A Sharding-Based Approach for Enhancing Efficiency in ISSDOs for Sharing Scattered Values

Reza Ghasemi
Bu-Ali Sina University
Department of Mathematics, Faculty of Sciences
Email: r.ghasemi@basu.ac.ir

***Abstract*—Data outsourcing is a solution aimed at addressing the security and reliability issues of data storage by ensuring professional handling of the data. The growing use of outsourcing is causing concern among users due to the lack of assurance regarding the security and reliability of data stored on servers. To address these issues, some attempts have been made to implement Secret Sharing-based Data Outsourcing (SSDO) schemes. The low efficiency of these schemes led researchers to use an index server (IS). However, IS are susceptible to frequency analysis. Bucket-Chain B^+Tree (BCB^+Tree) was proposed to tackle the frequency analysis in the current Index Server Secret Sharing-based Data Outsourcing (ISSDO) schemes. Nevertheless, this scheme works very well when the data is discrete with a limited range. Otherwise, the scheme's efficiency declines significantly as it has to store one index in each bucket and the number of buckets rises significantly, rendering the use of an IS useless. In this paper, a new data structure is proposed to store the indexes in IS to mitigate this efficiency concern. Briefly, the domain of values is divided into several segments, and indexes of values in each segment are stored inside a $Shard$. Additionally, a data outsourcing scheme has been presented based on the proposed data structure. It can withstand collaboration from up to $k-1$ dishonest servers even if they have access to the IS.**

I. Introduction

Data is considered to be a valuable asset for both individuals and organizations. Any unauthorized exposure or leakage of this data can result in serious consequences, including financial and reputational losses. Therefore, it is essential for companies to employ experts in data security to safeguard their sensitive information. Similarly, individuals are also interested in learning about the best practices and strategies to ensure the protection of their confidential data.

Despite advancements in data security, daily data breaches still occur due to outdated knowledge and systems [1]. To address this, some companies offer Data-as-a-Service (DaaS), where they handle the data storage for their customers, reducing hardware and software costs. These companies hire skilled professionals and regularly update their systems to ensure robust protection against attacks. However, outsourcing data to DaaS providers raises privacy concerns, as they may extract metadata and potentially sell it [2]. When privacy is a priority, these current data outsourcing methods may not be suitable.

Outsourcing raw data to an untrusted party may not fully address privacy and reliability concerns. Encrypting the data and outsourcing it to DaaS providers can address privacy concerns but reduces scalability [3]. Symmetric encryption allows for simple queries but requires frequent interactions with the data owner, affecting performance. Asymmetric encryption allows the database to perform operations without the owner's help, but increases computation and becomes impractical as data size grows [4, 5]. Ultimately, encryption-based, namely symmetric and asymmetric, schemes may not be practical when outsourcing data and requiring the database to perform operations.

Data outsourcing schemes based on Secret sharing (SSDO) emerged as a possible solution for data outsourcing. In these schemes, data is shared among several servers. Each server receives a piece of information, called share, such that solely predefined subsets of these servers can recover the shared data with the help of their shares. Therefore, if some servers lose stored shares, the shared data is still recoverable, ensuring reliability. The main advantage of such schemes is that data servers can answer most, if not all, types of users' queries [6]. Recently, the weakness of SSDOs has been highlighted by successful attacks [7]. This weakness stems from the use of a fixed secret vector for all secrets in the sharing process. Some authors [8] have proposed using ISs to address this issue. In this scheme, denoted by ISSDO, each record is assigned a unique secret vector and an index. Then, generated shares and the index are sent to the data servers. In addition, there is an index server that stores these generated indexes. Ghasemi [9] has demonstrated that, despite the benefits of index trees, these schemes remain vulnerable. Bahrami et. al. [10] proposed Bucket Chain B^+tree (BCB^+tree) to hide the frequency of shared records that can be leveraged to recover the shared records. In their scheme, a chain of buckets is used to store the indexes corresponding to records that have the same value.

While Bahrami et al.'s scheme can handle multiple queries, it can be observed that When records have dispersed values, running time increases substantially since for each record a bucket should be assigned. As a result, scattered values incur high communication costs which slow down the scheme's performance and increase the running time. Consequently, leveraging BCB^+tree will not bring benefit to the scheme greatly when the values are dispersed. In this paper, an ISSDO scheme enjoying a new data structure for saving the indexes is presented to handle scattered values. The domain of the values is partitioned and each of these partitions is assigned a $shard$ that stores indexes of all records in the corresponding

partition. The reduction of communication cost, which is the primary bottleneck in running time, facilitates the scheme to enhance its performance.

The main contributions of this paper are,

1) Presenting a new data structure for saving the indexes.
2) Proposing an efficient and secure scheme to handle scattered values in records.

