International Journal of Digital Information and Wireless Communications (IJDIWC) 8(4): 288-303
The Society of Digital Information and Wireless Communications, 2018 ISSN: 2225-658X (Online); ISSN 2412-6551 (Print)

# Implementation and Analysis of Fully Homomorphic Encryption in Resource-Constrained Devices

Amonrat Prasitsupparote [1] Yohei Watanabe [2] Junichi Sakamoto [1]
Junji Shikata [1,3] Tsutomu Matsumoto [1,3]
[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
[3]Institute of Advanced Sciences, Yokohama National University, Japan
amonrat-prasitsupparote-zp@ynu.jp, yohei.watanabe@nict.go.jp, sakamoto-junichi-mb@ynu.jp,
shikata@ynu.ac.jp, tsutomu@ynu.ac.jp

## ABSTRACT

Currently, resource-constrained devices, which are known as one of the Internet of things (IoT) devices, have been widely used for healthcare systems. Most healthcare systems store users' health 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 would be 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, though FHE generally requires high computational and communication costs in the theoretical sense.

In this paper, we investigate practical feasibility of FHE in resource-constrained devices for healthcare systems. First, we define a privacy-preserving protocol for healthcare systems, and implement it on PC and Raspberry Pi by using a network simulator to measure its communication overhead, computational cost, and energy consumption over wireless body area network (WBAN). For this implementation, we suppose PC and Raspberry Pi as a cloud server and a resource-constrained device such as a smartphone or a wearable device, respectively. We use two FHE libraries, *HElib* and *SEAL*, for the implementation. Our result shows that the protocol with SEAL is better than that with HElib in terms of the communication overhead and energy consumption in transmission. 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. Furthermore, the energy to execute each algorithm in the libraries is very small compared to the energy required in transmission. SEAL produces smaller sizes of ciphertexts than HElib, and therefore consumes few energy consumptions. As a result, we observe that both HElib and SEAL would be used on restricted resource devices, and in particular, SEAL would be more suitable for practical use in resource-constrained devices from our analysis.

## KEYWORDS

Fully homomorphic encryption, Healthcare system, Implementation, Resource-constrained devices, WBAN.

## 1 INTRODUCTION

Recently, the area of a resource-constrained device has grown significantly supporting wide range applications including medical and healthcare systems, especially wearable devices. Due to the cheap prices, wearable devices have been widely used in our daily lives. In particular, smartwatches are popular among various wearable devices sold in the market. Generally, people tend to use wearable devices by placing the devices around (i.e., inside or outside) their bodies for monitoring their health conditions. The popular wearable-device applications use symmetric-key encryption and/or public-key encryption schemes to encrypt the health data and store it in third-party storage such as a cloud. In such a system, the cloud sometimes needs to analyze the data according to users' request. To do so, the cloud is allowed to decrypt the encrypted data, and thus can have access to the data. Furthermore, medical sensor devices or wearable devices in healthcare systems usually exchange the personal health record (PHR) among patients, caregivers, and physicians through a cloud or a data server. This will result in information leakage issues as mentioned above. Therefore, privacy-preserving is important in healthcare systems.

One solution to solve the privacy-preserving problem in healthcare systems is the usage of homomorphic encryption (HE), especially fully homomorphic encryption (FHE). HE allows ones to perform algebraic operations over encrypted data, and therefore a third party such as a data server does not need to decrypt the encrypted data to perform the operation. On the other hand, in general HE often requires expensive computation and large memory storage due to its large ciphertexts. Due to the resource limitations and constrained memories of wearable devices, it might be difficult to apply and implement HE in wearable devices. There are several researchers which tried to apply HE to healthcare systems. In [1, 2, 3, 4], Kocabas et al. proposed a general architecture for medical cyber-physical systems (MCPSs). The architecture employed two schemes of HE: the Paillier scheme [5] to compute the average heart rate and the BGV scheme [6] from HElib library [7]) for the long QT syndrome detection. Their results showed that they could retrieve the average heart rate on the cloud nearly with real-time response. However, their scheme actually required high computation and communication costs on both the user and the server sides. Our work has been motivated by this problem. In addition, Sun et al. [8] presented similar results to Kocabas et al.'s one, however, the difference between them is that for the underlying FHE scheme, Kocabas et al. used the HElib library while Sun et al. used Dowlin's scheme.

To the best of our knowledge, most previous works only implemented HE on PC, not on resource-restricted devices. Preuveneers and Joosen [9] implemented FHE by using HElib library on wearable devices for analyzing diabetics and sharing data with physicians or other caregivers (e.g. parents of diabetic children). They concluded that they demonstrated the practical feasibility of this solution, but it was not so practical due to the resource limitation.

