# Protect Applications and Data in Use in IoT Environment Using Collaborative Computing

Xincai Peng[1], Li Shan Cang[1(✉)], Shuai Zhang[1], and Muddesar Iqbal[2]

[1] University of Electronic Science and Technology of China, Chengdu, China
shancang.li@outlook.com

[2] College of Engineering, Prince Sultan University, Riyadh, Saudi Arabia

**Abstract.** In IoT systems, traditional encryption can be used to protect IoT applications and data at rest or in transit that transforms data in to ciphertext making it unreadable. However, it is very challenging to protect IoT systems against attacks targeting data and applications in use. Using homomorphic encryption, this work proposed a lightweight collaborative computing scheme to protect both applications and data in IoT environment that includes IoT devices, mobile apps, and cloud server. A novel key management system scheme proposed as a trusted third party to collaboratively generate and distribute keys by cloud servers and IoT devices, in which data is only visible to the data owner but keep encrypted to other parties. A SEAL-CKKS scheme and a K-means clustering algorithms were validated, and the experimental results demonstrated the effectiveness of proposed schemes, in which the K-means clustering algorithm in the plaintext state, the proposed scheme still maintains an accuracy up to 84.1%.

**Keywords:** Collaborative computing · IoT · Data Privacy · Data Security

## 1 Introduction

The Internet of Things (IoT) has been one of the fastest developing techniques in recent years and significantly increasing number of IoT devices are interconnected in Internet of Things (IoT) systems. According to [6,11], there will be more than 27 billion IoT devices will be connected to Internet by 2025. This makes it is very challenging to protect sensitive information and privacy in IoT environment due to increasing security concerns [1,7], such as *software vulnerabilities, enlarged attack surface, limited resources, lack of standardisation, data diversity, etc.*

Data collected by IoT devices is usually transmitted to IoT applications or cloud server for analysis. Any vulnerability can lead sensitive data disclosure or system failure, which can affect the IoT users, applications, third-parties and even can cause security issues [4]. In IoT applications, smart sensors, devices, and


H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 562, pp. 265–280, 2024.
https://doi.org/10.1007/978-3-031-54528-3_15

IoT systems can collaboratively collect, process, and share data in an efficient way. In IoT collaborative computing, following main features need to be considered [14]: (1) data sharing and fusion; (2) interoperability; (3) collaborative analytics; (4) security and privacy [17].

The IoT data lifecycle includes *data generation, data transmission, data storage, data in use, data processing, data analytics, etc.* Data in use security is a critical stage in IoT data lifecycle, which focus the security of a specific state of data within the overall data lifecycle [9,16]. Ensuring the security of data in use is crucial for safeguarding IoT systems from different IoT threats and vulnerabilities, key security measurements include [20]: *data encryption, communication security, access control, authentication and authorisation, data anonymisation and masking, real-time threat detection, etc.* [18].

The homomorphic encryption (HE) shows great potential to encrypt the data in use, and IoT applications or cloud sever could then perform computations on encrypted data, such that statistical analysis or machine learning (ML), without decryption the data [2,8,12]. The HE can significantly enhance data privacy in IoT applications where user data is collected and analyzed [5]. For example, in a healthcare IoT system, HE could be used to encrypt patient health data before transmission to a cloud server for analysis [10]. This would ensure that the patient's sensitive health data remains private and secure, even during the analysis process.

Using HE and smart IoT devices, this work developed a lightweight privacy enhancing solution for IoT applications that can enhance privacy protection in IoT systems by allowing data to be encrypted before transmission, while still allowing computations to be performed on the encrypted data. This can help to protect sensitive data and maintain user privacy in IoT applications. The main contribution of this work are summarised as:

1) Aiming at deploying HE over IoT devices, a four phase scheme: *data encoding, encryption, decryption, and decoding* was introduced to enhance the efficiency of HE;
2) A lightweight secure collaborative computing scheme was proposed for IoT devices using the homomorphic encryption to conduct secret computing without concerning data disclosure concerns;
3) A robust key management scheme (KM was proposed for the HE enabled lightweight secure collaborative computing protocol over resource constrained IoT scenarios;
4) To validate the proposed solution in cloud scenarios, this work re-implemented the $k$-means scheme proposed in [21], the experimental results demonstrate affordable performance can be achieved.

## 2 Related Works

The amount of sensitive data being collected and transmitted has increased significantly in IoT systems. This data may involve personal individual information (PII), sensitive information such as *financial data, healthcare recordes,*

*confidential business information, etc.* that make privacy a critical concern for IoT systems. In the past decade, the HE based solutions have been developed to enhance privacy [8]. In our previous works [15], we proposed a privacy-preserving IoT data analysis system that uses HE to protect sensitive data. The proposed system uses the Paillier homomorphic encryption (PHE) algorithm to encrypt the data before transmitting it to the cloud server for analysis.

Shrestha *et al.* proposed a secure computing system for IoT using the Fan-Vercauteren (FV) HE algorithm to enable computations to be performed on encrypted data [19]. The system was evaluated using a synthetic IoT dataset and was found to be effective in protecting the privacy of the data while still allowing computations to be performed on it. Halder *et al.* proposed a secure data storage system for IoT using the Brakerski-Gentry-Vaikuntanathan (BGV) HE algorithm to encrypt the data before storing it on the cloud server [4].

