Safer and lighter: New Privacy-Preserving joint Computations for Optimal Location Selection

Shuai Shang, Xiong Li, Rongxing Lu, *Fellow, IEEE*, Jianwei Niu, Xiaosong Zhang, Mohsen Guizani, *Fellow, IEEE*

***Abstract*—Edge-supported industrial Internet of Things (IIoT) has recently received significant attention since the edge computing can greatly improve the service quality of IIoT applications. However, edge servers are not fully trusted and are often deployed at the edge of the network. Therefore, there are some security issues. For edge-supported IIoT, privacy-preserving range query is one of the most important functional requirements. Recently, some privacy-preserving range query solutions have been proposed in different fields. However, most of them only support single-dimensional range query, which are inefficient for the requirement of multi-dimensional range query. To address these problems, we propose a privacy-preserving multi-dimensional range query scheme for edge-supported IIoT, called Edge-PPMRQ, in this paper. In Edge-PPMRQ, a novel range division algorithm is designed, through which the multi-dimensional ranges can be merged into one range, so as to achieve multi-dimensional range query through one query request. In addition, Edge-PPMRQ also supports the range queries for continuous, discontinuous and arbitrary boundary ranges. The detailed security analysis proves that Edge-PPMRQ is privacy-preserving for the query ranges, the query result and the sensed data of IIoT devices. Furthermore, extensive comparison experiments also illustrate that Edge-PPMRQ is efficient in communication and computation.**

***Index Terms*—Range query, multi-dimensional, privacy-preserving, industrial Internet of Things (IIoT), edge computing.**

I. Introduction

With the significant advances of information technology, Internet of Things (IoT) [**?** **?**] has been widely applied in different fields, e.g., smart grids, intelligent parking and smart homes, which makes our daily lives much more convenient. Especially, with the applications of IoT in industrial fields, the so-called Industrial Internet of Things (IIoT) [**?**] has emerged, which can greatly improve work efficiency and reduce resource consumption. As an emerging computing technology, edge computing [**?**] can store and process data near the IIoT devices, so it can not only greatly reduce the communication load and computational cost of IIoT devices to extend their life cycle, but also achieve almost real-time data processing. Therefore, edge computing is very suitable for supporting IIoT applications. In edge-supported IIoT, a large number of sensors are deployed in industrial environment to collect different types of real-time data, and the edge server is capable to receive and process the sensed data of IIoT sensors locally. The results of data processing can help the industrial devices make precise decision, which can evidently improve the work efficiency and reliability of industrial devices.

For edge-supported IIoT applications, range query is one of the most significant functional requirements. For example, in a factory, different types of industrial sensors are deployed to collect real-time environment data, such as water volume, power consumption, pressure and temperature. Based on the number of sensors whose sensed data exceeds the threshold values, the manager of the factory can determine whether water volume and temperature value are in normal ranges or not. In IIoT environment, various types of data are generated. To query different types of data, a single-dimensional range query scheme is inefficient since it needs to send multiple query requests, while a multi-dimensional range query scheme can achieve the same purpose by only one request. Therefore, in such scenario, a multi-dimensional range query scheme can play a more important role than a single-dimensional solution. At the same time, in range query, privacy preservation [**?**] is a critical problem. Some adversaries may try to get the query range and the sensed data of the factory, and further infer the secret information. For example, if the sensor data generated in the production environment is obtained by a competitor, e.g., temperature, humidity, and material consumption, the competitor can easily refer important information about the production technology directly or indirectly, which is likely to bring huge economic losses to the factory. In other words, privacy-preserving multi-dimensioanl range query scheme is considerably meaningful for edge-supported IIoT.

A. Related Work

In recent years, many privacy-preserving range query solutions have been proposed in different fields, e.g., wireless sensor networks (WSN) [**?** **?** **?** **?**], cloud computing [**?** **?** **?** **?**] and IoT [**?** **?** **?** **?** **?** **?**]. In WSN, Yi et al. [**?**] proposed an efficient scheme to process range queries in two-tier sensor networks, which supports the functions of privacy and integrity preservation. However, it cannot support discontinuous range query. Tsou et al. [**?**] proposed an

S. Shang and X. Li are with the Institute for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: shshang180@gmail.com, lixiong@uestc.edu.cn).

R. Lu is with the Faculty of Computer Science, University of New Brunswick, Fredericton, E3B 5A3, Canada (e-mail: RLU1@unb.ca)

J. Niu is with the State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science and Engineering, Beihang University, Beijing 100191, China (e-mail: niujianwei@buaa.edu.cn).

X. Zhang is with the Institute for Cyber Security, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China and with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, Guangdong 518040, China (e-mail: johnsonzxs@uestc.edu.cn).

Mohsen Guizani is with College of Engineering, Qatar University, Qatar (E-mail: mguizani@ieee.org).

efficient and secure anonymous range query scheme for two-tier WSN. The scheme can not only prevent privacy leakage, but also detect the storage nodes which are compromised by attackers. Moreover, the privileges of querists can be verified without revealing their identities. However, the scheme cannot achieve multi-dimensional range query. Zeng et al. [**?**] proposed an energy-efficient range query scheme in 2017, which supports multi-dimensional range query, while the sum function of the sensed data in query ranges cannot be obtained by the query user. Liu et al. [**?**] designed a spatial range aggregation query scheme for dynamic sensor networks supporting privacy-preserving. In cloud computing, Li et al. [**?**] proposed a range query protocol in cloud based on PBtree data structure. It not only achieves strong privacy-preserving, but also supports real-time queries. Xu et al. [**?**] designed a lightweight range query scheme supporting both single- and multi-dimensional range queries. Besides, it can protect the data privacy and integrity of the query results. Li et al. [**?**] proposed a secure scheme of multi-dimensional range query. It not only achieves sub-linear search efficiency, but also is secure in known-background model. Liang et al. [**?**] designed a multi-source scheme with order-preserving encryption for eHealth systems, which supports range queries for different patients. In IoT applications, Li et al. [**?**] proposed a multi-attribute aggregation query mechanism in the context of edge computing, where an energy-aware IR-tree is constructed to process query requests in a single edge network, and a routing graph of edge nodes is established to facilitate query processing for marginal smart devices contained in contiguous edge networks. Djellabi et al. [**?**] proposed a scheme for efficient range queries in IoT. It adopts a data distribution model based on both consistent and order-preserving hash to efficiently handle the range queries. Wan et al. [**?**] proposed a multi-dimensional data indexing scheme, which is energy- and time-efficient. Mahdikhani et al. [**?**] presented an efficient and privacy preservation single-dimensional range query scheme, which achieves $(n+|E|)\cdot \log n$-bit communication efficiency. However, a query user must launch multiple query requests to realize the query for multi-dimentional sensed data, which is relatively inefficient. In the same year, Mahdikhani et al. [**?**] proposed a single-dimensional privacy-preserving range query scheme in fog-based IoT, which achieves $O(\log^3 n)$ communication efficiency. But the lower and upper boundaries of the query ranges in their scheme must be the powers of 2, which is inconvenient for arbitrary boundary query ranges. In 2021, Mahdikhani et al. [**?**] presented a privacy-preserving range query scheme using reduced paths, which employs a symmetric homomorphic encryption (SHE) to encrypt the reduced paths and achieves $O(\log^2 n)$ communication efficiency. Although the scheme is computationally efficient in fog node side, it is at the cost of a large amount of computational overhead for IoT devices, which is not suitable for IIoT applications.

Although several range query solutions have been proposed for different scenarios, most of them just support the single-dimensional range query, i.e., the query user can only query a single kind of sensed data by one query request. In the industrial field, different kinds of data is usually needed to make intelligent decision. For example, in industrial environment, a manager needs different kinds of data collected from a variety of sensors, such as temperature, operation speed and power consumption, to determine if the devices are running normally. Consequently, in such scenarios, the query user has to launch many query requests to achieve the multi-dimensional range query, i.e., the query user needs to launch m query requests to get m-dimensional sensed data. It not only wastes time and bandwidth, but also results in the problem of response delay. However, all the flaws mentioned above can be solved by a multi-dimensional range query scheme. Therefore, in the paper, we focus on the design of an efficient privacy-preserving multi-dimensional range query scheme for edge-supported IIoT.