The present paper is structured as follows. The succeeding section constitutes the problem statement. The subsequent section, referred to as preliminaries, presents the necessary primitives. A novel ISSDO, capable of effectively handling scattered values, is proposed in the Section four. In the subsequent section, the proposed approach's scalability is demonstrated through various implementations. The paper concludes with a final section. The security proof has been put in an Appendix. Interested readers are referred to see Appendix A for formal proof of why the proposed scheme is $k-1$-secure (Appendix is not included in the ePrint version. A full version of this paper will be published later.).

II. Problem Statement

Data outsourcing is crucial in data management as it addresses the issue of insufficient data privacy knowledge and high hardware costs faced by individuals. As a result, data owners opt for outsourcing their data instead of storing it locally. Sharing data among multiple servers can enhance security and reliability, and secret sharing techniques have been utilized to develop new ISSDOs schemes.

A. Raw Data

A data owner with sensitive information desires to outsource its data for the sake of security and reliability. Assume that the data has the following format,

#	Name	SIN Number	Age	Salary	Weight (gr.)
1	Sarah	R5234D56	28	45k	57463
2	Jack	G556F35	48	52k	75938
3	David	S098A34	21	39k	69884
4	John	S930F44	52	72k	101075
5	Thomas	K898L88	61	63k	81640
6	Daniel	R117L08	13	0k	48523

TABLE I: Confidential raw data which data owner wants to outsource.

Table (I) is the raw data. Rows and columns represent records and features, respectively. For clarity of presentation, we solely focus on sharing one feature, say *Weight*. Note that the proposed schemes can easily be adapted to share more features. Weight can be considered as a scattered data as it ranges from $2k$ to $150k$ kilograms.

B. Main structure and threat model

ISSDOs enjoy an index server that stores indexes assigned to each record. Consequently, there is no need to use a fixed secret vector, improving the security and reliability of the schemes. Interested readers are referred to read [9] paper for more details of how assigning different secret vectors can secure the scheme against current frequency analysis attacks.

Generally, ISSDOs include four different entities, depicted in Fig. 1.

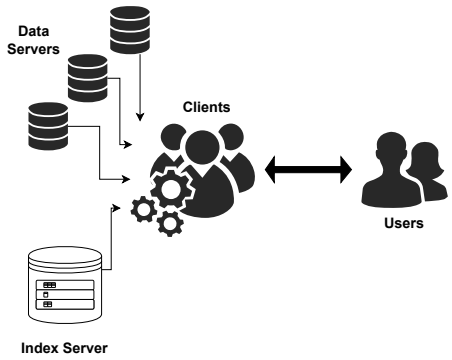

Fig. 1: General structure of ISSDO schemes.

- **Data servers (DS):** Data servers are semi-honest entities that store the generated shares in the sharing process. They might collaborate with each other to infer meaningful information from the stored shares.
- **Index server (IS):** Index server stores indexes of records, improving the scheme's performance in terms of both speed and security. The IS is considered semi-honest and may work with corrupt DSs to obtain information about the shared records.
- **Clients (CL):** Clients serve as a trusted intermediary connecting all parties. They store confidential information necessary for sharing and retrieval. In addition to data sharing and retrieval, clients respond to inquiries after communicating with IS and DSs.
- **Users:** Users' queries will be answered by sending their queries to the clients.

C. Problem

Bahrami et. al. [10] proposed a new data structure, the BCB^+tree, for storing indexes in an index tree. This structure significantly improves running time and strengthens the scheme's security against potential attacks. The BCB^+tree dedicates a chain of buckets to a value and stores the corresponding indexes within a chain of buckets. Only clients, equipped with a secret key, can identify the relation between the buckets and retrieve all indexes associated with a specific value. However, this scheme only works effectively when the values are not dispersed. if values are scattered, a large number of chains with only one stored index are generated, leading to increased communication between the IS and clients. This significantly reduces the scheme's performance due to the fact that clients need numerous communication with IS to handle queries which is a bottleneck in running time. Scattered values are very common and should not be neglected. This paper proposes a new efficient and secure ISSDO scheme to address records with scattered values.

III. Preliminaries

In this section, secret sharing is reviewed because it is used in sharing process in ISSDO schemes.

A. Shamir Secret Sharing

When handling confidential information, it is imperative to prioritize both security and reliability. For example, losing an encryption key for a database can have catastrophic consequences, as data recovery would then become impossible. To mitigate this risk, it is necessary to employ a method for securely storing the sensitive information that keeps it protected from unauthorized access, yet also allows for its recovery by authorized entities when needed. Shamir [11] and Blakely [12] proposed secret sharing scheme in which a secret is shared among some participants by assigning a share to each of them such that predefined subsets of participants can recover the shared secret through putting their shared together. The creation of these shares is designed to ensure two key objectives: confidentiality and recoverability. Firstly, each share, or even some combination of shares, should not reveal significant information to the participants. Secondly, in the event that some participants lose their shares, the secret remains recoverable through the predefined combination of the remaining shares. These features reinforce both security and reliability.