Louki *et al.* provided an overview of the use of HE for IoT data security in [10] that reviewed various HE schemes that can be used for IoT security, as well as the challenges and limitations of using homomorphic encryption in IoT. Overall, HE has been shown to be an effective solution to several IoT security challenges. However, there are still challenges to be addressed, such as the high computational overhead of homomorphic encryption and the need for standardized homomorphic encryption algorithms for IoT [6].

Some other approaches have also been developed to implement HE over IoT and cloud scenarios to protect the confidentiality of data stored on IoT devices. For example, HE can be used to encrypt sensor data before it is stored on the device, thereby preventing unauthorized access to the data. HE can also be used to perform analytics on encrypted data, enabling privacy-preserving machine learning algorithms to be used in IoT systems.

Marcolla *et al.* detailed the implementation of HE on FPGA for IoT applications [13], in which the benefits of implementing HE on FPGA were highlighted, such as the ability to perform high-speed computations with low latency and power consumption. This work also discussed the challenges of implementing HE on FPGA, such as the need for efficient hardware design and optimization techniques.

While the HE has the potential to enhance IoT privacy, there are several challenges that need to be addressed, including the high computational overhead of HE, key management scheme, and the lack of standardization. In the following works, we focus on developing more efficient HE algorithms, improving key management for HE in IoT systems.

## 3 Methodology

This work focuses on an 'IoT Device' - 'Cloud Server' scenario to achieve outsourced secure computing of individual IoT device data. In this scenario, both IoT device and cloud server do not support multi-party collaborative computing between multiple devices, so data between multiple devices of the same type cannot be merged for more efficient utilization. Therefore, we first propose an improved model with a key management system (KMS) to address the issues

of the above model. KMS is used to generate keys for collaborative computing, publish the public key to IoT devices in need and the cloud server, and send the private key to the data owner to protect data while ensuring that the cloud server can merge and calculate data from different devices.

In recent, researchers proposed the "multi-key" model to solve the problem of data collaborative computing [3,11]. One drawback of this "multi-key" scheme is that the upper limit of the number of participants must be known when generating keys, as the parameters increase with the increase of the number of participants. However, the number of IoT devices is often dynamic and growing rapidly in the real IoT environment, using a "multi-key" solution is not suitable for IoT environments.

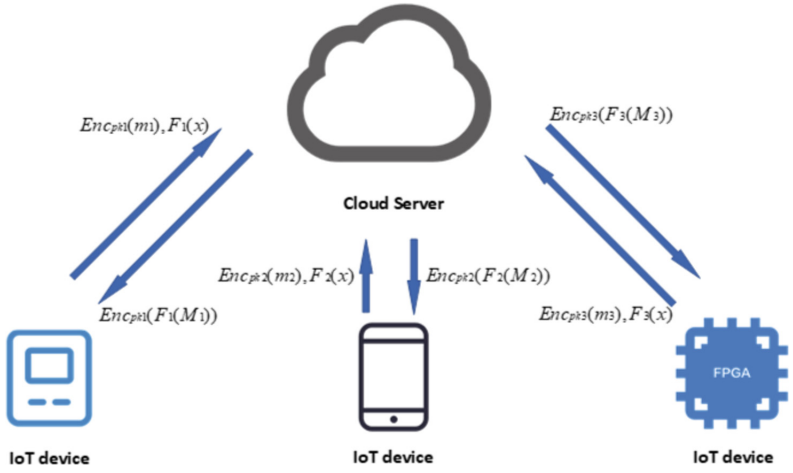


**Fig. 1.** Lightweight HE in IoT scenario

### 3.1 Key Management Scheme

To address above concern, we re-designed the KMS to form a trusted IoT systems, in which IoT devices are in a mutually trusted system, they can directly use the same key pair inside the system to encrypt data with homomorphic encryption and send those encrypted data to the Cloud Sever(CS). IoT devices in the same system trust each other and can share plaintext data. Therefore, data within the system does not need to be shared through encryption.

In multiple independent IoT systems, as shown in the Fig. 1, data owners (DOs) directly interact with the CS, transmit encrypted data through communication channels, and store it on the cloud server. The DOs can read and use this batch of data encrypted by their own. If there is no prior agreement, each system will use different homomorphic encryption keys, which hinders the collaborative computing of data in different systems on the CS.

To enhance the trustworthiness, a reliable KMS can be introduced as a trusted third party for generating and publishing keys. With the help of KMS,

the IoT devices in different systems could obtain the same key pairs of HE. This means that IoT devices can use the same public key to encrypt their data and send it to CS for data sharing, without exposing the content of the data. On the other hand, the CS is responsible for responding to user requests, integrating data with the same features but different ones, which can be seen as a horizontal federated learning model, and providing computing services, without sharing the ciphertext data of DOs with other users. Figure 2 shows the structure of privacy protection scheme with KMS.