B. Our Contributions

To achieve the multi-dimensional range query, we propose a range division algorithm to process multi-dimensional query ranges. Based on the algorithm, we propose an efficient privacy-preserving multi-dimensional range query scheme for edge-supported IIoT, called Edge-PPMRQ. Concretely, the main contributions are as below:

1) A range division algorithm is designed to divide multi-dimensional query ranges into the corresponding sub-ranges. Then the sub-ranges are mapped into a group of bloom filters, through which multi-dimensional query ranges are integrated into one query request instead of multiple query requests.
2) Based on the range division algorithm, the bloom filter [**?**], and the OU cryptosystem [**?**], Edge-PPMRQ not only realizes the privacy-preserving multi-dimensional range query by one request, but also supports continuous, discontinuous and arbitrary boundary range queries for data of any dimension.
3) The security analysis shows that Edge-PPMRQ guarantees the privacy of the query ranges, the query results and the sensed data of IIoT devices.
4) Extensive experiments are performed to evaluate and compare the performance of Edge-PPMRQ and related schemes, and the results demonstrate that Edge-PPMRQ is efficient in communication and computation.

C. Organization of the paper

The rest of this paper are arranged as follows. In section II, the preliminary knowledge is introduced. Section III describes our system model, security model and design goals. The proposed privacy-preserving multi-dimensional range query scheme for edge-supported IIoT and the corresponding security analysis are described in section IV and V, respectively. Section VI evaluates the communication and computation performance of Edge-PPMRQ by comparing with other related schemes. Finally, section VIII draws a conclusion.

II. Preliminaries

This section briefly introduces two preliminaries used in Edge-PPMRQ, i.e., the bloom filter [**?**] and the OU cryptosystem [**?**].

A. Bloom Filter

The bloom filter (BF) [**?**] is a kind of data structure composed of a n-bit binary vector and k independent hash functions. It can be used to check whether an element is in a set or not. To achieve this mission, all the bits of the vector are set to 0 initially. For a set of integers $I = \{I_1, I_2, ..., I_s\}$, k hash functions $H_1, H_2, ..., H_k$ are called to compute $H_i(I_j) \in [1, n]$, where $i \in [1, k]$ and $j \in [1, s]$. Then all the $H_i(I_j)$-th bits in the vector are set to 1. Until now, set I is mapped into the BF. To check if an element e is in set I, we compute the values $H_i(e)$, where $i \in [1, k]$. If all the $H_i(e)$-th bits in the BF are 1, it can be confirmed that e is in the set I. Otherwise, e is not in the set. Since only hash operations are used in BF, it is very easy to be implemented on hardware at a high speed. Compared with other methods solving the same problem, BF costs much less storage space, inserting time and query time. However, due to the fact that BF is a probability-based data structure, there exists false positive rate in BF. Fortunately, based on the relationship among the false positive rate P_f, the vector's length n, the number of inserted elements s and the number of hash functions k: $P_f = (1 - (1 - \frac{1}{n})^{ks})^k$, P_f can be constrained within an acceptable range by adjusting n, s and k. In order to understand BF better, an example is given here. Suppose that a set $I = \{11, 12, ..., 20\}$ is mapped into a 1000-bit length bloom filter by using 3 hash functions H_1, H_2 and H_3, whose output ranges are all [1, 1000]. Firstly, the hashed values $H_1(11), H_2(11), H_3(11), H_1(12), H_2(12), H_3(12), ..., H_1(20)$, $H_2(20)$, $H_3(20)$ are computed. Then, the $H_1(11)$-th, $H_2(11)$-th, $H_3(11)$-th, ..., $H_3(20)$-th bits in the vector are set to 1. When we check if 5 and 20 are the elements of I, $H_1(5), H_2(5), H_3(5), H_1(20), H_2(20), H_3(20)$ are computed to check the values of the corresponding positions in the vector. Obviously, all the $H_1(20), H_2(20), H_3(20)$-th bits are 1, while some or all of the $H_1(5), H_2(5), H_3(5)$-th bits are 0. So, it can be confirmed that 20 is in I while 5 is not. For more detailed information about BF, please refer to [**?**].

B. OU cryptosystem

The OU cryptosystem [**?**] is widely used in privacy preservation applications because of its homomorphic properties. It consists of three algorithms, i.e., key generation, encryption and decryption. We describe the OU cryptosystem as below:

1) *KeyGeneration*: Given a security parameter κ, two large prime numbers p and q with the same length κ are chosen. A function $L(x)$ is defined as $L(x) = (x - 1)/p$. Then, $n = p^2q$ is calculated and $g \in Z_n^*$ is chosen, which satisfies that the order of $g^{p-1} \bmod p^2$ is p. Additionally, h is computed as $h = g^n \bmod n$. Finally, the public key and private key of the system are $pk = (n, g, h, \kappa)$ and $sk = (p, q)$, respectively.
2) *Encryption*: Given a plaintext m, $0 \leq m \leq 2^{\kappa-1}$, a random number $r \in Z_n$ is selected. Then, m can be encrypted to the ciphertext $C = E(m) = g^m h^r \bmod n$.
3) *Decryption*: For the ciphertext C, $C_p = C^{p-1} \bmod p^2$ and $g_p = g^{p-1} \bmod p^2$ are computed, and then the plaintext can be recovered as $m = L(C_p)/L(g_p) \bmod p$.

OU cryptosystem is a homomorphic encryption algorithm supporting additive homomorphism and scalar multiplication. Given two plaintext-ciphertext pairs (m_1, C_1) and (m_2, C_2), where $C_1 = E(m_1)$ and $C_2 = E(m_2)$, $C_1 \cdot C_2 = E(m_1) \cdot E(m_2) = E(m_1 + m_2)$ and $C_1^{m_2} = E(m_1 \cdot m_2)$, according to the homomorphic properties.

III. Models and Design Goals

In this section, we describe the system model, the security model and design goals of Edge-PPMRQ.

A. System Model

There are three types of entities involved in Edge-PPMRQ, i.e., a group of IIoT devices $D = \{D_1, D_2, ..., D_N\}$, an edge server and a query user, as shown in Fig. 1.

1) IIoT devices $D = \{D_1, D_2, ..., D_N\}$: A group of N IIoT sensors. They are deployed in industrial environment to collect m-dimensional data, i.e., m types of data, such as water temperature, pressure, power consumption, etc. For each sensor, it first senses the data of the corresponding dimension in the environment, and transfers the sensed data to an edge server subsequently. In reality, the sensed data is not always integers, e.g., 4.957. To handle range query for such data, 4.957 can be transformed into an integer 4957 by multiplying 1000. Therefore, all the sensed data can be processed as integers. Without loss of generality, we assume that in edge-PPMRQ, the sensed data d_k of the IIoT device D_k is an integer within the range of $[1, n]$.
2) Edge server: An edge server bridges IIoT devices and a query user. It usually has more powerful capabilities of storage, communication and computation than that IIoT devices have, which can relieve the computational cost of IIoT devices and respond to the query user at an almost real-time manner. Besides, compared to cloud servers, edge servers have lower cost and can be easily deployed on a large scale. Moreover, the edge servers are capable to process the sensed data in IIoT. Therefore, the edge server is utilized in the model. During the range query, it receives the query request from a query user, and processes the sensed data from IIoT devices accordingly to generate the ciphertexts of query results $C = \{C_1.C_2, ..., C_m\}$. Finally, C are returned to the query user.
3) Query user: A query user can directly send a multi-dimensional range query request to the edge server. For example, a query user requests a query for the number of ith-dimensional IIoT devices whose sensed data is within the range $[L_i, U_i]$, where $1 \leq i \leq m$ and $1 \leq L_i \leq U_i \leq n$. After receiving the ciphertexts of the query result $|D_i'|$ from the edge server, where $|D_i'| = \text{Count}(D_i')$, $i \in [1, m]$ and $D_i' = \{D_k | TID_k = \lambda_i, d_k \in [L_i, U_i]\}$, the query user can recover the query result $|D_i'|$ by decrypting the ciphertexts.

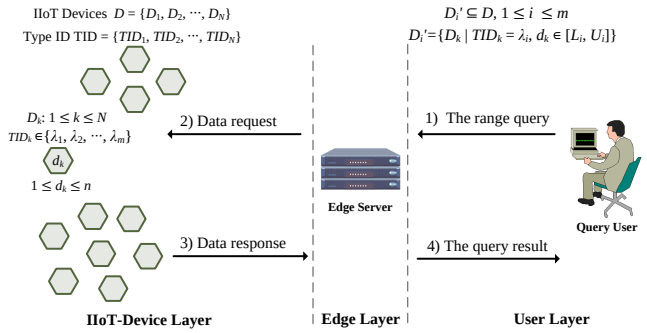


Fig. 1: System model of Edge-PPMRQ

B. Security model

In Edge-PPMRQ, we assume that all entities are honest-but-curious, i.e., each entity performs the protocol honestly, but it also wants to reveal the privacy of other entities. For example, IIoT devices and the edge server are curious about the query ranges $[L_i, U_i]$ and the corresponding query results $|D_i'|$, where $1 \leq i \leq m$; the edge server and the query user are also curious about the sensed data d_k of IIoT device D_k. This paper focuses on the problem of the privacy-preserving multi-dimensional range query. Similar to the preview work, we assume that there is no collusion among the entities. Meanwhile, we don't consider active attacks of external adversaries, which will be discussed in future work.

