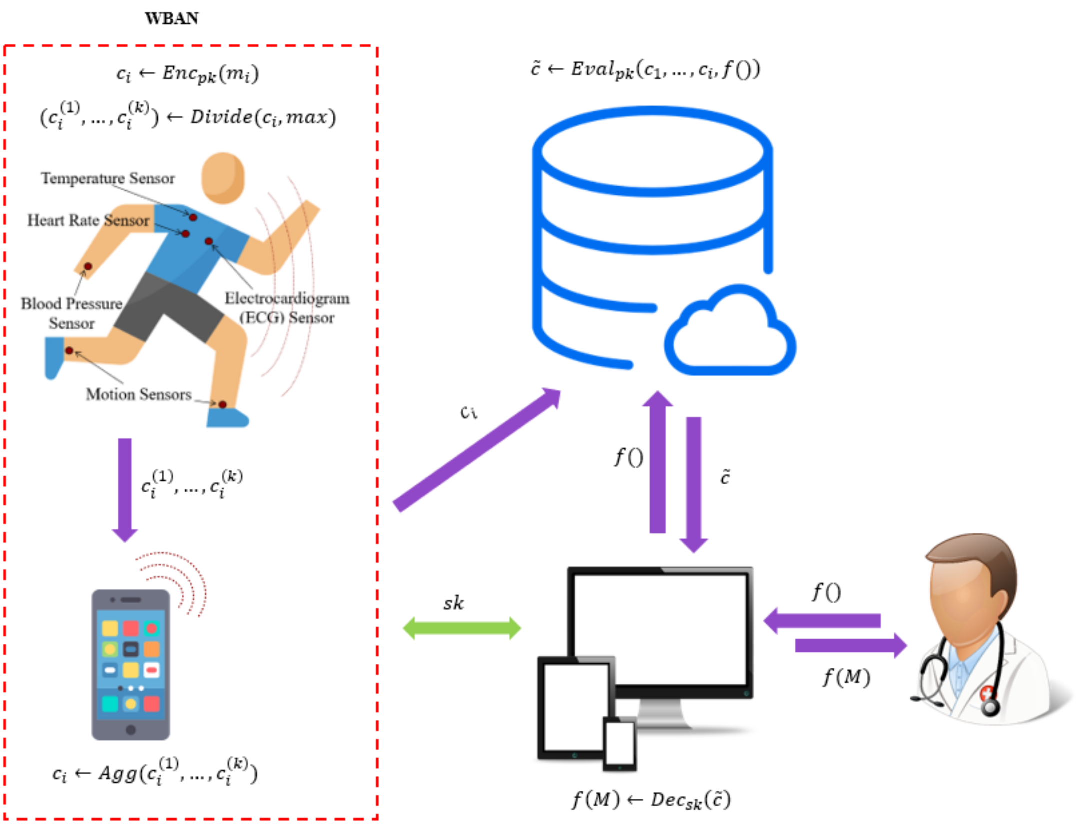
WBAN
$c_i \leftarrow Enc_{pk}(m_i)$
$(c_i^{(1)},\dots,c_i^{(k)}) \leftarrow Divide(c_i, max)$
Temperature Sensor
Heart Rate Sensor
Blood Pressure Sensor
Electrocardiogram (ECG) Sensor
Motion Sensors
$c_i^{(1)},\dots,c_i^{(k)}$
$c_i \leftarrow Agg(c_i^{(1)},\dots,c_i^{(k)})$
$\tilde{c} \leftarrow Eval_{pk}(c_1,\dots,c_i,f())$
c_i
$f()$
$\tilde{c}$
sk
$f()$
$f(M)$
$f(M) \leftarrow Dec_{sk}(\tilde{c})$

Figure 1. A privacy preserving protocol for wearable devices in healthcare systems.