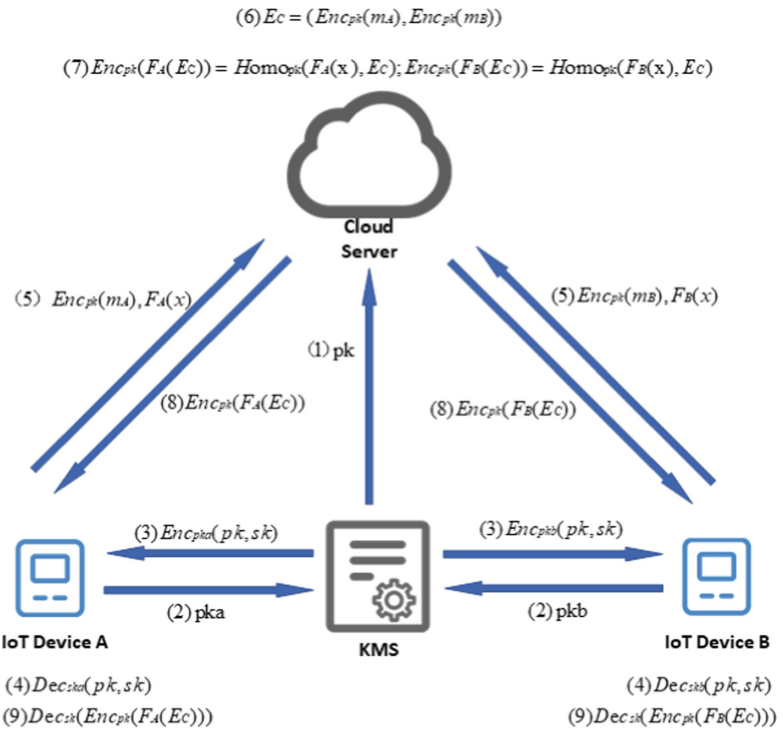


**Fig. 2.** The HE-based collaboration computing system

**System Initialisation and Keys Generation.** In Fig. 2, the KMS can help CS to collect ciphertext from different IoT systems and integrate them for collaborative computing over devices do not trust each other (e.g., IoT device $A$ and $B$), which includes following nine steps:

Step 1 KMS generates a pair of homomorphic encryption keys, including public key $pk$ and private key $sk$, $(pk, sk)$, and then sends the collaborative homomorphic public key $pk$ to CS;

Step 2 Device $A$ generates a pair of keys $(pka, ska)$ for communication and sends the public key $pka$ to KMS; Device $B$ generates a pair of keys $(pkb, skb)$ for communication and sends the public key $pkb$ to KMS;

Step 3 KMS encrypts homomorphic keys $(pk, sk)$ used for collaborative operations using $pka$ and $pkb$ and returns the encryption result $Enc_{pka(pk,sk)}$ to $A$, $Enc_{pkb(pk,sk)}$ to $B$;
Step 4 To obtain the collaborative homomorphic public key $pk$, device $A$ decrypts $Enc_{pka}(pk)$ through $ska$, device $B$ decrypts $Enc_{pkb}(pk)$ through $skb$;
Step 5 Using the collaborative homomorphic public key $pk$, device $A$ encrypts its own data $mA$, device $B$ encrypts its own $mB$, and then $A$ send the ciphertext $Enc_{pk}(mA)$ to CS with $F_A(x)$, $B$ send the ciphertext $Enc_{pk}(mB)$ to CS with $F_B(x)$, while $F_A(x)$ and $F_B(x)$ represent collaborative computing requests sent by devices A and B to the server, which can be an arithmetic function or a complex model, such as the Kmeans algorithm;
Step 6 CS consolidates $Enc_{pk}(mA)$ and $Enc_{pk}(mB)$ to form a larger dataset $E_c$;
Step 7 CS perform collaborative homomorphic operations on $E_c$ according to different requests from $A$ and $B$ to obtain the results $Enc_{pk}(F_A(E_C))$ and $Enc_{pk}(F_B(E_C))$
Step 8 CS send $Enc_{pk}(F_A(E_C))$ to $A$ and send $Enc_{pk}(F_B(E_C))$ to $B$.
Step 9 A decrypts the ciphertext $Enc_{pk}(F_A(E_C))$ with the private key $sk$, and obtains the operation result $F_A(E_C)$. B decrypts the ciphertext $Enc_{pk}(F_B(E_C))$ with $sk$, and obtains the operation result $F_B(E_C)$.

**Key Distribution.** The proposed key distribution scheme includes three stages: *Initialization, aggregation, and collaboration*:

Stage 1 *Initialization*: KMS **generates** a collaborative key pair $(pk, sk)$ ($pk$ is the collaborative public key, $sk$ is the private key), **synchronizes** the key pair to the requesting IoT devices, and **shares** the $pk$ with the cloud server (CS) which serves the corresponding IoT devices;
Stage 2 *Aggregation*: IoT device $I$ use collaborative public key $pk$ to **encrypt** its own data, and **send** the ciphertext and its requests $F_I(x)$ to the cloud, who integrates the data encrypted with the same key.
Stage 3 *Collaboration*: CS firstly consolidates encrypted data from different IoT devices, and then calculates the integrated data $E_C$ based on functions $F_I(x)$ and $pk$, and returns the results to the corresponding IoT device.

### 3.2 Implementation of Collaborative Computing Services

As mentioned above, the collaborative computing can be delivered over ciphertext as a service. In this sub section, we will introduce a collaborative $k$-means clustering algorithm over ciphertext as a use case of collaborative computing application.