There are two works [10, 11] which reported the comparison of famous HE schemes. In [10], Lepoint and Naehrig reported comparison results of the BGV and YASHE schemes. After that, Costache and Smart [11] extended the work [10] and showed comparable results among four HE schemes, the FV, YASHE, BGV, and NTRU schemes. They applied the same API and the same optimization to the four schemes and concluded that the BGV scheme appears to be the most efficient for large plaintext moduli, while the YASHE scheme seems the most efficient for small plaintext moduli. Note that the BGV scheme means HElib library, and the YASHE scheme means SEAL library. However, to the best of our knowledge, there is no research on estimating the efficiency of HElib and SEAL libraries on resource-constrained devices in a general setting, by which our work is also motivated.

There are several works that reported energy costs of cryptographic algorithms in resource-constrained devices. They can be categorized into two kinds: measuring the energy consumption by using an event-driven simulation and measuring the energy consumption by devices such as Oscilloscope and Picoscope. In researches on the former [12, 13, 14], they measured the energy consumption of AES, RSA, ECDSA, RC5, and RC6 by PowerTOSSIM. In researches on the latter [15, 16, 17, 18], they used Picoscope or Oscilloscope to measure the energy consumption of AES, ECDSA, RC5, DES, XTEA, SHA2, and Keccak. All previous studies in energy costs of cryptographic algorithms for resource-constrained devices were done for the lightweight cryptography or public-key cryptography. To the best of our knowledge, there is no work on energy costs of FHE in resource-constrained devices. Our work is also motivated by this problem.

Specifically, the contribution of this paper is as follows:

- This paper investigates the communication overhead of a certain privacy-preserving protocol by 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 overhead 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 overhead as the *trivial* protocol, which is the same protocol without considering privacy (i.e., a protocol without FHE).

- This paper evaluates efficiency of two FHE libraries, *HElib* and *SEAL*, on a resource-constrained device. We implement them on PC supposed to be a cloud server, and Raspberry Pi supposed to be a wearable device such as a smartphone or any resource-constrained 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.

- This paper observes energy consumption on Raspberry Pi, which is supposed to be a wearable device or a smartphone in our protocol. We investigate the energy required in transmission, the energy for executing each algorithm and the total amount of energy consumption obtained by them. Our result shows that the energy for executing each algorithm is very small compared to the energy required in transmission. SEAL produces smaller sizes of ciphertexts than HElib and it consumes few energy consumptions, though the massive energy is consumed in transmission due to the large ciphertexts in both HElib and SEAL.

As a result, both HElib and SEAL would be used on restricted resource devices from our analysis. In particular, SEAL would be more suitable for practical use in resource-constrained devices.
The primary version of this paper appeared in [19], and this paper is an extended and full version of it. The main difference between this paper and the primary version [19] is that this paper newly includes an analysis of energy consumption for FHE in Section 6.

## 2 FULLY HOMOMORPHIC ENCRYPTION (FHE)

Homomorphic encryption (HE) is 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 $\tilde{c} := \textbf{\textit{Eval}}_{pk}(\textbf{\textit{Enc}}_{pk}(m_1), \textbf{\textit{Enc}}_{pk}(m_2), \star)$, which satisfies $\textbf{\textit{Dec}}_{sk}(\tilde{c}) = m_1 \star m_2$. HE can be categorized into three types with respect to the number of allowed operations on the ciphertexts 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 [5] allows only addition and was used in [4] to calculate the average heart rate on the cloud, while the RSA scheme [20] allows only multiplication.

- 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. In the BGN scheme [21], 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 [22] 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 mainly the following FHE schemes proposed after Gentry's work: Ideal lattice-based FHE [22], FHE over integers [23, 24], FHE from the learning with errors (LWE) assumption [25, 26], and NTRU-like FHE [27, 28]. FHE from LWE assumption is an important step to a practical FHE, which first made by Brakerski and Vaikuntanathan, and they take the advantage of efficiency feature of Ring-Learning with Errors (RLWE) [25, 26]. Both LWE and RLWE problems are the assumed infeasibility problems, and the performance of the scheme based on RLWE is better than that of the scheme based on LWE. In particular, the most famous FHE libraries with RLWE assumption are HElib and SEAL, and they used the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [6] and the Brakerski/Fan-Vercauteren scheme (BFV) [29], respectively. Whenever a homomorphic operation is applied, some noise will be added into the ciphertext and then 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 function to reduce the noise and get a fresh ciphertext from the noisy ciphertext corresponding to the same plaintext. The fresh ciphertext can continue to allow us to apply a homomorphic operation until the noise is not over the limitation.