a caregiver (e.g., a physician). In Fig. 1, the green arrow means a secure wireless channel, and other arrows mean insecure channels, which can be wired or wireless channels. The red dotted square means a wireless body area network (WBAN) following IEEE 802.15.6 [17]. A working group of IEEE defines IEEE 802.15.6 that specifies the standard for low-power and short-range wireless devices on, in, or around human bodies, called WBAN. This standard also means a wireless network of wearable devices in healthcare systems. Currently, WBAN consists of one or more wireless medical sensor devices, and sink nodes (i.e., gateway nodes). A sink node can be a smartphone, a PC, or a high performance sensor node, however, it must be connected in a wireless environment. WBAN in our protocol consists of wearable devices and a smartphone as a sink node which communicate through Wi-Fi. Each wearable device may contain single or multiple medical sensors depending on its aim, however in this paper we assume there are n sensor nodes $SN_1,\dots,SN_n$ in total on a user's body. A cloud server stores the health data, and performs operations then sends the result back to a caregiver. Our protocol consists of the following four phases:

1) **Key generation:** We omit this phase in Fig. 1 for simplicity. At the first time of using a wearable device, a user calls a key generation algorithm $KeyGen$ to generate a key-pair (pk, sk) through the wearable device's application on the user's sink node (e.g., smartphone). The sink node broadcasts pk to all devices over WBAN, and keeps sk. All devices in WBAN obtain pk, and store it in their memory. In addition, the sink node can send sk to a caregiver's application through a secure wireless channel. On the other hand, a caregiver can call $KeyGen$ on behalf of a user though an application on his/her PC, tablet or smartphone, and the application can send sk to the sink node through a secure wireless channel.

2) **Encryption and transmission:** When

each sensor SN_i in a wearable device reads the health data m_i, it was encrypted by the encryption algorithm Enc immediately. We write this operation as $c_i \leftarrow Enc_{pk}(m_i)$. In case that the size of c_i is greater than the maximum packet size,[1] we divide the ciphertext c_i into k pieces for some k. We write this operation as $(c_i^{(1)}, \ldots, c_i^{(k)}) \leftarrow Divide(c_i, max)$, where max is the maximum packet size. The wearable device stores the divided ciphertexts $c_i^{(j)}$ in transmission's buffer, and transmits $c_i^{(j)}$ in each time slot. When the sink node receives all $c_i^{(j)}$ $(1 \leq j \leq k)$, it reconstructs c_i from them. We write this operation as $c_i \leftarrow Agg(c_i^{(1)}, \ldots, c_i^{(k)})$. After that, the sink node uploads the ciphertext c_i to the cloud server.

3) **Homomorphic operation:** A caregiver's request is denoted by $f()$, and we assume that every $f()$ is expressed by addition and/or multiplication operations. After the cloud server receives the request $f()$, it runs $Eval$ and outputs the resulting ciphertext $\tilde{c}$ to the requester. The noise in ciphertexts becomes to be large whenever $Eval$ is applied. When the noise is close to the limit, the cloud server must perform bootstrapping to reduce the noise, and get a fresh ciphertext having the same underlying plaintext. After applying bootstrapping, a fresh ciphertext allows us to continue to perform homomorphic operations as long as the noise is within the limitation.

4) **Decryption:** A caregiver's device decrypts the ciphertext $\tilde{c}$ by the decryption algorithm Dec using a secret key sk. We write this operation as $f(M) \leftarrow Dec_{sk}(\tilde{c})$, where M denotes the health data stored in the cloud with the encrypted form. The caregiver finally obtains the result $f(M)$ for his/her request $f()$ on the data M.

[1]We consider the maximum packet size based on IEEE 802.15.6 [17].

4 ANALYSIS OF COMMUNICATION OVERHEAD IN WBAN

In [35], it is stated: "The total number of packets are to be transferred or transmitted from one node to another, which is known as the communication overhead; It includes the overhead of routing process, routing table and packet preparation in a sensor node". This implies the communication cost treated in Kocabas's work [4]. Kocabas also mentioned that the communication overhead was concern in systems. Therefore, we implement our protocol with a network simulator to investigate the communication overhead in WBAN on a PC with Intel Core i7 processor running at 4.0 GHz and 32 GB of RAM where it is running on Ubuntu 64-bit operating system.