C. Design goals

Based on the aforementioned system and security models, a privacy-preserving multi-dimensional range query scheme for edge-supported IIoT should achieve the following goals:

1) *Multi-dimensional range query*: Most of traditional range query schemes only support the single-dimensional range query through a query request, while the multi-dimensional range query scheme can realize the query for multi-dimensional data by only one query request.
2) *Privacy preservation*: The query ranges $[L_i, U_i]$ and the query results $|D_i'|$, $i \in [1, m]$, cannot be revealed by any other entities except the query user. Besides, the edge server and the query user can get neither IIoT devices' plaintext of sensed data nor the fact that if the sensed data of IIoT devices is within the query ranges.
3) *Continuous, discontinuous and arbitrary-boundary range query*: Most of the traditional range query schemes only support continuous range query, while in reality, the query ranges may be discontinuous and have arbitrary boundary. Consequently, it is very necessary for the multi-dimensional range query schemes to support all the three types of range queries.
4) *Efficient communication and computation*: Compared to single-dimensional range query schemes, a multi-dimensional range query scheme should achieve the multi-dimensional range query more efficiently in terms of communication and computation. Especially in edge-supported IIoT, the communication overhead and computational costs of IIoT side should be reduced as much as possible.

IV. The Proposed Scheme: Edge-PPMRQ

In this section, it introduces a privacy-preserving multi-dimensional range query scheme for edge-supported IIoT (Edge-PPMRQ). Before the detailed description of the scheme, our designed range division algorithm is first presented, which is a crucial technology in Edge-PPMRQ. Besides, the notations used in Edge-PPMRQ are listed in Table I.

TABLE I: Notations

Notation	Description
N	The number of IIoT devices
n	The maximum value of IIoT devices' sensed data
d	The data in the example, $d \in [1, n]$
D_k, d_k	The k-th IIoT device and its sensed data, $1 \leq k \leq N$
m	The number of query dimensions
t	The number of query sub-ranges
λ_i	The index of i-th dimension, $i \in [1, m]$
λ	The set of λ_i
TID_i	The dimension identifier of D_i's sensed data
TID	The set of TID_i
$[L_i, U_i]$	The query range for i-th dimension data
$[l_j, u_j]$	The j-th query sub-range
Q_1, Q_2	The set of all boundaries in query ranges.
D_i'	The set of IIoT devices, whose sensed data is in $[L_i, U_i]$
$\|D_i'\|$	The number of elements in set D_i'
D'	The set of $\|D_i'\|$, $i \in [1, m]$
BF_j	The j-th bloom filter, $j \in [1, t]$
BF	The set of bloom filters
P_f	The false positive rate of BF
$E_{ij}(r)$	The cipher of λ_i's tab corresponding to BF_j, $r \in \{0, 1\}$
EM	The set of ciphertext $E_{ij}(r)$
C_{ij}	The λ_i's counter corresponding to BF_j
C	The set of counter C_{ij}
κ	The security parameter to establish OU cryptosystem
pk, sk	The public key and the private key of OU cryptosystem
s	The number of elements mapped into bloom filters
s_j	The length of sub-range $[l_j, u_j]$
p	A large prime number
g	A primitive root of Z_p^*
a, g^a	The query user's private and public parameters
b, g^b	The IIoT devices' private and public parameters
g^{ab}	The key shared among query user and IIoT devices
h	A public hash function
h_k	The keyed hash value of d_k, i.e., $h(g^{ab} \| d_k)$

A. Range division algorithm

In order to achieve the multi-dimensional range query, multi-dimensional query ranges should be processed by the range division algorithm as the following steps, which is also shown in Algorithm 1.

1) *Query boundary extraction:* Suppose that $[L_1, U_1]$, $[L_2, U_2]$, ..., and $[L_m, U_m]$ represent the query ranges of m dimensions $\lambda_1, \lambda_2, ...,$ and λ_m, respectively. For $\forall i \in [1, m]$, $1 \leq L_i \leq U_i \leq n$. Then, all the lower boundaries and upper boundaries are extracted to a set $Q_1 = \{L_1, U_1, L_2, U_2, ..., L_m, U_m\}$. After that, both the minimum value 1 and maximum value n are inserted into Q_1. Finally, $Q_1 = \{1, L_1, U_1, L_2, U_2, ..., L_m, U_m, n\}$.
2) *Query boundary sort*: In order to make the range division easier, set Q_1 is sorted from smallest to largest and the repeated boundaries in Q_1 are deleted. Finally, a sorted set $Q_2 = \{l_1, l_2, ..., l_t, l_{t+1}\}$ is generated from Q_1, where $l_1 = 1, l_{t+1} = n, l_j < l_{j+1}$ and $j \in [1, t]$.
3) *Query sub-ranges generation:* According to Q_2, t sub-ranges are generated, i.e., $[l_1, l_2]$, $[l_2, l_3]$, ..., and $[l_t, l_{t+1}]$.

For clarity, we denote the sub-ranges as $[l_1, u_1]$, $[l_2, u_2]$, ..., and $[l_t, u_t]$. Furthermore, the boundaries should be adjusted by adding 1, subtracting 1 or remaining unchanged to get the final sub-ranges $[l'_1, u'_1]$, $[l'_2, u'_2]$, ..., and $[l'_t, u'_t]$, and the following two conditions should be satisfied for all query ranges $[L_i, U_i]$, $1 \leq i \leq m$:

a) $\exists a, b, 1 \leq a \leq b \leq t, [l'_a, u'_a] \cup [l'_{a+1}, u'_{a+1}] \cup ... \cup [l'_b, u'_b] = [L_i, U_i]$, where $l'_a = L_i$ and $u'_b = U_i$.

b) $\forall v, w \in [a, b]$, $v \neq w$, $[l'_v, u'_v] \cap [l'_w, u'_w] = \emptyset$.

Algorithm 1 Range division algorithm.

Input: Query ranges $[L_1, U_1], [L_2, U_2], ..., [L_m, U_m]$ of m dimensions $\lambda_1, \lambda_2, ..., \lambda_m$ and maximum value of range n.
Output: the set of t query sub-ranges $[l_1, u_1]$, $[l_2, u_2]$, ..., $[l_t, u_t]$, i.e., Q_2.

Initialize Q_l, Q_u, Q_1, Q_2 to empty arrays;
Insert 1 into Q_l;
Insert n into Q_u;
Insert $1, n$ into Q_1;
for each $i \in [1, m]$ **do**
    insert L_i into Q_l;
    insert U_i into Q_u;
    insert L_i, U_i into Q_1;
end for
// $Q_l = [1, L_1, L_2, ..., L_m]$;
// $Q_u = [n, U_1, U_2, ..., U_m]$;
// $Q_1 = [1, n, L_1, U_1, L_2, U_2, ..., L_m, U_m]$;
De-duplication (Q_1);
Sort (Q_1) from smallest to biggest;
// $Q_1 = [l_1, l_2, ..., l_t, l_{t+1}]$, where $l_1 = 1$ and $l_{t+1} = n$;
for each $j \in [1, t]$ **do**
    extract l_j, l_{j+1} from Q_1;
    generate range $[l_j, l_{j+1}]$;
    represent range $[l_j, l_{j+1}]$ as $[l_j, u_j]$;
    insert $[l_j, u_j]$ into Q_2;
end for
for each $j \in [1, t]$ **do**
    extract $[l_j, u_j], [l_{j+1}, u_{j+1}]$ from Q_2;
    while $[l_j, u_j] \cap [l_{j+1}, u_{j+1}] \neq \emptyset$ **do**
        if $u_j \in Q_l$ **then**
            $u_j = u_j - 1$
        end if
        if $u_j \in Q_u$ **then**
            $l_{j+1} = l_{j+1} + 1$
        end if
    end while
    insert $[l_j, u_j], [l_{j+1}, u_{j+1}]$ into Q_2;
end for
return Q_2.