In this paper, we focus on two well-known FHE libraries, HElib and SEAL, which we summarize in Table 1.
Firstly, *HElib* [7] is an open source library implemented by Halevi and Shoup in 2014, which is based on Brakerski-Gentry-Vaikuntanathan (BGV) scheme [6]. Halevi and Shoup also applied several techniques: a ciphertext packing proposed by Smart and Vercauteren [30], an optimization for homomorphic evaluation proposed by Gentry, Halevi, and Smart [31], and noise management by bootstrapping [32]. This library is written in C++ and has several parameters which effect to the performance and security level, thus it seems to be difficult to select the suitable parameters for non-experts. Moreover, it requires two prerequisite libraries: GNU Multiple Precision Arithmetic (GMP) library [33] and NTL mathematical library [34] (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 [35].
Secondly, *Simple Encrypted Arithmetic Library (SEAL)* [36] was developed by Cryptography Research Group at Microsoft Research in 2015. It is based on Brakerski/Fan-Vercauteren scheme (BFV) [29], 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 esti-

mator tools instead. It needs the Microsoft Research License Agreement to use and be free for the research purpose.

**Table 1.** The property of FHE libraries.

| **Name** | **HElib [7]** | **SEAL [36]** |
|---|---|---|
| Base Scheme | BGV [6] | BFV [29] |
| Language | C/C++ | C++/.NET |
| Required Libraries | GMP [33], NTL [34] | No |
| Bootstrapping | Yes | No |

## 3 PRIVACY-PRESERVING PROTOCOL FOR WEARABLE DEVICES IN HEALTHCARE SYSTEMS

We assume that there is a user (e.g., a patient), a user's smartphone, a cloud server, and a caregiver (or 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 [37]. 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 communicates 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 $SN_1, \ldots, SN_n$ 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 through 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 $Enc$. 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

[1]We consider the maximum packet size based on IEEE 802.15.6 [37].

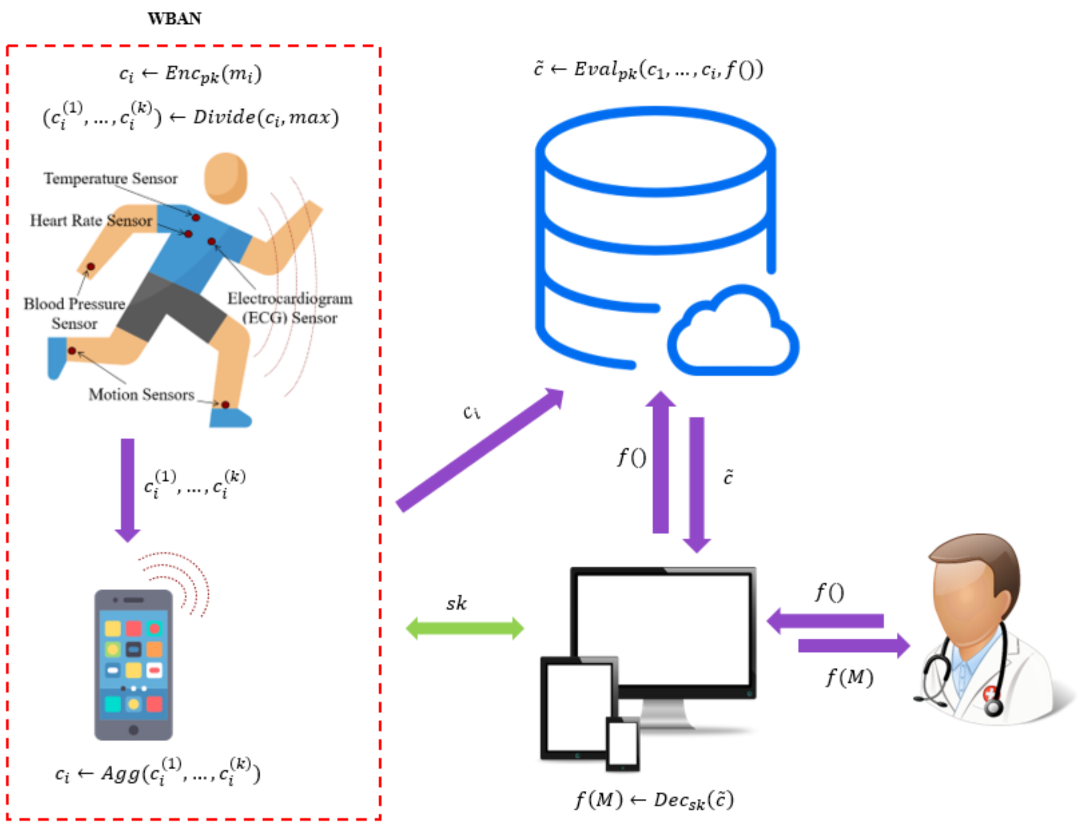
WBAN
$c_i \leftarrow Enc_{pk}(m_i)$
$(c_i^{(1)}, \dots, c_i^{(k)}) \leftarrow Divide(c_i, max)$
Temperature Sensor
Heart Rate Sensor
Blood Pressure Sensor
Electrocardiogram (ECG) Sensor
Motion Sensors
$c_i^{(1)}, \dots, c_i^{(k)}$
$c_i \leftarrow Agg(c_i^{(1)}, \dots, c_i^{(k)})$
$\tilde{c} \leftarrow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