4.1 Experimental Setup

Xian et al. [36] presented the comparision of several major wireless sensor network (WSN) simulators. The results show that OMNET++ [37] is better than NS2 and OPNET in terms of execution time and memory usage in simulating WSN. In addition, OMNET++ was widely used in this research area. In 2007, Australia's Information and Communications Technology Research Centre of Excellence (NICTA) published a simulator for WSN, WBAN, and more generally for networks of low-power embedded devices, which is called *Castalia*. It is based on the OMNET++ platform with realistic node behaviours and Baseline MAC for Body Area Networks (BAN), following IEEE 802.15.6. The WBAN testbed of Castalia collected data from the real wearable sensors on human bodies in daily life activities such as walking, running, jogging, and sleeping. It is an open source and available on github [38]. Our experiment simulates WBAN through OMNET++ version 4.6 and Castalia version 3.3.
We simulate WBAN through OMNET++ and Castalia with parameters in Table 2. Firstly, we limit the number of sensor nodes from 2 to 16, and the assigned Medium Access Control protocol (MAC) is Baseline BAN MAC, which is

Table 2. Simulation parameters.

Parameters	**Value**
Number of sensor nodes	$2-16$
Medium Access Control protocol (MAC)	Baseline BAN MAC
Read data interval	30s
Maximum packet size	2kb
Delayed limit	30s
Packet rate	5s
Simulation time	3600s

important for the node's behavior. Generally, medical sensor devices generate many small packets in a short time interval, thereby we suppose that each node reads sensitive data every 30 seconds. If the node's buffer is full, it skips reading sensitive data, and will read it again in a next period. Moreover, we assume that the maximum packet size is 2kb, and the delay limit in transmission is 30 seconds, because we assume each node reads the data every 30 seconds. Each node sends a packet every 5 seconds, and limit our simulation time to 3600 seconds.

We implement our protocol with two FHE libraries: HElib and SEAL, and compare their performance with NoEncrypt scheme. The NoEncrypt scheme is a trivial protocol without encryption, where all packets are transimtted in cleartext. One of disadvantages in HElib library is the parameter selection for non-experts because it has a lot of parameters and some parameters have a relationship, however the several appropriate parameters are suggested in [7]. Therefore, this implementation use their suggested parameters. In contrast, SEAL has a tool for automatic parameter selection for non-experts: a user only defines the plaintext-size (i.e., plaintext modulus in SEAL library) and runs an automatic parameter selection tool. Our experiment assigns a security parameter to be 110-bit for all libraries. It is known that the multiplication operation causes a large noise to ciphertexts compared to addition operation, a user cannot use a multiplication operation if the plaintext-space is not large enough. Therefore, our experiment selects the minimum plaintext-size such that a homomorphic operation for multiplications can be executed at least one time. Taking into the above conditions, we have selected the following parameters: the plaintext-space of HElib is GF(p) with $p = 1693$, that of SEAL is GF(2^{10}) (i.e., its size is $|GF(2^{10})| = 1024$).

This simulation assigns a sink node SN_0, and SN_0 receives a packet from other nodes $SN_1, SN_2, ..., SN_n$ every 5 seconds, where n is the number of sensor nodes. We consider four measurements to evaluate communication overhead as follows:

1) What is the number of packets per ciphertext?: This is evaluated as follows. Let J be the number of ciphertexts by which SN_0 could reconstruct by receiving all pieces of the ciphertext from some sensor nodes, and we do not count ciphertexts such that SN_0 could not reconstruct them. Suppose that such J ciphertexts are denoted by $C(1), C(2), \ldots, C(J)$ and that j-th ciphertext $C(j)$ $(1 \leq j \leq J)$ was divided into k_j packets in transmission. Then, we calculate *the number of packets per ciphertext* by $\sum_{j=1}^{J} k_j/J$.

2) What is the number of packets transmitted from each sensor node?: This is evaluated as follows. Let K be the amount number of packets which arrived at SN_0 from I sensor nodes in total minus one because of fixed one sink node. Note that K includes a re-transmitted packet. Then, we calculate *the number of packets transmitted from each sensor node* by the average K/I.

3) What is the number of delayed packets from each sensor node?: We define that a packet is *delayed*, if the time difference between the time when the packet was created and the time when the packet arrived at SN_0 is more than 30 seconds. This is reasonable in our simulation, since we assume each node reads data every 30 seconds.