**Use case: Collaborative $k$-Means Clustering Algorithm.** The data in IoT needs to be well protected using collaborative computing services, which could be an algorithm, a function, or a complex task in IoT scenarios. In this work, we will implement the $k$-means algorithms as an example to demonstrate the proposed collaborative computing scheme. The $k$-means algorithm is widely used in many IoT applications, such as *healthcare data analysis, finace analysis, etc.*, in which key information is often used for analysis but can only be shared in a trusted way.

In implementing collaborative $k$-means algorithm, following four components were introduced: *$K$-means clustering under ciphertext, homomorphic squared Euclidean distance calculation, homomorphic comparison, and homomorphic minization*, to achieve clustering analysis of multi-party data in the constructed privacy protection architecture.

**$K$-Means Algorithm.** As shown in Algorithm 1, in the $k$-means clustering algorithm the data $X = \{\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_{N-1}}\}$ has $M$ attributes, namely $0 \leq i \leq N-1$, the number of clusters is $k$, and the maximum number of iterations $\epsilon$. The output is clustering result $C^{(\epsilon)}$.

During the initialization process, $t = 0$ and randomly select $k$ sample points as the cluster center as $\mathbf{m^{(0)}} = (\mathbf{m_1^{(0)}}, \ldots, \mathbf{m_k^{(0)}})$.

At *$t$-th* iteration of the clustering process, for $\mathbf{x}_i \notin \mathbf{m}^{(t)}$, the square Euclidean distance from $\mathbf{x_i}$ to cluster center $\mathbf{m_l^{(t)}}$, $d_{il}^2$, can be achieved by

$$d_{il}^2 = \sum_{j=0}^{M-1} (x_i[j] - m_l[j])^2 \tag{1}$$

where $l = 1, ..., k$, and merge them into the cluster where the nearest sample center is located. Then we can obtain the clustering results of this round of iteration $t$.

For clustering result $C^t$ obtained at *$t$-th* iteration, we can iteratively calculate the mean of all samples in each cluster to obtain a new sample center point $\mathbf{m^{(t)}}$. Until the termination condition is reached when the number of iterations is greater than or equal to $\epsilon$.

**Homomorphic Squared Euclidean Distance.** The general definition of Euclidean distance for any two given sample points $\mathbf{x_i}, \mathbf{y_i}$ is

$$d_{ij}^2(x_i, x_j) = \sum_{j=0}^{M-1} \sqrt{(x_i[j] - y_i[j])^2} \tag{2}$$

where $M$ is the dimension of vector $\mathbf{x}$.

In the operation of homomorphic scheme, in order to avoid the calculation of roots, we use the form of squared Euclidean distance, and define the loss function as

**Algorithm 1.** K-means algorithm

**Input**: $X = (\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_{N-1}})$, $k$, $\epsilon$
**Output**: Clustering of $k$ sample sets $C^{(\epsilon)}$

1: **Initialization.** Let $t = 0$, randomly select $K$ sample points as the initial clustering center $\mathbf{m^{(0)}} = (\mathbf{m_1^{(0)}}, \ldots, \mathbf{m_k^{(0)}})$.
2: **Clustering.** For a fixed sample center $\mathbf{m^{(t)}}$, in which $m_l^{(t)}$ is the center of class $G_l$, calculate the square Euclidean distance $d_{it}$ from each sample $(\mathbf{x_i}) \notin \mathbf{m^{(t)}}$ to the center of the class, assign each sample to the class closest to it, and form the clustering result $C^{(t)} = (C_1^{(t)}, \ldots, C_k^{(t)})$.
3: **Update Class Centers.** For clustering result $C^{(t)}$, calculate the mean $(\overline{\mathbf{x_l}}, \overline{\mathbf{y_l}})$ of all samples in different clusters $C_l^{(t)}$ as the new class center $\mathbf{m^{(t+1)}} = (\mathbf{m_1^{(t+1)}})$.
4: **if** $t \geq \epsilon$ **then**
5: $\quad C^* = C^{(t)}$
6: **else**
7: $\quad t = t + 1$ goto step 2.
8: **end if**

$$L(C) = \sum_{l=1}^{k} \sum_{i=0}^{M-1} ||x_i - m_l||^2 \tag{3}$$

Then, the square Euclidean distance can be obtained by

$$d_{ij}^2(x_i, x_j) = \sum_{j=0}^{M-1} (x_i[j] -_h m_l[j]) \times_h (x_i[j] -_h m_l[j]) \tag{4}$$

**Homomorphic Comparison.** In clustering algorithms, the main goal is to minimize the loss function $L$. It is an critical process to compare the distance between sample point and the sample center in each iteration, this can be can be implemented via the *homomorphic comparison algorithm*, as shown in Algorithm 2, in which the input parameters include a plaintext space modulus $p$, and two distances $d_{C_i}, d_{C_j}$, obtained by Eq. (4).

The goal is to obtain ciphertext with lower plaintext values. For scheme's security, we add a random number $b \in \{0, 1\}$ to avoid the IoT devices learning the order of ciphertexts. According to the value of $b$, the CS can perform corresponding homomorphic operation. $Cq$ is the operated result of CS, and be sent to the IoT devices. The IoT devices can decrypt the $Cq$ to get plaintext $q$.