To better understand the proposed range division algorithm, we give a toy example here, which is also shown in Fig. 2. For $n = 100, m = 2$ and two dimensions λ_1 and λ_2 with the corresponding query ranges $[20, 60]$ and $[40, 80]$, respectively, the query sub-ranges can be generated by the above range division algorithm:

1) According to ranges $[20, 60]$ and $[40, 80]$, it gets set $Q_1 = \{1, 20, 60, 40, 80, 100\}$.
2) Based on Q_1, it gets the sorted set $Q_2 = \{1, 20, 40, 60, 80, 100\}$.
3) Then it gets 5 sub-ranges $[1, 20], [20, 40], [40, 60], [60, 80]$ and $[80, 100]$. Moreover, the boundaries of the sub-ranges are adjusted to generate the final sub-ranges $[1, 19], [20, 39], [40, 60], [61, 80]$ and $[81, 100]$. From the sub-ranges, we can see that:
    a) $[20, 39] \cup [40, 60] = [20, 60]$ and $[40, 60] \cup [61, 80] = [40, 80]$.
    b) $[20, 39] \cap [40, 60] = \emptyset$ and $[40, 60] \cap [61, 80] = \emptyset$.

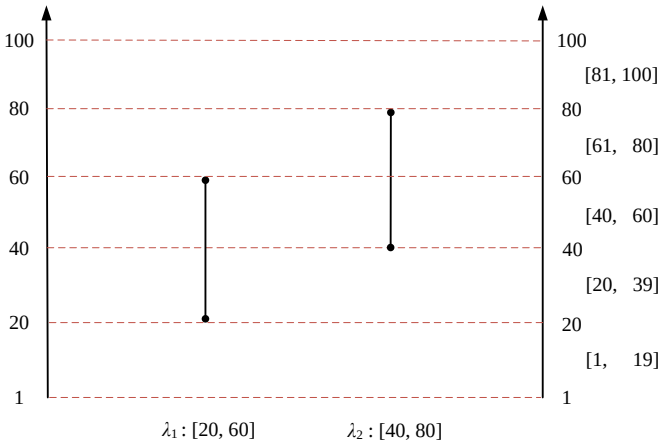


Fig. 2: An example of range division algorithm

It is worth noting that different types of data may locate in ranges with great distance, which exist very small or even no overlap. However, different kinds of data, such as the temperature and pressure, have different accuracy requirements. When the sensed data is transformed into integers based on the aforementioned method, the corresponding ranges may have some overlap. For example, the query ranges of temperature values and pressure values are $[120.5, 200.5]$ and $[1000, 1800]$, respectively. Then, the ranges will be transformed into $[1205, 2005]$ and $[1000, 1800]$ and they have a big overlap $[1205, 1800]$, which is helpful for our scheme. Therefore, in this situation, our scheme can also achieve efficient communication performance.

B. Description of Edge-PPMRQ

Our privacy-preserving multi-dimensional range query scheme for edge-supported IIoT (Edge-PPMRQ) is described in detail as below.

1) System initialization: To initialize the system, the service provider selects a hash function h, a large prime number p, a primitive root $g \in Z_p^*$ and a random number $b \in Z_p^*$. Then, $g^b \bmod p$ can be computed accordingly. Finally, the service provider publishes the parameters h, p, g and g^b. After that, the service provider deploys an edge server and various types of IIoT devices to the target industrial environment.

2) User query request generation: The query user launches a query for data of m dimensions $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$, i.e., "How many IIoT devices of dimension λ_i, whose sensed data d_k is within range $[L_i, U_i]$, where $1 \leq i \leq m$ and $1 \leq k \leq N$?" In other words, the query user would like to figure out:

For $\forall i \in [1,m]$ and $k \in [1,N]$, $|D_i'| = Count(D_i')$, where $D_i' = \{D_k | TID_k = \lambda_i, d_k \in [L_i, U_i]\}$. To generate the query request in a privacy-preserving way, the query user performs the following steps.

Step 1: Given a security parameter κ, the public and private key pair (pk, sk) of OU cryptosystem is generated by the key generation algorithm. Besides, a random number $a \in Z_p^*$ is chosen to compute $g^a \bmod p$ and $g^{ab} = (g^b)^a \bmod p$.

Step 2: For m query ranges $[L_1, U_1], [L_2, U_2], ...,$ and $[L_m, U_m]$ corresponding to dimensions $\lambda_1, \lambda_2, ...,$ and λ_m, respectively, the range division algorithm is called to output t query sub-ranges $[l_1, u_1], [l_2, u_2], ...,$ and $[l_t, u_t]$.

Step 3: A group of t bloom filters with n-bit vectors and k hash functions are chosen. Here, for each bloom filter, s, i.e. the number of mapped elements, should satisfy $s \leq \lceil \frac{n}{\log n} \rceil$ and k should meet $k = \frac{n}{s} \ln 2$ to ensure the false positive rate $P_f = (1 - (1 - \frac{1}{n})^{ks})^k \approx (1 - e^{-\frac{ks}{n}})^k = n^{-\ln 2}$.

Step 4: For each sub-range $[l_j, u_j]$, $1 < j < t$, all integers between l_j and u_j are mapped into BF_j. Note that, we don't map the integer value $d \in [l_j, u_j]$ itself but its keyed hash value $h(g^{ab} \| d)$ into BF_j. For sub-range $[l_i, u_i]$ with length s_i, where $1 \leq i \leq t$, if $s_i > \lceil \frac{n}{\log n} \rceil$, we divide it into some shorter sub-ranges to ensure the false positive rate. For simplicity, we assume that the length s_i is smaller than $\lceil \frac{n}{\log n} \rceil$.

Step 5: For dimension λ_i, the query user constructs a query set $\{BF_j, E_{ij}(r), C_{ij}\}$, where $i \in [1,m]$, $j \in [1,t]$ and $r \in \{0,1\}$. BF_j is the bloom filter which the j-th sub-range $[l_j, u_j]$ is mapped into. $E_{ij}(r)$ is the OU ciphertext of r, where r is a label to indicate the relationship between the i-th dimension λ_i and the j-th bloom filter BF_j. Concretely, if the j-th sub-range $[l_j, u_j]$ (mapped into BF_j) is a sub-set of the dimension λ_i's query range $[L_i, U_i]$, the r is labeled as 1 ($E_{ij}(r) = E(1)$); otherwise, r is set as 0 ($E_{ij}(r) = E(0)$). C_{ij} is a counter with initial value 0, which is used to record how many IIoT devices of dimension λ_i, whose sensed data is in the j-th sub-range $[l_j, u_j]$. Finally, we get:

$$EM = \begin{bmatrix} E_{11}(r) & E_{12}(r) & \cdots & E_{1t}(r) \\ E_{21}(r) & E_{22}(r) & \cdots & E_{2t}(r) \\ \vdots & \vdots & \ddots & \vdots \\ E_{m1}(r) & E_{m2}(r) & \cdots & E_{mt}(r) \end{bmatrix},$$

$$C = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1t} \\ C_{21} & C_{22} & \cdots & C_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m1} & C_{m2} & \cdots & C_{mt} \end{bmatrix}.$$

Step 6: As shown in Fig. 3, the query user constructs a query request $\{\lambda, BF, EM, C, g^a \bmod p\}$, where $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$ and $BF = \{BF_1, BF_2, ..., BF_t\}$. Then the query request is sent to the edge server. Accordingly, the edge server broadcasts $\{\lambda, g^a \bmod p\}$ to all IIoT devices.

Based on the example given in range division algorithm, the concrete process of the user query request generation algorithm is shown as below:

1) For the 5 sub-ranges $[1,19], [20,39], [40,60], [61,80]$ and $[81,100]$ generated in the example of IV-A, all the values in the sub-ranges are mapped into 5 bloom filters BF_1, BF_2, BF_3, BF_4 and BF_5, respectively. For instance, all the values $1, 2, ...,$ and 19 in the sub-range $[1,19]$ are mapped into BF_1.
2) For the query range $[20,60]$ of dimension λ_1, only $[20,39]$ and $[40,60]$ are its sub-sets. Therefore, only $E_{12}(r)$ and $E_{13}(r)$ are 1. Then, the query user constructs a query set $\{BF_j, E_{1j}(r), C_{1j}\}$, $j \in [1,5]$ and $r \in \{0,1\}$, i.e., $\{\{BF_1, BF_2, BF_3, BF_4, BF_5\}, \{E_{11}(0), E_{12}(1), E_{13}(1), E_{14}(0), E_{15}(0)\}, \{C_{11}, C_{12}, C_{13}, C_{14}, C_{15}\}\}$. Similarly, for the query range $[40,80]$ of dimension λ_2, the query user can also generate a query set correspondingly. Finally, the query request, i.e., two query sets, is generated as shown in Fig. 4.