4) How many times a packet was successfully transmitted?: This is investigated at

the MAC level. SN_0 must receive all packets from other nodes, and aggregates them to reconstruct a ciphertext c_i. However, in the process of transmission, there would be packet loss from some node, and then the node has to re-send such a packet until SN_0 will successfully receive it. Moreover, if SN_0 is in the collision state, some packets must be re-sent many times. By taking into account such a situation, we count a number of times that the packet was successfully transmitted, namely a packet was successfully transmitted in the first time (1st-try), a packet was successfully transmitted in the second time (2nd-try), ..., and a packet was successfully transmitted in the sixth time or more (6 or more tries).

4.2 Simulation results

Firstly, we investigate the reasonable number of sensor nodes for our protocol by comparing that of NoEncrypt scheme. In Fig. 2, the x-axis means the number of nodes in WBAN and the y-axis means the number of delayed packets in average. It can be seen that the number of delayed packets in our protocol with all libraries are almost same as that in NoEncrypt scheme, when WBAN consists of two nodes. However, the number of delayed packets in our protocol with SEAL library is close to that in NoEncrypt scheme, when WBAN consists of two to ten nodes. In fact, there is a moderate difference of the number of delayed packets between our protocol with SEAL library and NoEncrypt scheme, when WBAN consists of eight nodes. Thereby, the reasonable number of sensor nodes for our protocol is not over ten nodes. In addition, Castalia recommends that the number of nodes in WBAN is six. According to this result and recommendation by Castalia, we change the number of sensor nodes in our simulation to six.
Next, we observe the number of packets per ciphertext and the number of packets transmitted from each sensor node in Table 3. It can be seen that NoEncrypt method has only one packet per ciphertext, however our protocol

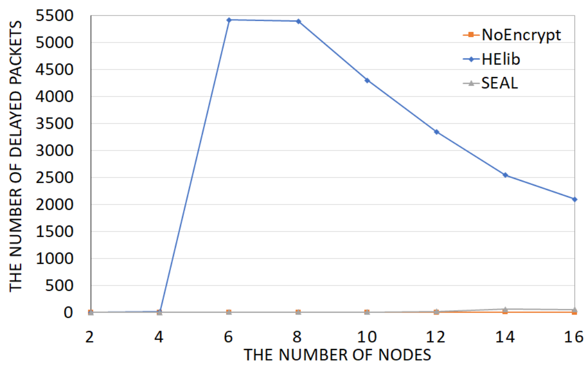

Figure 2. The number of delayed packets with vary the number of nodes.

Table 3. The number of packets per ciphertext and the number of packets transmitted from each sensor node.

Name	Packets/ciphertext	Packets transmitted
NoEncrypt	1	120.02
HElib	80	9601.78
SEAL	17	2040.02

with HElib has the highest number of packets per ciphertext, which means HElib library produces the largest ciphertext-size. In contrast, the number of packets per ciphertext of our protocol with SEAL is close to that of NoEncrypt scheme. The tendency of the number of packets per ciphertext is the same as the number of packets transmitted from each sensor node. The number of packets transmitted from each sensor node of our protocol with HElib is the highest, while that of SEAL is close to that of NoEncrypt scheme.

Table 4. The number of delayed packets from each sensor node.

Name	NoEncrypt	HElib	SEAL
SN_1	0.0	7234.8	0.0
SN_2	0.3	172.4	5.1
SN_3	0.1	6206.8	1.7
SN_4	0.4	7662.6	6.8
SN_5	0.3	5761.6	6.0
Average	0.2	5407.6	3.9

Table 4 shows the number of delayed packets from each sensor node. It can be seen that the number of delayed packets from each sensor node of our protocol with HElib is enormous more than that of other methods, and that of SEAL is close to that of NoEncrypt. The number of delay packets in SN_1 of our protocol with SEAL and NoEncrypt scheme is 0, which means SN_0 received all packets from SN_1. According to this situation and the result in Fig. 2, we confirm that the best number of sensor nodes for our protocol is only two.

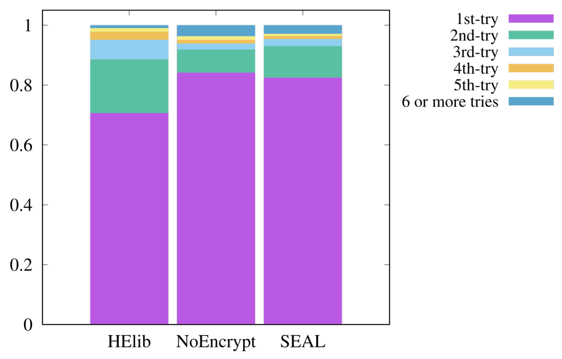

Figure 3. A number of times that the packet was successfully transmitted by expression with fractions of 1.