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

close to the limit, the cloud server needs to perform bootstrapping to reduce the noise and get a fresh ciphertext having the same underlying plaintext. After applying the bootstrapping, a fresh ciphertext allows us to continue to perform homomorphic operations if 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 $M$.

## 4 ANALYSIS OF COMMUNICATION OVERHEAD IN WBAN

In [38], 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 [4], and it was also mentioned that the communication overhead was a 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. [39] presented the comparison of several major wireless sensor network (WSN) simulators. The results show that OMNET++ [40] 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 behaviors 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 [41]. Our experiment simulates WBAN through OMNET++ version 4.6 and Castalia version 3.3.

**Table 2.** Simulation parameters.

| Parameters | Value |
|---|---|
| Number of sensor nodes | 6 |
| 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 |

We simulate WBAN through OMNET++ and Castalia with parameters in Table 2. Firstly, we limit the number of sensor nodes to 6 nodes, and assigned Medium Access Control protocol (MAC) is Baseline BAN MAC, which is 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 the 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 limits 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 transmitted in cleartexts. One of the 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 uses 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 must 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-sending 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 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 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 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 |

**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 |

Next, we investigate the number of delayed packets from each sensor node in Table 4. It can be seen that the number of delayed packets from each sensor node of our protocol with HElib is enormously 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 $SN_1$.

Furthermore, we depict a number of times that the packet was successfully transmitted in Fig. 2. 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 packet was successfully transmitted in the first time (1st-try) of NoEncrypt scheme and our protocol with SEAL approximately

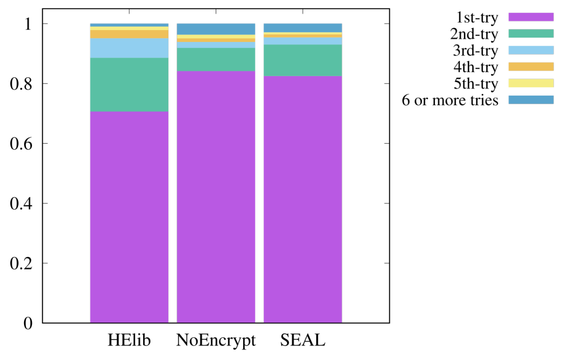

**Figure 2.** A number of times that the packet was successfully transmitted by expression with fractions of 1.

80%, while that of HElib approximately 70%. This result corresponds to the result in Table 4. As a result, our protocol with HElib produces much communication overhead in WBAN. On the other hand, our protocol with SEAL produces few communication overheads nearly close to those of NoEncrypt scheme. Therefore, we can conclude that our protocol with SEAL is better in terms of communication costs.

## 5 ANALYSIS OF EFFICIENCY FOR FHE

In this section, we investigate the 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 the market have much higher performance than this Raspberry Pi, and we can expect that implementation results on a 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 the 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 a fair comparison. Therefore, this experiment uses the plaintext-space GF($2^{10}$) whose size is 1024 in SEAL library (the same as the previous experiment in Section 4.1); and the size of the plaintext-space of HElib must be a prime number (or a power of a prime), whereby we use the size 1021 which is close to 1024.

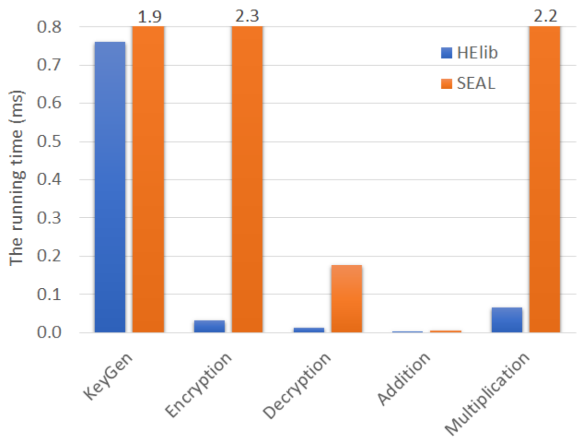

**Figure 3.** The running time on PC (Milliseconds).

### 5.2 Comparison Results

Firstly, we observe the running time (in milliseconds) in each algorithm of HElib and SEAL libraries on the PC and Raspberry Pi, and the results are summarized in Fig. 3 and 4, respectively. The $x$-axis means the algorithms and the $y$-axis means the running time in milliseconds. For readability, Fig. 3 and 4 limit the $y$-axis at 0.8 and 90 milliseconds, respectively, and place the data labels at the