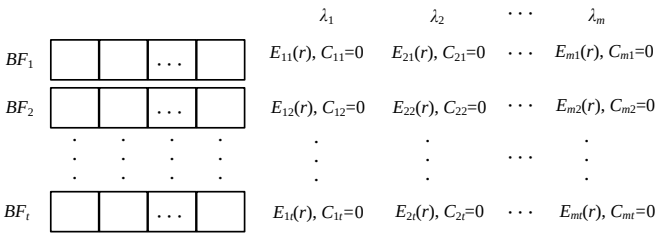


Fig. 3: Query request generation

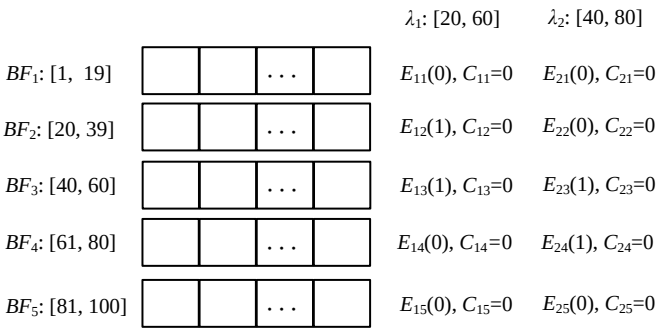


Fig. 4: An example of query request

3) IIoT devices' data response: When receiving the data request $\{\lambda, g^a \bmod p\}$ forwarded by the edge server, each IIoT device $D_k \in D$ checks if its type ID TID_k is in the queried dimensions $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$. If TID_k doesn't belong to λ, D_k discards the data request. Otherwise, D_k computes the shared key $g^{ab} \bmod p$ by using b and $g^a \bmod p$. Then, D_k computes the keyed hash value $h_k = h(g^{ab} \| d_k)$. Finally, $\{TID_k, h_k\}$ is responded to the edge server via a secure channel.

4) Edge server's data aggregation: Upon receiving all the responses $\{TID_k, h_k\}, k \in [1,N]$ from IIoT devices, the edge server performs the following steps to aggregate the data.

Step 1: Edge server takes TID_k from $\{TID_k, h_k\}$ and find the corresponding query dimension λ_i from λ, where $1 \leq i \leq m$.

Step 2: Edge server takes h_k from $\{TID_k, h_k\}$ and checks if h_k is in BF_j, where $BF_j \in BF$ and $j \in [1,t]$. If so, the counter C_{ij} increases by 1. Otherwise, C_{ij} keeps unchanged.

Step 3: After the above processes, the edge server aggregates the data as $C_i = \prod_{j=1}^{t} (E_{ij}(r)^{C_{ij}})$, where $1 \leq i \leq m$, and responses the results $C = \{C_1, C_2, ..., C_m\}$ to the query user.

Following the last example in section IV-B2, suppose that the data responses from IIoT devices are $\{\lambda_1, h(g^{ab}\|5)\}$, $\{\lambda_1, h(g^{ab}\|20)\}$, $\{\lambda_1, h(g^{ab}\|30)\}$, $\{\lambda_2, h(g^{ab}\|35)\}$, $\{\lambda_2, h(g^{ab}\|45)\}$, $\{\lambda_1, h(g^{ab}\|50)\}$, $\{\lambda_1, h(g^{ab}\|55)\}$, $\{\lambda_2, h(g^{ab}\|55)\}$, $\{\lambda_2, h(g^{ab}\|65)\}$, $\{\lambda_2, h(g^{ab}\|90)\}$. The edge server performs the steps as below to aggregate the sensed data.

Step 1: When receiving the data responses, the edge server classifies the dimension λ_1's sensed data $h(g^{ab}\|5), h(g^{ab}\|20)$, $h(g^{ab}\|30)$, $h(g^{ab}\|50)$ and $h(g^{ab}\|55)$, and λ_2's sensed data $h(g^{ab}\|35), h(g^{ab}\|45)$, $h(g^{ab}\|55), h(g^{ab}\|65)$ and $h(g^{ab}\|90)$.

Step 2: It checks the bloom filter which the sensed data belongs to and increases the corresponding counter, e.g., $h(g^{ab}\|5)$ of λ_1 is checked in BF_1, so the corresponding counter C_{11} increases by 1. Finally, the counters $C_{11} = 1, C_{12} = 2, C_{13} = 2, C_{14} = 0, C_{15} = 0, C_{21} = 0, C_{22} = 1, C_{23} = 2$, $C_{24} = 1$ and $C_{25} = 1$.

Step 3: According to $E_{11}(0), E_{12}(1), E_{13}(1), E_{14}(0), E_{15}(0), E_{21}(0), E_{22}(1), E_{23}(1), E_{24}(0)$ and $E_{25}(0)$ and counters, the data is aggregated as:

$$\begin{aligned} C_1 &= \prod_{j=1}^{5}(E_{1j}(r)^{C_{1j}}) \\ &= E_{11}(0)^1 \cdot E_{12}(1)^2 \cdot E_{13}(1)^2 \cdot E_{14}(0)^0 \cdot E_{15}(0)^0 \\ &= E_{11}(0) \cdot E_{12}(2) \cdot E_{13}(2) \cdot 1 \cdot 1 \\ &= E(4) \\ C_2 &= \prod_{j=1}^{5}(E_{2j}(r)^{C_{2j}}) \\ &= E_{21}(0)^0 \cdot E_{22}(0)^1 \cdot E_{23}(1)^2 \cdot E_{24}(1)^1 \cdot E_{25}(0)^1 \\ &= 1 \cdot E_{22}(0) \cdot E_{23}(2) \cdot E_{24}(1) \cdot E_{25}(0) \\ &= E(3) \end{aligned}$$

Finally, $C = \{C_1, C_2\}$ is replied to the query user.

5) Response recovery: When receiving $C = \{C_1, C_2, ..., C_m\}$ from the edge server, the query user recovers the corresponding query results $D' = \{|D'_1|, |D'_2|, ..., |D'_m|\}$ by decrypting C as

$$|D'_i| = Count(D'_i) = Dec(C_i), 1 \le i \le m$$

The correctness of the result is shown as

$$\begin{aligned} Dec(C_i) &= Dec(\prod_{j=1}^{t}(E_{ij}(r)^{C_{ij}})) \\ &= Dec(\prod_{BF_j \in [L_i, U_i]}(E_{ij}(1)^{C_{ij}}) \cdot \prod_{BF_j \notin [L_i, U_i]}(E_{ij}(0)^{C_{ij}})) \\ &= Dec(E(\sum_{BF_j \in [L_i, U_i]}(1 \cdot C_{ij}) + \sum_{BF_j \notin [L_i, U_i]}(0 \cdot C_{ij}))) \\ &= Dec(E(\sum_{BF_j \in [L_i, U_i]} C_{ij})) \\ &= \sum_{BF_j \in [L_i, U_i]} C_{ij} \\ &= |D'_i|. \end{aligned}$$

Following the example in section IV-B4, when receiving $C = \{C_1, C_2\}$, the query user can recover the query result as:

$$\begin{aligned} |D'_1| &= Count(D'_1) = Dec(C_1) = 4. \\ |D'_2| &= Count(D'_2) = Dec(C_2) = 3. \end{aligned}$$

Therefore, the query user knows that there are 4 sensors of dimension λ_1, whose sensed data is in the query range $[20, 60]$ and 3 sensors of dimension λ_2, whose sensed data is in the range $[40, 80]$.

V. Security Analysis

In this section, we analyze the security features of Edge-PPMRQ and prove that it achieves the aforementioned security requirements. We especially focus on privacy-preserving properties as below.