In Fig. 3, we depict a number of times that the packet was successfully transmitted. The x-axis means the schemes, NoEncrypt and our protocols with HElib and SEAL libraries. The y-axis means a number of times that the packet was successfully transmitted by expression with fractions of 1. It is clearly seen that the result of our protocol with SEAL is almost the same as NoEncrypt scheme. Moreover, the packets was successfully transmitted in the first time (1st-try) of NoEncrypt scheme and our protocol with SEAL approximately 80%, while that of HElib approximately 70%. This result corresponds to the result in Table 4 and Fig. 2.

As a result, it can be seen that our protocol with HElib produces much communication overhead in WBAN. However, our protocol with SEAL produces few communication overhead close to NoEncrypt scheme. Therefore, we can conclude that our protocol with SEAL is best in terms of communication cost, and the best number of sensor nodes for our protocol is two, however our protocol can efficiently perform if the number of sensor nodes are not over six.

5 ANALYSIS OF EFFICIENCY FOR FHE

In this section, we investigate efficiency of HElib and SEAL libraries. As explained in Section 3, our protocol has three kinds of components, wearable devices (for sensor nodes), smartphone (for a sink node and a caregiver's device), and a cloud server. This experiment uses a PC with Intel Core i7 processor running at 4.0 GHz and 32 GB of RAM which is supposed to be a cloud server, and a Raspberry Pi Model B+ v1.2 with ARM11 at 700 MHz and 512 MB SDRAM which is supposed to be a wearable device or a smartphone. Nowadays, smartphones in market have much higher performance than this Raspberry Pi, and we can expect that implementation results on smartphone would be much better than our implementation results on this Raspberry Pi. In addition, the hardware of this Raspberry Pi is almost the same as those of cheap wearable devices in market, thereby this Raspberry Pi can be used instead of implementation in wearable devices.

5.1 Experimental Setup

We export health data from the network simulator in the previous experiment (see Section 4) and use it as the input in this experiment as well. The previous experiment used the minimum plaintext-space that can allow us to perform homomorphic operations for both addition and multiplication. However, this experiment observes the running time for computing homomorphic operations and that for computing bootstrapping, thus our experiment should use the same plaintext-size for fair comparison. Therefore, this experiment uses the plaintext-sapce is 1024 in SEAL library (the same as the previous experiment in Section 4.1); and the plaintext-space of HElib must be a prime number (or a power of a prime), thereby we use the size 1021 which is close to 1024.

5.2 Comparison Results

Firstly, we observe the running time (in milliseconds) in each algorithm of HElib and

Table 5. The running time (Milliseconds)

		KeyGen	Encryption	Decryption	Addition	Multiplication	Bootstrapping
PC	HElib	0.760058	0.032094	0.013368	0.000094	0.066178	85.323830
	SEAL	1.930100	2.342723	0.177101	0.005329	2.187750	-
Raspberry Pi	HElib	79.933075	2.084733	1.258043	0.006370	4.707492	7,846.207000
	SEAL	181.319900	229.979548	46.673325	0.920642	480.622600	-

SEAL libraries on PC and Raspberry Pi in Table 5. It can be seen that HElib library is faster than SEAL library on both platforms. The bootstrapping takes a long time than other algorithms. Note that SEAL library does not provide a bootstrapping function. We can explicitly observe that the running time for performing a homomorphic operation for multiplication is much more than that of a homomorphic operation for addition in each library. Additionally, Raspberry Pi takes much more time than PC in every algorithm, however the running time of HElib on Raspberry Pi is acceptable for the practical use.

Table 6. The ciphertext-size of HElib library before/after using bootstrapping (bytes)

	PC	Raspberry Pi
Before	84,251.4	63,486.8
After	29,138.6	29,156.8

Moreover, we observe the ciphertext-size in HElib library before/after using bootstrapping on PC and Raspberry Pi in Table 6. Our results in Section 4.2 show that the large ciphertexts lead to a high communication overhead. The usage of bootstrapping can reduce the ciphertext-size over 60% on both platforms.