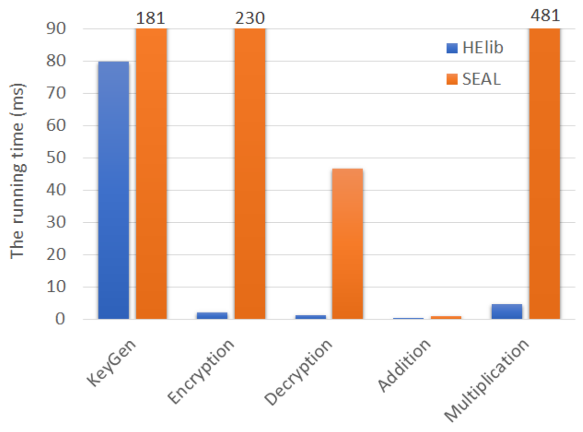

**Figure 4.** The running time on Raspberry Pi (Milliseconds).

top of each algorithm whose running time exceeds the limitation. The exact running time in each algorithm and that of bootstrapping are provided in Appendix A. It can be seen that HElib library is faster than SEAL library on both platforms. 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.

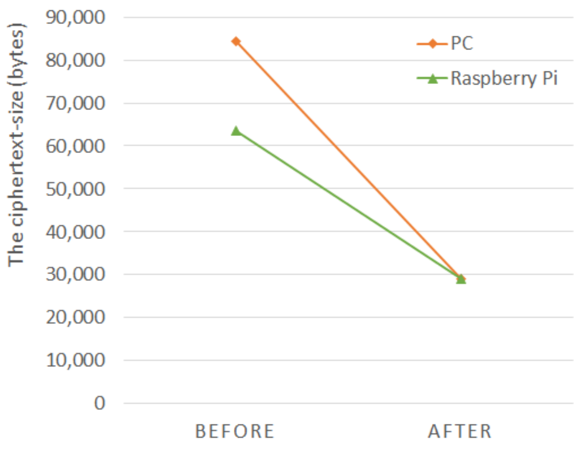

**Figure 5.** The ciphertext-size of HElib library before/after using bootstrapping (bytes).

Moreover, we investigate the ciphertext-size in HElib library before/after using bootstrapping on PC and Raspberry Pi, and the results are summarized in Fig. 5. The $x$-axis means before/after using bootstrapping and the $y$-axis means the ciphertext-size in bytes. 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.

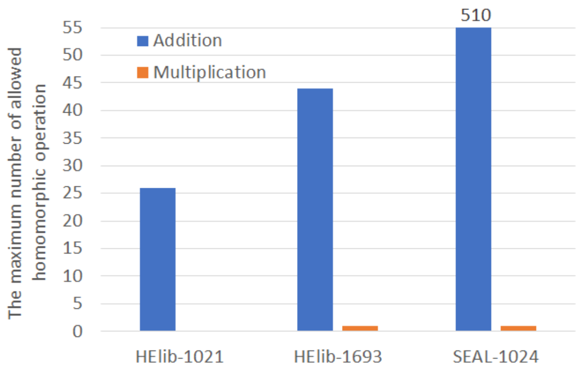

**Figure 6.** The maximum number of allowed homomorphic operations.

Furthermore, we investigate the maximum number up to which homomorphic operations can apply, and the results are summarized in Fig. 6. The $x$-axis means the plaintext-size in each library and the $y$-axis means the maximum number of allowed homomorphic operations. For readability, Fig. 6 limits the $y$-axis at 55 times and place the data labels at the top of the SEAL having plaintext-size 1024 (SEAL-1024) that exceeds the limitation. 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 larger 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. Afterward, 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 larger 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.

## 6 ANALYSIS OF ENERGY CONSUMPTION FOR FHE

This section analyzes energy consumption for two FHE libraries, HElib and SEAL, on Raspberry Pi Model B+ v1.2 with ARM11 at 700 MHz and 512 MB SDRAM.