The $\lfloor log_2 q \rfloor$-*th* bit of $q$ determines the return value $\omega$ from IoT devices to cloud server. Then, the Cloud Server judges the size of $d_{C_i}$, $d_{C_j}$ based on the values of $b$ and $\omega$. The important steps of the Algorithm 2 are explained as follows:

**In Stage 1-CS:**

**Input** $d_{C_i}, d_{C_j}$,**and** $C_p$. $p$ is plaintext modulus of homomorphic parameters and $C_p$ is the ciphertext of $p$ under the same $pk$ with $d_{C_i}, d_{C_j}$.

**Line 1: Generate a random number** $b$. It can be implemented using a simple random number generator.

**Line 2 to 5: Compute to get** $C_q$. According to $b$, the algorithm will choose Line 3 or 5 as the next step.

**In Stage 2-IoT devices:**

**Line 1: Decrypt** $C_q$ **to obtain** $q$. $C_q$ is the ciphertext of $p$ under the same $pk$ with $d_{C_i}, d_{C_j}$ and can be decrypt with $sk$ owned by IoT devices.

**Line 2 to 5: Get** $\omega$**, the** $\lfloor log_2 q \rfloor$***-th* bit of** $q$. IoT devices do not know how CS compares $d_{C_i}$ and $d_{C_j}$ without knowing the value of the random number $b$ selected by CS, so they cannot determine the meaning of $\omega$. Meanwhile, CS does not have $sk$, it needs this $\omega$ to confirm the next step of operation.

**In Stage 3-CS:**

**Line 1: Make a Judgement.** According to the value of $b$ and $\omega$, CS could determine the return value. For example, if $b = 0$, it means that $C_q = C_p +_h d_{C_i} -_h d_{C_j}$, i.e. finding the difference between $d_{C_i}$ and $d_{C_j}$ in plaintext. At this time, if $\omega = 0$, it means that the difference between $d_{C_i}$ and $d_{C_j}$ in plaintext is less than 0, which means $d_{C_i} < d_{C_j}$, so we return the smaller one, $d_{C_i}$.

**Algorithm 2.** Homomorphic Comparison Algorithm

**Function: Comp**$(\mathbf{d_{C_i}}, \mathbf{d_{C_j}})$
**Input**:$d_{C_i}, d_{C_j}$, and $C_p$
**Output**:The comparison result.

**Stage 1-CS:**

1: Choose a random bit $b \in \{0, 1\}$
2: **if** b=0 **then**
3: Compute $C_q = C_p +_h d_{C_i} -_h d_{C_j}$
4: **else**
5: Compute $C_q = C_p +_h d_{C_j} -_h d_{C_i}$
6: send $Cq$ to IoT devices;
7: **end if**

**Stage 2-IoT devices:**

1: Decrypt $C_q$ to obtain $q$ with secret key $sk$;
2: **if** the $\lfloor log_2 q \rfloor$-$th$ bit of $q$ is 0 **then**
3: send $\omega = 0$ to cloud server
4: **else**
5: send $\omega = 1$ to cloud server
6: **end if**

**Stage 3-CS:**

1: **if** $b = 0, \omega = 0$ *or* $b = 1, \omega = 1$ **then**
2: **return** $d_{C_i}$
3: **else**
4: **return** $d_{C_j}$
5: **end if**

**Homomorphic Minimisation.** For every sample in each iteration, there will be $k$ distances, $d_{C_i}, i = 1, \ldots, k$. Initially we set $d_{C_{min}} = INF$ to as the minimum

value, where $INF$ is a ciphertext with plaintext value that larger than all possible plaintext value of $d_{C_i}$. By iterating through all the elements in $d_{C_i}$, we can obtain the minimum distance $d_{C_{min}}$ using Algorithm 3.

**Algorithm 3.** Homomorphic Minimization Algorithm

**Function: Min**$(\mathbf{d_{C_1}}, ..., \mathbf{d_{C_k}})$
**Input**: $d_{C_i}, i = 1, ..., k$, and $d_{C_{min}} = INF$.
**Output**:The minimum distance $d_{C_{min}}$. **CS:**
1: **for** $i = 1$ to $k$ **do**
2: $\quad d_{C_{min}} = Comp(d_{C_{min}}, d_{C_i})$
3: **end for**
4: **return** $d_{C_{min}}$

## 4 Validation

To validate the proposed solution, we introduced a practical cloud scenario with I*oT device, IoT app, edge device, and cloud server* to conduct tests. In order to eliminate interference, the experiment is only used to verify the functionality and efficiency of each part, ignoring the time consumed by data transmission in the communication channel.

A Raspberry *pi* is used as typical IoT devices, an Android mobile *app* was developed on *IQOO Z*5 as typical IoT applications, and the Altera Cyclone *DE*0 platform as an *Edgedevices* with ARM + FPGA architecture. We have built a local server for low latency communication on a desktop (Intel i7 2.5GHz, 32GB). A cloud server (CS) was set up at the *Alibaba Cloud* platform to meet more realistic scenario requirements.

We use physiological information about heart disease patients dataset (301 records each with 11 features) for an unsupervised clustering analysis, which can category similar treatment methods to patients[1].

**Evaluation Criteria.** In this work, we introduce *time efficiency, space efficiency, accuracy*, and *energy efficiency* to evaluate the proposed solution. The key generation and distribution speed are used to evaluate the efficiency of KMS systems. The efficiency and accuracy of the $K$-means algorithm based on homomorphic encryption were used to evaluate the proposed scheme in executing the function $F(x)$ on the CS. The pace of *encoding, decoding, encryption, and decryption* is used to evaluate the efficiency of IoT devices in the entire environment. The spatial and energy efficiency of IoT devices have also been included in the evaluation criteria.