Table 7. The maximum number of allowed homomorphic operations

	Plaintext-space	Addition	Multiplication
HElib	1021	26	0
	1693	44	1
SEAL	1024	510	1

We also investigate the maximum number up to which homomorphic operations are allowed to apply, and the results are summarized in Table 7. Under the condition that plaintext-size in each library is almost the same (i.e., 1021 in HElib, and 1024 in SEAL), SEAL provides the largest number up to which homomorphic operations for addition or multiplication are allowed. In particular, HElib cannot allow a homomorphic operation for multiplication for the selected plaintext-size. Afterwards, we have increased the size of plaintexts in HElib so that we can apply a homomorphic operation for multiplication at least one time, and the resulting size of plaintexts in HElib is 1693.

As a result, HElib is better than SEAL in terms of running time, while our protocol with SEAL is better than our protocol with HElib in terms of the communication overhead in WBAN. In addition, SEAL can provide us the largest number up to which homomorphic operations are applied than HElib when we regard those as SHE schemes. However, HElib provides bootstrapping for refreshing the ciphertexts, and hence we can continue to use homomorphic operations. Furthermore, in HElib, we can observe the running time of bootstrapping is much more than those of other algorithms, whereas the ciphertext-size of HElib is reduced over 60% after applying bootstrapping.

6 CONCLUSION

We investigated the communication overhead in WBAN when using FHE in the privacy preserving protocol for healthcare. We implemented the protocol by using two FHE libraries, *HElib* and *SEAL*, on a PC, Raspberry Pi and network simulator (OMNET++ and Castalia) to measure both computational and communication costs in WBAN. In terms of the communication overhead, our result shows that the protocol with SEAL is better than that with HElib. In particular, the protocol with SEAL has almost the same communication costs as the *trivial* protocol, which is the same protocol without encryption. However, HElib is better than SEAL in terms of the running time, while SEAL can perform more homomorphic operations than HElib for the almost same plaintext-size as SHE schemes. Therefore, HElib is suitable for applications which require small time complexity, and SEAL is suitable for applications which require many homomorphic operations. It would be interesting to investigate power consumption required in the protocol by implementation, and it will be our future work.

REFERENCES

[1] O. Kocabas, T. Soyata, J. Couderc, M. Aktas, J. Xia, and M. Huang, "Assessment of cloud-based health monitoring using homomorphic encryption," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Oct 2013, pp. 443–446.

[2] v. Kocabaş and T. Soyata, "Medical Data Analytics in the Cloud Using Homomorphic Encryption," *Handbook of Research on Cloud Infrastructures for Big Data Analytics*, pp. 471–488, 2014.

[3] O. Kocabas and T. Soyata, "Utilizing Homomorphic Encryption to Implement Secure and Private Medical Cloud Computing," in *2015 IEEE 8th International Conference on Cloud Computing*, Jun. 2015, pp. 540–547.

[4] O. Kocabas, T. Soyata, and M. K. Aktas, "Emerging security mechanisms for medical cyber physical systems," vol. 13, no. 3, May 2016, pp. 401–416.

[5] D. Preuveneers and W. Joosen, "Privacy-enabled Remote Health Monitoring Applications for Resource Constrained Wearable Devices," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ser. SAC '16. New York, NY, USA: ACM, 2016, pp. 119–124.

[6] X. Sun, P. Zhang, M. Sookhak, J. Yu, and W. Xie, "Utilizing fully homomorphic encryption to implement secure medical computation in smart cities," *Personal and Ubiquitous Computing*, vol. 21, no. 5, pp. 831–839, Oct 2017.

[7] S. Halevi and V. Shoup, "Algorithms in helib," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 554–571.

[8] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Manual for using homomorphic encryption for bioinformatics," *Proceedings of the IEEE*, vol. 105, no. 3, pp. 552–567, March 2017.

[9] B. Riedl, V. Grascher, S. Fenz, and T. Neubauer, "Pseudonymization for improving the privacy in e-health applications," in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, Jan 2008, pp. 255–255.