1) *Query ranges $[L_i, U_i], i \in [1, m]$ are privacy-preserving in Edge-PPMRQ*: In order to achieve multi-dimensional range query, the m-dimensional query ranges are divided into t sub-ranges by the proposed range division algorithm. Then, the sub-ranges are mapped into t bloom filters to generate the query request. Subsequently, the query user transfers the query request to the edge server via a public channel. Through the channel, an IIoT device D_k can eavesdrop the query request. At the same time, $D_k \in D$ owns shared key g^{ab} mod p. Therefore, for $\forall d \in [1, n]$, D_k can compute all keyed hash values $h(g^{ab}\|d)$ and check the bloom filter which $h(g^{ab}\|d)$ belongs to. In such way, D_k can recover the sub-range $[l_j, u_j]$ which is mapped into BF_j, $j \in [1, t]$. However, D_k cannot distinguish the ciphertexts of 0 and 1 because OU cryptosystem is semantically secure [**?**], i.e., it cannot know the real value of r in $E_{ij}(r)$. As a result, D_k cannot confirm if the query sub-range $[l_j, u_j]$ is a sub-set of the query range $[L_i, U_i]$. Due to the same reason, even though D_k knows sub-range $[l_j, u_j]$ is mapped into BF_j, D_k still cannot get any information about the query range $[L_i, U_i]$ of dimension λ_i. Besides, the edge server can neither get shared key g^{ab} mod p nor distinguish the ciphertexts of 0 and 1. Therefore, it cannot get any information about $[l_j, u_j]$ and $[L_i, U_i]$. Based on above analysis, the query ranges $[L_i, U_i]$ of dimension λ_i, $i \in [1, m]$ are privacy-preserving for IIoT devices and the edge server in Edge-PPMRQ.

2) *The query result $D' = \{|D'_1|, |D'_2|, ..., |D'_m|\}$ is also privacy-preserving in Edge-PPMRQ*: For the edge server, it receives the query request $\{\lambda, BF, EM, C, g^a \text{ mod } p\}$ from the query user and the data response $\{TID_k, h_k\}$, $k \in [1, N]$, from IIoT devices. Even though it can know the bloom filter which the keyed hash value $h_k = h(g^{ab}\|d_k)$ belongs to, it cannot confirm if a sub-range $[l_j, u_j]$, which is mapped into the bloom filter BF_j, is the sub-set of the query range $[L_i, U_i]$. The reason is that it cannot distinguish the ciphertexts of 0 and 1, according to the semantic security of OU cryptosystem [**?**]. Therefore, the edge server cannot confirm if the sensed data d_k of IIoT device D_k is in the query range $[L_i, U_i]$ of dimension λ_i, $i \in [1, m]$, i.e., it has no idea about that how many

IIoT devices whose sensed data is in range $[L_i, U_i]$. In other words, the query result D' is privacy-preserving for the edge server. Besides, each IIoT device $D_k \in D$ owns g^{ab} mod p and can compute all keyed hash values of data in $[1, n]$. However, every IIoT device transfers its data response via a secure channel. As a result, D_k's sensed data d_k cannot be recovered by other IIoT devices. At the same time, $E_{ij}(r)$ is also indistinguishable for IIoT devices. Thus, IIoT devices have no way to know the query result D'. Based on the above analysis, we prove that the query result D' is privacy-preserving for the edge server and IIoT devices.

3) *Plaintext data d_k of IIoT device D_k is also privacy-preserving in Edge-PPMRQ*: For the edge server, it can get g^a mod p from the query request. However, b is the private parameter of IIoT devices. Therefore, the edge server cannot compute g^{ab} mod p. In other words, it cannot recover d_k from $h_k = h(g^{ab} \| d_k)$. Similar to some previous work, in our model, the IIoT devices send their data response via a secure channel. As a result, D_k's sensed data d_k cannot be recovered by other IIoT devices. Likewise, the query user also cannot recover the sensed data of IIoT devices. Based on the above analysis, the plaintext data d_k of IIoT device D_k is also privacy-preserving.

VI. Performance Evaluation

In this section, we evaluate the performance of Edge-PPMRQ and three related work, i.e., BFPRQ [**?**], CEPRQ [**?**] and URPRQ [**?**] from the aspects of communication overhead and computational cost. Here, it should be noted that Edge-PPMRQ and BFPRQ [**?**] are based on OU cryptosystem [**?**] and Paillier cryptosystem [**?**], respectively, while CEPRQ [**?**] and URPRQ [**?**] are based on their own constructed homomorphic encryption SHE. All the schemes are simulated with Python 3.8, Gmpy 2 and Math library. The experiments are carried out on an Intel(R) Core (TM) i5-7400 CPU @3.00GHz with Windows 10 system and 24GB RAM. To ensure the fairness, we keep the false positive rate of bloom filters in BFPRQ [**?**] and Edge-PPMRQ as $n^{-\ln 2}$. Besides, we simulate m-dimensional range query of BFPRQ [**?**], CEPRQ [**?**] and URPRQ [**?**] by performing m single-dimensional query requests since they don't support multi-dimensional range query. Furthermore, we evaluate and compare the communication overhead and computational cost of these four schemes in two conditions: (1) n is varying from 2^{10} to 2^{20}, while m is fixed at 16; (2) m is varying from 5 to 55, while n is fixed at 2^{20}. In addition, the detailed parameters setting and notations used in the comparison are shown in Table II.

TABLE II: The parameters setting and notations

Parameter	Value
κ	$\kappa = 512, \|p\| = \|q\| = \kappa$
h	The public hash function SHA-256
k	The number of hash functions k= log$n$×ln2
N	The number of single dimension's IIoT devices. N=1000
$\|\lambda_i\|$	The length of dimension index with the length of 6 bits
m	The number of dimensions in the query request
b	log$n$
$\|E_{paillier}\|$	The length of paillier ciphertext
$\|E_{SHE}\|$	The length of SHE ciphertext
$\|E_{OU}\|$	The length of OU ciphertext
$\|H\|$	The length of SHA-256's output

A. Communication overhead

In this section, we compare the communication overhead of BFPRQ [**?**], CEPRQ [**?**], URPRQ [**?**] and Edge-PPMRQ. The communication overhead between the edge server and the query user includes both the query request and the query response. Besides, the communication overhead between the edge server and IIoT devices consists of the data request and the data response.

For condition 1, we calculate the communication overhead of the four schemes between the edge server and the query user, and between the edge server and IIoT devces, which are respectively shown in table III and table IV. In order to compare them more intuitively, Fig. 5(a) and Fig. 5(b) depict the communication overhead in condition 1. From the Fig. 5(a), we can see that the communication overhead between the edge server and the query user in Edge-PPMRQ keeps very low with n varying from 2^{10} to 2^{20}, while in BFPRQ [**?**], CEPRQ [**?**] and URPRQ [**?**], they grow rapidly. Fig. 5(b) shows the communication overhead comparison between the edge server and IIoT devices, from which we can know that both Edge-PPMRQ and BFPRQ [**?**] are equally efficient and perform significantly better than CEPRQ [**?**] and URPRQ [**?**].

For condition 2, we also analyze the communication overhead of the four schemes between the edge server and the query user, and between the edge server and IIoT devices, which are respectively shown in table V and table VI. Similarly, Fig. 5(c) and Fig. 5(d) intuitively present the communication overhead of the four schemes in condition 2. From Fig. 5(c), the communication overhead between the edge server and the query user of the four schemes all show a linear growth trend, but the communication cost curve of Edge-PPMRQ is almost flat with m varying from 5 to 55, while the slopes of other schemes, BFPRQ [**?**], CEPRQ [**?**] and URPRQ [**?**], are much greater than that of Edge-PPMRQ. The reason is that Edge-PPMRQ achieves multi-dimensional range query by a query request, which saves a large number of communication costs for multi-dimensional query range. Fig. 5(d) depicts the communication overhead between the edge server and IIoT devices. From the figure, we find that the communication overhead in both Edge-PPMRQ and BFPRQ [**?**] keeps efficient, but the communication costs of CEPRQ [**?**] and URPRQ [**?**] are almost 8 times and 50 times of that in Edge-PPMRQ, respectively. This is because in CEPRQ [**?**] and URPRQ [**?**], IIoT devices not only receive the query request forwarded by the edge server, but also send ciphertext response back, which results in a large amount of communication overhead, while IIoT devices of Edge-PPMRQ and BFPRQ [**?**] only send keyed hash values $h(g^{ab} \| d_k)$ and dimension index λ_i to the edge server, which are far shorter than ciphertext.

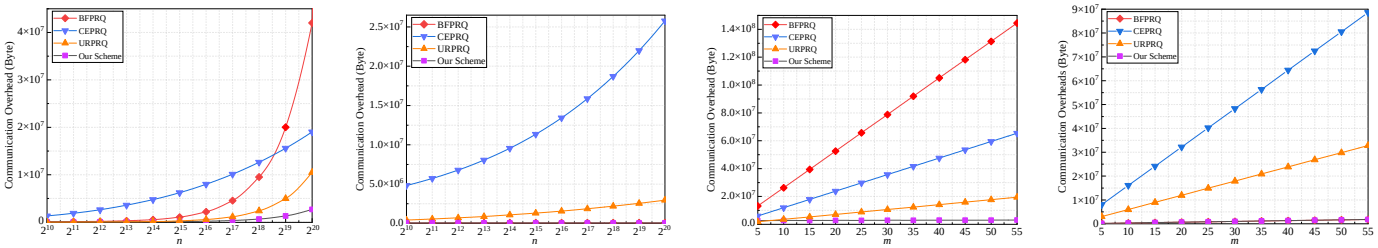
Based on the above comparisons, we can conclude that Edge-PPMRQ is remarkably communication-efficient.