[1] https://www.kaggle.com/datasets/kingabzpro/heart-disease-patients.

**Time Efficiency.** Regarding the time efficiency of our proposed scheme, (1) the IoT devices first submit an encryption request to KMS; (2) After receiving the request, KMS generates key pairs for collaborative computing and encrypts the collaborative key pair $(pk, sk)$ using the IoT device's public key; (3) Then, the encryption result is packaged and sent to the IoT device.

We compared the performance with the widely used CKKS scheme [11]. There are two main parameters in CKKS scheme: *polynomial modulus degree* and *coefficient modulus*: (1) the polynomial modulus degree is a power of 2 whose value will affect the calculation speed. A large polynomial modulus degree will cause a low calculation pace, while it can support more complex encryption operations; (2) The length of the coefficient modulus is the sum of its prime factor lengths.

In order to conduct control experiments, in the following analysis, we will uniformly set the polynomial modulus degree value of 8192, and set the coefficient modulus value of 200. The time consumption for initializing homomorphic encryption all related variables with the above parameters is 0.131 s.

**Table 1.** Time consumption of proposed scheme for *encoding & encrypting* batch data vs the SEAL-CKKS scheme(ms).

| | encoding | encryption | decryption | decoding |
|---|---|---|---|---|
| SEAL-CKKS | 2.407 | 3.381 | 0.194 | 2.504 |
| Proposed scheme | 0.1125 | 3.1488 | 0.1347 | 1.8486 |

Table 1 shows the performance of proposed scheme compared with *SEAL-CKKS* scheme, it can be seen that the average times of SEAL-CKKS scheme for *encoding, encryption, decryption, and decoding* are 2.407, 3.381, 0.1940, and 2.500 respectively, while the consumed time in proposed scheme were 0.1125, 3.1488, 0.1347, 1.8486, respectively. The performance of encoding in the proposed scheme can significantly reduce the time than that of SEAL-CKKS based scheme and the performance of *encryption, decryption*, and *decoding* are slightly less than in SEAL-CKKS scheme.

Table 2 shows the performance of proposed scheme over IoT devices. The average times for *encoding, encryption, decryption, decoding* on IoT devices are 0.3749, 10.4948, 0.4491, 6.1614 respectively.

Figure 3 presents the time consumed by different IoT devices are very similar, and the time consumed for subtraction and addition operation is pretty low and the time for square operation is about 200 us. Actually, the operation relinearize costs the main time, average 1.19 ms. Figure 4 shows the execution efficiency of proposed scheme on a dataset of heart disease patients. We conducted 40, 60, and 80 rounds of iterations for clusters value of 2, 4, and 6, respectively. It can be seen in Fig. 4 that under the same number of iterations, the algorithm's time varies with the number of clusters.