[10] J. W. Bos, K. Lauter, and M. Naehrig, "Private predictive analysis on encrypted medical data," *Journal of Biomedical Informatics*, vol. 50, pp. 234–243, Aug. 2014.

[11] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '12. New York, NY, USA: ACM, 2012, pp. 1219–1234.

[12] "Simple Encrypted Arithmetic Library - SEAL Crypto." [Online]. Available: https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/

[13] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[14] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: ACM, 2012, pp. 309–325.

[15] A. Costache and N. P. Smart, "Which ring based somewhat homomorphic encryption scheme is best?" in *Topics in Cryptology - CT-RSA 2016*, K. Sako, Ed. Cham: Springer International Publishing, 2016, pp. 325–340.

[16] T. Lepoint and M. Naehrig, "A comparison of the homomorphic encryption schemes fv and yashe," in *Progress in Cryptology – AFRICACRYPT 2014*, D. Pointcheval and D. Vergnaud, Eds. Cham: Springer International Publishing, 2014, pp. 318–335.

[17] "IEEE 802.15.6-2012 - IEEE Standard for Local and metropolitan area networks - Part 15.6:

Wireless Body Area Networks." [Online]. Available: http://standards.ieee.org/findstds/standard/802.15.6-2012.html

[18] T. Jager, *The Generic Composite Residuosity Problem*. Wiesbaden: Vieweg+Teubner Verlag, 2012, pp. 49–56. [Online]. Available: https://doi.org/10.1007/978-3-8348-1990-1_5

[19] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Theory of Cryptography, TCC 2005*, J. Kilian, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 325–341.

[20] K. Gjøsteen, *Subgroup membership problems and public key cryptosystems*, 2004. [Online]. Available: https://brage.bibsys.no/xmlui/handle/11250/249681

[21] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 169–178.

[22] J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi, "Fully homomorphic encryption over the integers with shorter public keys," in *Advances in Cryptology – CRYPTO 2011*, P. Rogaway, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 487–504.

[23] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology – EUROCRYPT 2010*, H. Gilbert, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–43.

[24] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Advances in Cryptology – CRYPTO 2011*, P. Rogaway, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 505–524.

[25] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Advances in Cryptology – CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 868–886.

[26] Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) LWE," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.

[27] K. Rohloff and D. B. Cousins, "A scalable implementation of fully homomorphic encryption built on NTRU," in *Financial Cryptography and Data Security, FC 2014*, R. Böhme, M. Brenner, T. Moore, and M. Smith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 221–234.

[28] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Report 2012/144, 2012, https://eprint.iacr.org/2012/144.

[29] "The GNU MP Bignum Library." [Online]. Available: https://gmplib.org/

[30] "NTL: A Library for doing Number Theory." [Online]. Available: https://www.shoup.net/ntl/

[31] N. P. Smart and F. Vercauteren, "Fully homomorphic simd operations," *Designs, Codes and Cryptography*, vol. 71, no. 1, pp. 57–81, Apr 2014.

[32] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Advances in Cryptology – EUROCRYPT 2012*, D. Pointcheval and T. Johansson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 465–482.

[33] S. Halevi and V. Shoup, "Bootstrapping for helib," in *Advances in Cryptology – EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 641–670.

[34] S. Halevi, "HElib: An Implementation of homomorphic encryption." [Online]. Available: https://github.com/shaih/HElib

[35] N. Kumar and Y. Singh, "Routing Protocols in Wireless Sensor Networks," *Handbook of Research on Advanced Wireless Sensor Network Applications, Protocols, and Architectures*, pp. 86–128, 2017. [Online]. Available: https://www.igi-global.com/chapter/routing-protocols-in-wireless-sensor-networks/162116

[36] X. Xian, W. Shi, and H. Huang, "Comparison of OMNET++ and other simulator for WSN simulation," in *2008 3rd IEEE Conference on Industrial Electronics and Applications*, Jun. 2008, pp. 1439–1443.

[37] "OMNeT++ Discrete Event Simulator - Home." [Online]. Available: https://omnetpp.org/

[38] T. Boulis, “Castalia: An OMNeT-based simulator for low-power wireless networks such as Wireless Sensor Networks and Body Area Networks.” [Online]. Available: https://github.com/boulis/Castalia