### 6.1 Experimental Setup

This experiment can be divided into two parts. Firstly, the energy in transmission was investigated by using the energy consumption model in [41] through a network simulator with the same parameter setting in Section 4.1. Secondly, the energy consumption of HElib and SEAL libraries on Raspberry Pi was investigated by the setting in Fig. 7. For estimating the energy consumption on Raspberry Pi, this experiment uses a Teledyne LeCroy HDO6104A high definition Oscilloscope and uses the previous experiment setting in Section 5.1. The three algorithms were investigated in terms of energy consumption on Raspberry Pi, Key generation, Encryption and Decryption algorithms which are executed on WBAN devices in our protocol (see Fig. 1). To evaluate the power consumption of each algorithm in HElib and SEAL libraries, a 10 $\Omega$ resistor was placed between the power supply (5V) and the Raspberry Pi. Two channels of this Oscilloscope were used as follows: Channel 1 denoted by CH1 in Fig. 7 was used for measuring the voltage drop across the resistor, and Channel 2 denoted by CH2 in Fig. 7 was used to visualize the waveform when toggling the GPIO pins. From Ohm's Law, we have the power in watts $P = V^2/R$, where $V$ is the voltage consumed during the execution of each algorithm and $R$ is a resistor in $\Omega$. Then, we have the energy consumption in joules $E = P \cdot T$, where $T$ is the time in seconds. Therefore, this is written as $E = (V^2/10) \cdot T$ in our setting.

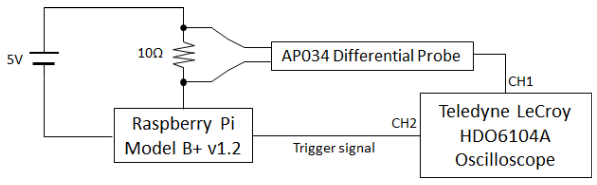
5V
10Ω
AP034 Differential Probe
CH1
Raspberry Pi Model B+ v1.2
CH2
Trigger signal
Teledyne LeCroy HDO6104A Oscilloscope

**Figure 7.** The experimental setup of energy consumption of HElib and SEAL libraries on Raspberry Pi.

### 6.2 Energy Consumption for FHE

Firstly, the energy required in transmission was investigated and shown in Fig. 8. The $x$-axis means NoEncrypt scheme, HElib and SEAL libraries, while the $y$-axis means the energy required in transmission (in millijoules), which was calculated based on the total number of bytes transmitted in Wi-Fi from $SN_1, ..., SN_n$. Note that Fig. 8 limits the $y$-axis at 6.5 millijoules and places the data label at the top of HElib library if it exceeds the limitation. NoEncrypt scheme consumes the lowest energy 0.074 millijoules in transmission, while HElib consumes the highest energy 12,154 millijoules in transmission. SEAL consumes the energy 5.974 millijoules in transmission. Hence, it is clearly observed that the tendency of the energy consumed in transmission and the communication costs in Section 4.2 are very similar, because the energy required in transmission relates to the ciphertext-size: A large ciphertext is divided into many packets (pieces of the ciphertext), and hence it will need high energy for transmitting the whole ciphertext.

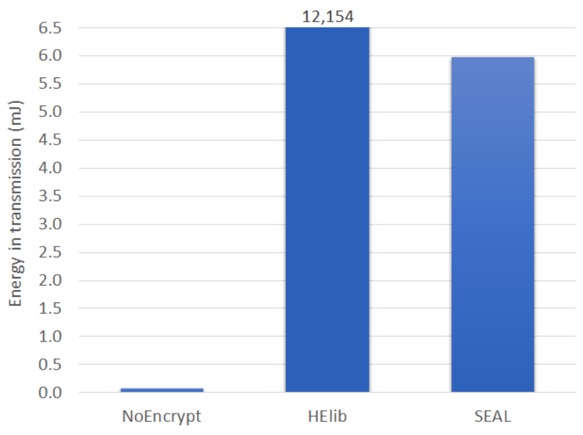

**Figure 8.** The energy in transmission (Millijoules).

Secondly, the energy consumption in each algorithm of HElib and SEAL libraries on Raspberry Pi was investigated and shown in Fig. 9. In Fig. 9, the $x$-axis means the algorithms, and the $y$-axis means the energy consumption (in millijoules) in the algorithms of HElib and SEAL libraries on Raspberry Pi. HElib consumes energy less than SEAL in every algorithm, however, each algorithm of both libraries consumes energy at most 1 millijoules. As mentioned in Section 6.1, the energy consumption was calculated by $E = (V^2/10) \cdot T$, thereby the algorithm which takes a long running time leads to high energy consumption. The tendency of energy consumption in each algorithm is similar to the running time shown in Fig. 4.