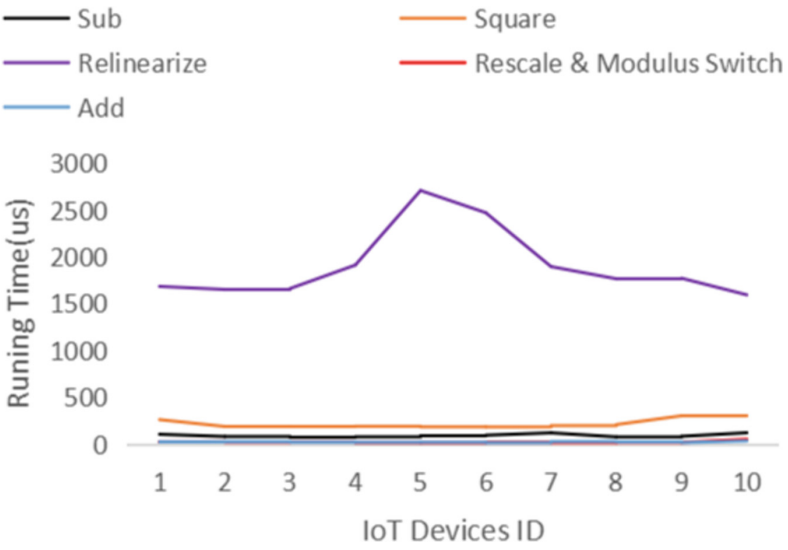


**Fig. 3.** Time consumption of steps of get Euclid distances

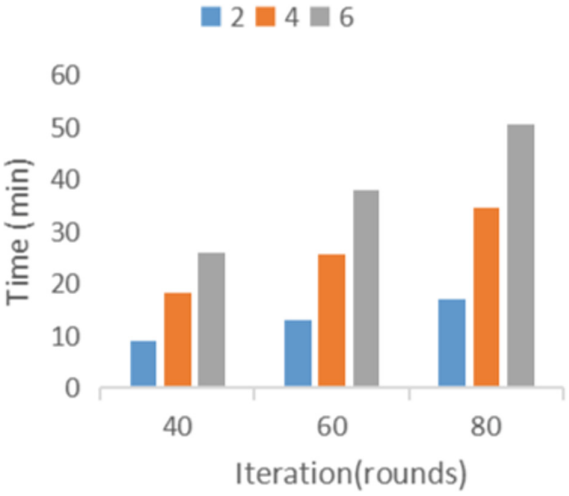


**Fig. 4.** The efficiency of our collaboration k-means clustering scheme.

**Table 2.** Encoding, encryption, decryption, and decoding of IoT devices(ms).

| Device ID | Encoding | Encryption | Decryption | Decoding |
|---|---|---|---|---|
| 1 | 0.3721 | 10.4739 | 0.4699 | 6.2976 |
| 2 | 0.3686 | 10.2436 | 0.4716 | 6.1405 |
| 3 | 0.3872 | 10.5427 | 0.4369 | 6.1829 |
| 4 | 0.3670 | 10.5438 | 0.4477 | 6.3591 |
| 5 | 0.3753 | 10.4750 | 0.4518 | 6.0873 |
| 6 | 0.3757 | 10.5477 | 0.4316 | 6.0702 |
| 7 | 0.3831 | 10.5776 | 0.4571 | 6.3392 |
| 8 | 0.3585 | 10.4122 | 0.4429 | 6.0313 |
| 9 | 0.3912 | 10.6525 | 0.4405 | 6.1123 |
| 10 | 0.3699 | 10.4791 | 0.4409 | 5.9939 |
| Average | 0.3749 | 10.4948 | 0.4491 | 6.1614 |

**Space Efficiency.** We tested the resident memory size consumed when encrypting data of different sizes on IoT devices. When the polynomial modulus is 8192 and the coefficient modulus is 200, the double precision floating point numbers with the size of 100, 1000, 2000, 3000 are encrypted, and the memory consumed is shown in the Fig. 6. Adopting the single instruction multiple data (SIMD) mode for batch processing of data can greatly save memory space and is friendly to the IoT environment.

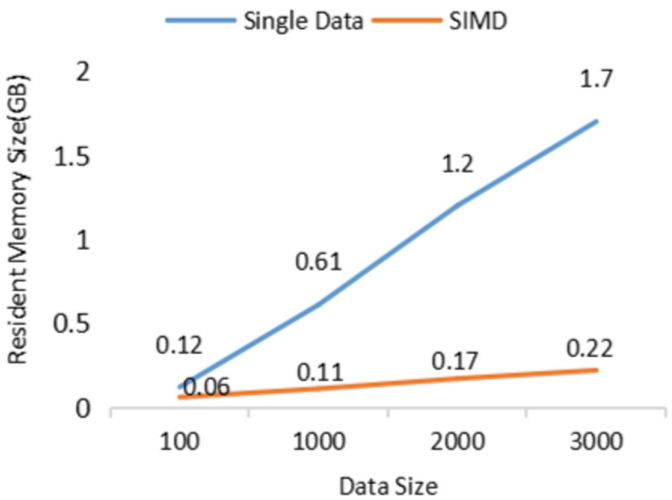


**Fig. 5.** Resident Memory Size by encrypting different sizes of Data.

**Accuracy.** Here we will compare and analyze the accuracy of the proposed k-means clustering algorithm for ciphertext and plaintext. The number of experiments increased from 40 to 80, and the accuracy of ciphertext increased from 58.2% to 84.1%. As the number of experiments increased from 40 to 80, the performance of ciphertext gradually approached that of plaintext (Fig. 5).

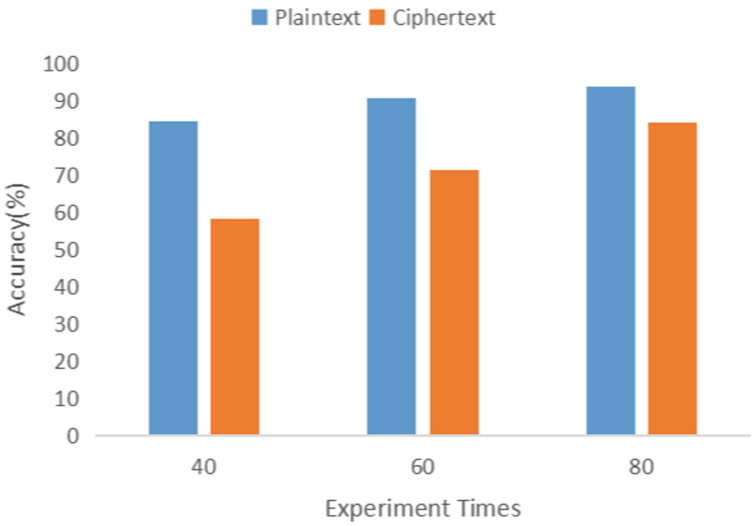


**Fig. 6.** The accuracy of k-means clustering on ciphertext compared with on plaintext.

**Energy Efficiency.** We mainly measured the power consumption of *pi* and DE10-nano in standby mode and continuously operating in proposed solution. In standby mode, the electricity consumption of the DE10-nano is $0.1216\,kW \cdot h$ and the *pi* is $0.065\,kW \cdot h$ within 24 h. Under continuous working conditions, the electrical energy consumption of DE10-nano is $0.144\,kW \cdot h$ and that of *pi* is $0.099\,kW \cdot h$ within 24 h.