TABLE III: Communication overhead between the edge server and the query user with varying n

Schemes	Query request (bits)	Query result (bits)	Total overhead (bits)
BFPRQ	$m \cdot [b \cdot (n + \lvert E_{paillier} \rvert) + \lvert \lambda_i \rvert)]$	$m \cdot (\lvert E_{paillier} \rvert + \lvert \lambda_i \rvert)$	$16 \cdot b \cdot n + 32768 \cdot b + 32864$
CEPRQ	$m \cdot [(2 + \frac{2 \cdot b^3 + 3 \cdot b^2 - 11 \cdot b}{6}) \cdot \lvert E_{SHE} \rvert + \lvert \lambda_i \rvert]$	$m \cdot (\lvert E_{SHE} \rvert + \lvert \lambda_i \rvert)$	$\frac{2560}{3} \cdot b^4 + \frac{6400}{3} \cdot b^3 + \frac{10240}{3} \cdot b^2 + \frac{124160}{3} \cdot b + 46176$
URPRQ	$m \cdot [(b-1) \cdot (b+2) \cdot \lvert E_{SHE} \rvert + \lvert \lambda_i \rvert]$	$m \cdot (\lvert E_{SHE} \rvert + \lvert \lambda_i \rvert)$	$2560 \cdot b^3 + 2560 \cdot b^2 + 96$
Edge-PPMRQ	$b \cdot (n + m \cdot \lvert E_{OU} \rvert) + m \cdot \lvert \lambda_i \rvert$	$m \cdot (\lvert E_{OU} \rvert + \lvert \lambda_i \rvert)$	$b \cdot n + 24576 \times b + 24672$

TABLE IV: Communication overhead between the edge server and IIoT devices with varying n

Schemes	Data request (bits)	Data response (bits)	Total overhead (bits)
BFPRQ	$m \cdot \lvert \lambda_i \rvert$	$m \cdot N \cdot (\lvert H \rvert + \lvert \lambda_i \rvert)$	4192096
CEPRQ	$m \cdot [(2 + \frac{2 \cdot b^3 + 3 \cdot b^2 - 11 \cdot b}{6}) \cdot \lvert E_{SHE} \rvert + \lvert \lambda_i \rvert]$	$m \cdot N \cdot (\lvert E_{SHE} \rvert + \lvert \lambda_i \rvert)$	$\frac{2560}{3} \cdot b^4 + \frac{6400}{3} \cdot b^3 + \frac{10240}{3} \cdot b^2 + \frac{124160}{3} \cdot b + 46176$
URPRQ	$m \cdot [(b-1) \cdot (b+2) \cdot \lvert E_{SHE} \rvert + \lvert \lambda_i \rvert]$	$m \cdot N \cdot (\lvert E_{SHE} \rvert + \lvert \lambda_i \rvert)$	$2560 \cdot b^3 + 2560 \cdot b^2 + 96$
Edge-PPMRQ	$m \cdot \lvert \lambda_i \rvert$	$m \cdot N \cdot (\lvert H \rvert + \lvert \lambda_i \rvert)$	4192096



(a) Between the edge server and the query user with varying n (b) Between the edge server and IIoT devices with varying n (c) Between the edge server and the query user with varying m (d) Between the edge server and IIoT devices with varying m

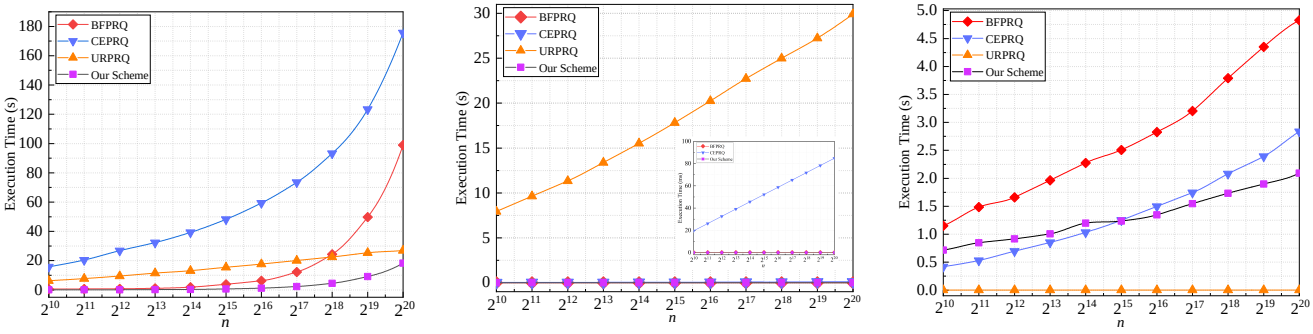
Fig. 5: Comparison of communication overhead

TABLE V: Communication overhead between the edge server and the query user

Schemes	Query request (bits)	Query result (bits)	Total overhead (bits)
BFPRQ	$m \cdot [b \cdot (n + \lvert E_{paillier} \rvert) + \lvert \lambda_i \rvert)]$	$m \cdot (\lvert E_{paillier} \rvert + \lvert \lambda_i \rvert)$	$21014534 \cdot m$
CEPRQ	$m \cdot [(2 + \frac{2 \cdot b^3 + 3 \cdot b^2 - 11 \cdot b}{6}) \cdot \lvert E_{SHE} \rvert + \lvert \lambda_i \rvert]$	$m \cdot (\lvert E_{SHE} \rvert + \lvert \lambda_i \rvert)$	$9518886 \cdot m$
URPRQ	$m \cdot [(b-1) \cdot (b+2) \cdot \lvert E_{SHE} \rvert + \lvert \lambda_i \rvert]$	$m \cdot (\lvert E_{SHE} \rvert + \lvert \lambda_i \rvert)$	$858124 \cdot m$
Edge-PPMRQ	$b \cdot (n + m \cdot \lvert E_{OU} \rvert) + m \cdot \lvert \lambda_i \rvert$	$m \cdot (\lvert E_{OU} \rvert + \lvert \lambda_i \rvert)$	$30726 \cdot m + 20971520$

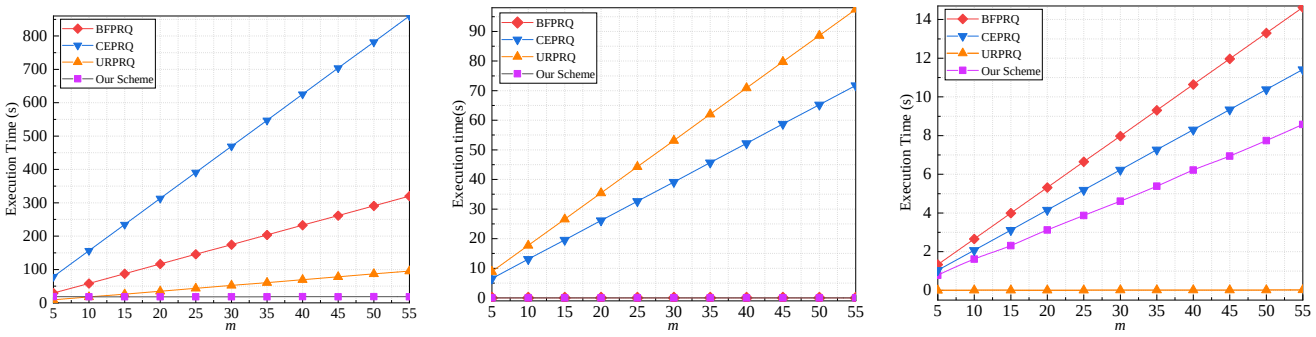
TABLE VI: Communication overhead between the edge server and the IIoT devices