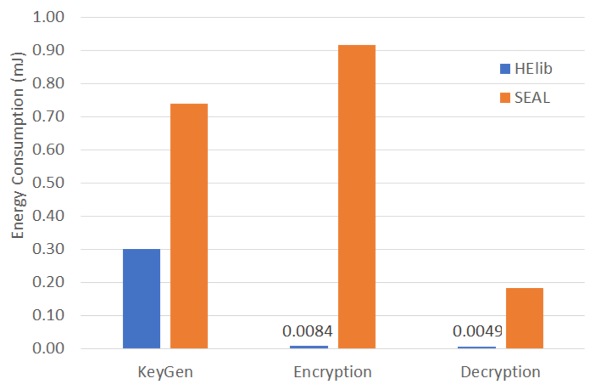

**Figure 9.** The energy consumption in each algorithm of HElib and SEAL libraries on Raspberry Pi (Millijoules).

In our protocol in Section 3, there are sensor nodes $SN_1, ..., SN_n$ and a sink node $SN_0$ which could be wearable devices and a smartphone, respectively. The smartphone (i.e., $SN_0$) will execute Key generation (KeyGen) algorithm, and it will also execute Decryption algorithm when a user or a caregiver retrieves data through the smartphone application. The wearable devices (i.e., $SN_i$ for $1 \leq i \leq n$) will execute Encryption algorithm. Hence, the total amount of energy consumption in $SN_0$ is evaluated as the total amount of the energy in boot situation, the energy in executing Key generation algorithm, the energy required in transmission and the energy in executing Decryption algorithm. On the other hand, the energy consumption in each $SN_i$ $(1 \leq i \leq n)$ is evaluated as the total amount of the energy in boot situation, the energy in executing Encryption algorithm and the energy required in transmission. Note that the FAQ in the documentation of Raspberry Pi [42] reported the average energy in boot situation for Model B+ v1.2 was 1.1 watts. The time for booting Raspberry Pi depends on the SD card read/write speed (the class of SD card). This experiment uses the SD card class 10, thereby the time for booting needs approximately 15 seconds. For the energy in boot situation in joules was calculated by $E = P \cdot T = 1.1 \cdot 15 = 16.5$ joules. Fig. 10 shows the total amount of energy consumption at $SN_0$ and $SN_n$ of HElib and SEAL libraries on Raspberry Pi, where $SN_n$ is selected as a representative of sensor nodes. The $x$-axis means $SN_0$ or $SN_n$ of HElib or SEAL libraries, and the $y$-axis means the total amount of energy consumption (in millijoules). The energy in the boot situation of every node is 16,500 millijoules, thereby the $y$-axis starts from 16,000 millijoules. It can be seen that the energy in transmission in HElib takes much energy consumption, therefore this figure uses the large scale on the $y$-axis and we cannot observe the energy in Encryption and KeyGen algorithms. The total amount of energy consumption in each node of HElib is 28.655 joules, and that of SEAL is 16.507 joules. Currently, a Lithium polymer battery is widely used in wearable devices. The battery has energy approximately 0.90-2.63 megajoules and the typical energy of two AA batteries is 18,720 joules, thus both HElib and SEAL can be used in WBAN devices. In fact, the energy consumed in KeyGen and Decryption algorithms at $SN_0$, and the energy in Encryption algorithm at $SN_n$ are less than 1 millijoules, which implies negligible costs. The highest energy consumption is used for the transmission in both libraries if we do not consider the energy in boot situation, because every node has the same initial values. As described earlier, the energy in transmission depends on the ciphertext-size, thereby HElib consumes the energy much higher than SEAL.

We can observe that the energy for executing each algorithm is very small compared to the

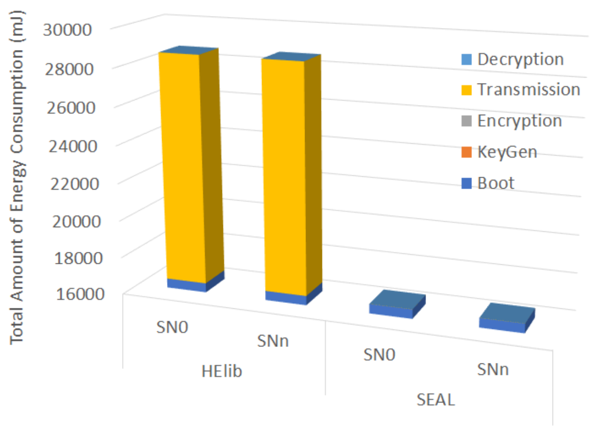

**Figure 10.** The total amount of energy consumption at $SN_0$ and $SN_n$ (Millijoules).

energy in transmission, and also that the energy in transmission depends on the ciphertext-size. SEAL produces smaller sizes of ciphertexts than HElib, therefore it consumes few energy consumptions. The time complexity of SEAL is close to that of HElib, thereby each algorithm of SEAL consumes energy close to HElib. As a result, both HElib and SEAL can be used on restricted resource devices in a general setting, and in particular, SEAL would be more suitable for practical use in resource-constrained devices.