## 5 Conclusion

Using the HE, a collaborative computing scheme was proposed to enhance the security and privacy protection for IoT applications and data in use. Specifically, a $k$-mean clustering algorithm were implemented to demonstrate how to secure IoT applications in use. The experimental results demonstrated that the developed scheme can provide affordable secure protection for both IoT application and data in use.

## References


1. Asare, B.T., Quist-Aphetsi, K., Nana, L., Simpson, G.: A nodal authentication iot data model for heterogeneous connected sensor nodes within a blockchain network. In: 2021 International Conference on Cyber Security and Internet of Things (ICSIoT), pp. 65–71 (2021). https://doi.org/10.1109/ICSIoT55070.2021.00021
2. Chaudhary, P., Gupta, R., Singh, A., Majumder, P.: Analysis and comparison of various fully homomorphic encryption techniques. In: 2019 International Conference on Computing, Power and Communication Technologies (GUCON), pp. 58–62 (2019)
3. Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from tfhe. In: Advances in Cryptology-ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part II 25, pp. 446–472. Springer (2019)
4. Halder, S., Newe, T.: Enabling secure time-series data sharing via homomorphic encryption in cloud-assisted iiot. Futur. Gener. Comput. Syst. **133**, 351–363 (2022)
5. Joshua, W.X.K., Justin, X.W.T., Yap, C.N.: Arithmetic circuit homomorphic encryption key pairing comparisons and analysis between elliptic curve Diffie Hellman and supersingular isogeny Diffie Hellman. In: 2021 2nd Asia Conference on Computers and Communications (ACCC), pp. 138–142 (2021). https://doi.org/10.1109/ACCC54619.2021.00030
6. Li, S.: Zero trust based internet of things. EAI Endorsed Trans. Internet Things **5**(20), e1–e1 (2019)
7. Li, S., Zhao, S., Min, G., Qi, L., Liu, G.: Lightweight privacy-preserving scheme using homomorphic encryption in industrial internet of things. IEEE Internet Things J. **9**(16), 14542–14550 (2022). https://doi.org/10.1109/JIOT.2021.3066427
8. Li, X., et al.: Design and verification of the arm confidential compute architecture. In: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), pp. 465–484 (2022)
9. Liu, X.K., Wang, S.Q., Chi, M., Liu, Z.W., Wang, Y.W.: Resilient secondary control and stability analysis for dc microgrids under mixed cyber attacks. IEEE Trans. Industr. Electron. **71**(2), 1938–1947 (2024). https://doi.org/10.1109/TIE.2023.3262893
10. Loukil, F., Ghedira-Guegan, C., Boukadi, K., Benharkat, A.N.: Privacy-preserving iot data aggregation based on blockchain and homomorphic encryption. Sensors **21**(7), 2452 (2021)
11. Ma, J., Naas, S.A., Sigg, S., Lyu, X.: Privacy-preserving federated learning based on multi-key homomorphic encryption. Int. J. Intell. Syst. **37**(9), 5880–5901 (2022)
12. Mahmood, Z.H., Ibrahem, M.K.: New fully homomorphic encryption scheme based on multistage partial homomorphic encryption applied in cloud computing. In: 2018 1st Annual International Conference on Information and Sciences (AiCIS), pp. 182–186 (2018). https://doi.org/10.1109/AiCIS.2018.00043
13. Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F.H., Aaraj, N.: Survey on fully homomorphic encryption, theory, and applications. Proc. IEEE **110**(10), 1572–1609 (2022)
14. Ni, C., Cang, L.S., Gope, P., Min, G.: Data anonymization evaluation for big data and iot environment. Inf. Sci. **605**, 381–392 (2022)
15. Ren, W., et al.: Privacy-preserving using homomorphic encryption in mobile iot systems. Comput. Commun. **165**, 105–111 (2021)

16. Sandomirskii, M., et al.: Femtosecond direct laser writing on bi-layer gold-silicon films for hidden data storage and random key generation. In: 2023 IEEE 23rd International Conference on Nanotechnology (NANO), pp. 1090–1094 (2023). https://doi.org/10.1109/NANO58406.2023.10231269
17. Sharbaf, M.S.: Iot driving new business model, and iot security, privacy, and awareness challenges. In: 2022 IEEE 8th World Forum on Internet of Things (WF-IoT), pp. 1–4 (2022). https://doi.org/10.1109/WF-IoT54382.2022.10152044
18. Shoji, Y., Nakauchi, K., Liu, W., Watanabe, Y., Maruyama, K., Okamoto, K.: A community-based iot service platform to locally disseminate socially-valuable data : Best effort local data sharing network with no conscious effort? In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), pp. 724–728 (2019). https://doi.org/10.1109/WF-IoT.2019.8767237
19. Shrestha, R., Kim, S.: Integration of iot with blockchain and homomorphic encryption: challenging issues and opportunities. In: Advances in Computers, vol. 115, pp. 293–331. Elsevier (2019)
20. Wang, G., Huang, X., Zhang, X., Liu, G., Zuo, F.: Design and analysis of secure localization against vulnerability-induced attack for internet of things. In: 2023 IEEE/CIC International Conference on Communications in China (ICCC Workshops), pp. 1–6 (2023). https://doi.org/10.1109/ICCCWorkshops57813.2023.10233751
21. Zhang, P., et al.: Privacy-preserving and outsourced multi-party k-means clustering based on multi-key fully homomorphic encryption. IEEE Trans. Dependable Secure Comput. (2022)