Schemes	Data request (bits)	Data response (bits)	Total overhead (bits)
BFPRQ	$m \cdot \lvert \lambda_i \rvert$	$m \cdot N \cdot (\lvert H \rvert + \lvert \lambda_i \rvert)$	$262000 \cdot m$
CEPRQ	$m \cdot [(2 + \frac{2 \cdot b^3 + 3 \cdot b^2 - 11 \cdot b}{6}) \cdot \lvert E_{SHE} \rvert + \lvert \lambda_i \rvert]$	$m \cdot N \cdot (\lvert E_{SHE} \rvert + \lvert \lambda_i \rvert)$	$12881520 \cdot m$
URPRQ	$m \cdot [(b-1) \cdot (b+2) \cdot \lvert E_{SHE} \rvert + \lvert \lambda_i \rvert]$	$m \cdot N \cdot (\lvert E_{SHE} \rvert + \lvert \lambda_i \rvert)$	$2048000 \cdot m$
Edge-PPMRQ	$m \cdot \lvert \lambda_i \rvert$	$m \cdot N \cdot (\lvert H \rvert + \lvert \lambda_i \rvert)$	$262000 \cdot m$



(a) Time cost of user side (b) Time cost of IIoT side (c) Time cost of edge server side

Fig. 6: Computational time cost comparison with varying n



(a) Time cost of user side (b) Time cost of IIoT side (c) Time cost of edge server side

Fig. 7: Computational time cost comparison with varAying m

TABLE VII: Comparison between Paillier and OU

Algorithm/Operations	Encryption(ms)	Decryption(ms)	Homomorphic Addition(ms)	Scalar Multiplication(ms)
Paillier	12.4	12.4	0.023	0.096
Okamoto Uchiyama	10.5	1.8	0.012	0.062

TABLE VIII: Functional comparison

Schemes	Edge-PPMRQ	BFPRQ [?]	CEPRQ [?]	URPRQ [?]
Multi-dimension	√	×	×	×
Discotinuous range	√	√	×	√
Arbitrary boundary range	√	√	×	√
Counting function	√	√	√	√
Sum function	×	×	√	√
Suitability for IIoT applications	√	√	×	×

B. Computational cost

In this section, we compare the execution time of the query user, IIoT devices and the edge server with varying n and m, respectively, and the experiment comparison results are taken from the average value of 10 times simulations. Our scheme uses OU cryptosystem, while some of the compared schemes utilize Paillier cryptosystem. Therefore, we give the time cost comparison between Paillier and OU in Table VII. From the table, we can see that OU cryptosystem is more efficient than Paillier cryptosystem.

Fig. 6 shows the average computational time of BFPRQ [?], CEPRQ [?], URPRQ [?] and Edge-PPMRQ in condition 1. Specifically, Fig. 6(a) illustrates the execution time of the query user, including user query request generation and response recovery. From the figure, we can see that Edge-PPMRQ is really efficient, while the execution time of BFPRQ [?], CEPRQ [?] and URPRQ [?] increases rapidly with n varying from 2^{10} to 2^{20}. For example, when $n = 2^{20}$, the execution time of BFPRQ [?] and CEPRQ [?] are 4 times and 7 times of that in Edge-PPMRQ, respectively. URPRQ [?] also obviously consumes more execution time than Edge-PPMRQ. The reason for the excellent performance of Edge-PPMRQ is that Edge-PPMRQ maps multiple query ranges into a group of bloom filters, so that the range query requests for multiple dimensions can be realized by only one query request, while in other schemes, the query user has to send multiple query requests to achieve the same purpose. Fig. 6(b) illustrates the execution time of IIoT devices. It shows that in URPRQ [?], IIoT devices undertake considerable computational costs, while the execution time of BFPRQ [?], CEPRQ [?] and Edge-PPMRQ is negligible. Furthermore, a sub-figure in Fig. 6(b) is given to better illustrate the comparison among BFPRQ [?], CEPRQ [?] and Edge-PPMRQ. From the sub-figure, we can see that the execution time of CEPRQ [?] also grows more rapidly than that of Edge-PPMRQ and BFPRQ [?] increases. The reason for the result is that in CEPRQ [?] and URPRQ [?], IIoT devices perform a lot of time-consuming operations, i.e., homomorphic addition and homomorphic xor, while in Edge-PPMRQ and BFPRQ [?], IIoT devices only compute the keyed hash value, which is much more efficient than homomorphic operations. Fig. 6(c) indicates the execution time of the edge server. From the figure, we can observe that Edge-PPMRQ, CEPRQ [?] and URPRQ [?] are more efficient than BFPRQ [?], and Edge-PPMRQ performs better than CEPRQ [?] when n is bigger than 2^{15}. Although the computational cost of the edge server in URPRQ [?] keeps very low, it is at the cost of a large amount of computational overhead for IIoT devices according to Fig. 6(b).

Fig. 7 depicts the average computational time of BFPRQ [?], CEPRQ [?], URPRQ [?] and Edge-PPMRQ in condition 2. Specifically, Fig. 7(a) shows the execution time of the query user, including two parts, i.e., user query request generation and response recovery. We can find that Edge-PPMRQ costs nearly constant execution time, which is apparently efficient than BFPRQ [?], CEPRQ [?] and URPRQ [?]. This is because the query user in Edge-PPMRQ performs less homomorphic operations than BFPRQ [?], CEPRQ [?] and URPRQ [?]. Fig. 7(b) depicts IIoT devices' data response time, which shows that both Edge-PPMRQ and BFPRQ [?]

are more efficient than CEPRQ [**?**] and URPRQ [**?**]. The reason is that IIoT devices in Edge-PPMRQ and BFPRQ [**?**] only need compute keyed hash values $H(g^{ab}\|d_i)$, while both in CEPRQ [**?**] and URPRQ [**?**], IIoT devices need perform a large number of homomorphic multiplication operations. Fig. 7(c) presents the data aggregation time cost of the edge server, which shows that Edge-PPMRQ performs better than BFPRQ [**?**] and CEPRQ [**?**] since in Edge-PPMRQ, the number of homomorphic operations are less than that of BFPRQ [**?**] and CEPRQ [**?**]. Obviously, URPRQ [**?**] consumes the least computational time. The reason is that the most computational cost is afforded by IIoT devices, and the edge server only performs homomorphic addition operations.

C. *Functional comparison*

Based on above introduction and analysis, a functional comparison of Edge-PPMRQ and related schemes is given in table VIII.

From a functional point of view, Edge-PPMRQ supports the functions of multi-dimensional, discontinuous and arbitrary-boundary range query, while BFPRQ [**?**], CEPRQ [**?**] and URPRQ [**?**] cannot support multi-dimensional range query. Additionally, CEPRQ [**?**] also cannot support discontinuous range query or arbitrary-boundary range query. Moreover, with respect to query functions, all the 4 schemes support the Counting function, i.e., the query user can get the number of the sensed data located in the query range. Besides, CEPRQ and URPRQ can also achieve sum function, i.e., the query user can obtain the sum of the sensed data within the query range.

From a real application perspective, edge/fog computing is introduced for decreasing the load of IIoT devices and reducing the delay of service. Therefore, in edge/fog-supported IIoT applications, some tasks of IIoT devices can be migrated to edge server/fog node, which not only remarkably extends the life period of IIoT devices, but also provides nearly real-time service. According to the comparison in the Fig. 5, Fig. 6 and Fig. 7, the costs of IIoT devices in Edge-PPMRQ and BFPRQ [**?**] evidently keeps very low, while in CEPRQ [**?**] and URPRQ [**?**], IIoT devices process a larger proportion of communication and computational tasks. Therefore, Edge-PPMRQ and BFPRQ [**?**] are considerably suitable for IIoT applications.

In a word, Edge-PPMRQ not only is functionally powerful for multi-dimensional, discontinuous and arbitrary boundary range queries, but also is significantly suitable for IIIoT applications.

VII. Conclusion and future work

Based on our proposed range division algorithm, this paper has designed a privacy-preserving multi-dimensional range query scheme for edge-supported IIoT. The scheme achieves the function of multi-dimensional range query, i.e. a user can query different types of data at once, which is very suitable for the real application of IIoT environment. Meanwhile, it also supports the continuous, discontinuous and arbitrary-boundary range queries. The security analysis proves that Edge-PPMRQ is privacy-preserving, i.e. the query ranges and the query results cannot be revealed by any entities except the query user, and the sensed data of each IIoT device cannot be recovered by other parties. Furthermore, a large number of experiments are conducted to evaluate and compare the performance of Edge-PPMRQ and other related work, and the results show that Edge-PPMRQ is really communication and computationally efficient. Comprehensively, Edge-PPMRQ achieves expected goals in aspects of functions, privacy preservation and efficiency.

In future work, we plan to study more efficient range query schemes for various functions, privacy requirements and multiple application scenarios, e.g., multi-user range query scheme.