## 7 CONCLUSION

This paper investigated practical feasibility of FHE in resource-constrained devices for healthcare systems. To do so, the privacy-preserving protocol was defined and implemented by using two FHE libraries, *HElib* and *SEAL*, on Raspberry Pi and the network simulator, OMNET++ and Castalia, to measure computational cost, communication overhead and energy consumption in WBAN.
First, we investigated the communication overhead of the privacy-preserving protocol by using the FHE libraries over WBAN. Our result showed that the protocol with SEAL was better than that with HElib.
Secondly, we evaluated efficiency of the FHE libraries on a resource-constrained device. We implemented them on PC supposed to be a cloud server, and Raspberry Pi supposed to be a wearable device such as a smartphone or any resource-constrained device over WBAN. Our result showed that HElib was better in terms of running time than SEAL, while SEAL could perform more homomorphic operations than HElib for the almost same plaintext size.
Thirdly, we investigated energy consumption on Raspberry Pi. Specifically, we investigated the energy required in transmission, the energy for executing each algorithm of the FHE libraries and the total amount of energy consumption obtained by them. Our result showed that the energy for executing each algorithm was very small compared to the energy required in transmission. SEAL produced smaller sizes of ciphertexts than HElib, therefore it consumed few energy consumptions. The massive energy was consumed in transmission due to the large ciphertexts in both HElib and SEAL.
Consequently, we observed that both HElib and SEAL would be used on resource-constrained devices in terms of computational cost, communication overhead, and energy consumption. In particular, SEAL would be more suitable for practical use in resource-constrained devices. As a future work, it would be interesting to implement our protocol in real devices and to observe the performance in a real environment.

## 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 phys-

ical systems," vol. 13, no. 3, May 2016, pp. 401–416.

[5] 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.

[6] 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.

[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] 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.

[9] 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.

[10] 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.

[11] 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.

[12] A. S. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Third IEEE International Conference on Pervasive Computing and Communications*, March 2005, pp. 324–328.

[13] S. B. Othman, A. Trad, and H. Youssef, "Performance evaluation of encryption algorithm for wireless sensor networks," in *2012 International Conference on Information Technology and e-Services*, March 2012, pp. 1–8.

[14] A. Trad, A. A. Bahattab, and S. B. Othman, "Performance trade-offs of encryption algorithms for wireless sensor networks," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, Jan 2014, pp. 1–6.

[15] C. Chang, S. Muftic, and D. J. Nagel, "Measurement of energy costs of security in wireless sensor nodes," in *2007 16th International Conference on Computer Communications and Networks*, Aug 2007, pp. 95–102.

[16] ——, "Security in operational wireless sensor networks," in *2008 5th IEEE Consumer Communications and Networking Conference*, Jan 2008, pp. 781–785.

[17] M. Zhang, M. M. Kermani, A. Raghunathan, and N. K. Jha, "Energy-efficient and secure sensor data transmission using encompression," in *2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems*, Jan 2013, pp. 31–36.

[18] L. P. I. Ledwaba, G. P. Hancke, H. S. Venter, and S. J. Isaac, "Performance costs of software cryptography in securing new-generation internet of energy endpoint devices," *IEEE Access*, vol. 6, pp. 9303–9323, 2018.

[19] A. Prasitsupparote, Y. Watanabe, and J. Shikata, "Implementation and analysis of fully homomorphic encryption in wearable devices," in *The Fourth International Conference on Information Security and Digital Forensics*. The Society of Digital Information and Wireless Communications, 2018, pp. 1–14.

[20] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

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

[31] 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.

[32] 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.

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

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

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

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

[37] "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

[38] 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.

[39] 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.

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

[41] 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

[42] FAQs - raspberry pi documentation. [Online]. Available: https://www.raspberrypi.org/documentation/faqs/

## APPENDIX A

**Table 5.** The running time on PC (ms).

| | **HElib** | **SEAL** |
|---|---:|---:|
| KeyGen | 0.760058 | 1.930100 |
| Encryption | 0.032094 | 2.342723 |
| Decryption | 0.013368 | 0.177101 |
| Addition | 0.000094 | 0.005329 |
| Multiplication | 0.066178 | 2.187750 |
| Bootstrapping | 85.323830 | - |

**Table 6.** The running time on Raspberry Pi (ms).

| | **HElib** | **SEAL** |
|---|---:|---:|
| KeyGen | 79.933075 | 181.319900 |
| Encryption | 2.084733 | 229.979548 |
| Decryption | 1.258043 | 46.673325 |
| Addition | 0.006370 | 0.920642 |
| Multiplication | 4.707492 | 480.622600 |
| Bootstrapping | 7,846.207000 | - |