Shamir proposed a (k, n)-threshold secret sharing scheme where k, n are positive integers and $k \leq n$. In his scheme, k represent threshold which is the minimum shares required to recover the shared secret. The parameter n denotes the number of all participants that receive a share after sharing process.

1) Sharing stage: In order to share a value v among n participants $P_1, \ldots, P_n$, a polynomial $f(x) \in \mathbb{F}[x]$ of degree $k-1$ is selected randomly, where $f(0) = v$. Then, every P_i $(i = 1, \ldots, n)$ is given $share_i(v) = f(i)$ as their shares.

2) Recovering stage: In order to recover the secret v, a coalition of k shares are needed. For simplicity, assume k participants $P_1, \ldots P_k$ decide to recover the secret. They will put their shares and using interpolation the secret can be recovered as follows,

$$v = f(0) = \sum_{i=1}^{k} f(i)\ell_i \quad , \quad \ell_i = \prod_{\substack{j=1 \\ i \neq j}}^{k} \frac{j}{j-i}$$

In secret sharing, all computations must be done in a finite field $\mathbb{Z}_p$ for a large prime number p. Therefore, in this paper, all computations are in a finite field.

B. Pseudorandom generators

In the proposed scheme, pseudorandom generators are used. Therefore, it is briefly explained here.

Definition 1. *[13] Let $\ell(\cdot)$ be a polynomial and let G be a deterministic polynomial time algorithm such that for any input $s \in \{0,1\}^n$, algorithm G outputs a string of length $\ell(n)$. We say that G is a pseudorandom generator (PRG) if the following two conditions hold:*

1) *For every n it holds that $\ell(n) > n$.*
2) *For all probabilistic polynomial-time distinguishers D, there exists a negligible function $negl$ such that:*

$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]| \leq negl(n)$$

where r is chosen uniformly at random from $\{0,1\}\ell(n)$, the seed s is chosen uniformly at random from $\{0,1\}^n$, and the probabilities are taken over the random coins used by D and the choice of r and s.

Approximately, pseudorandom generators produce sequences that are indistinguishable from actual random sequences from the perspective of polynomial observers.

C. Security definition

We adopt security definition from [10] paper. Roughly speaking, we call a scheme r-secure w.r.t this definition, if r data server collaborate, they cannot extract meaningful information about the shared data while accessing the index server.

Definition 2. *[10] A secret sharing based data outsourcing is called r-secure if any probabilistic polynomial time (PPT) adversary ($\mathcal{A}$) has negligible advantage in winning the following game,*

1) *Challenger runs $Setup(1^\lambda, n, k)$ and shares a random database D.*
2) *$\mathcal{A}$ receives the shares stored on r data servers and the index tree.*
3) *$\mathcal{A}$ selects two different values v_0 and v_1 and sends them to the challenger.*
4) *Challenger selects at random $b \in \{0,1\}$ and shares v_b.*
5) *Challenger sends the updated index tree and generated shares regarding the r servers to $\mathcal{A}$.*
6) *$\mathcal{A}$ outputs its guess of b represented by $out_\mathcal{A}$.*

The advantage of $\mathcal{A}$ is,

$$\mathrm{Adv}_{\mathcal{A},\mathrm{r}}(\lambda) = |\mathrm{Pr}[\mathrm{out}_\mathcal{A} = \mathrm{b}] - \frac{1}{2}|$$

IV. A FAST AND SECURE SECRET SHARING BASED DATA OUTSOURCING

In this section, a secure and efficient ISSDO is presented which mitigates the concern of dealing with scattered data. Roughly speaking, confidential features which are scattered are shared via Shamir's secret sharing. Then, an index is generated and stored in an index server for each record. In order to store indexes, the client partitions the domain of the confidential attribute, and for each segment generates a *Shard*, see Fig. (3). Each shard contains all indexes corresponding to the records whose confidential attribute's value lies in the segment of which the *Shard* has been generated. Indexes inside a shard are stored in a chain of buckets, increasing the security of the scheme against frequency analysis. Note that only clients who have access to a secret key can identify which bucket belongs to which shard. Consequently, an outsider cannot identify the relationship between these buckets.

A. The structure of the proposed ISSDO

Our scheme consists of four main entities, Data servers (DSs), Index Server (IS), Clients (CLs), and users as depicted in Fig. (1). There are two phases, namely *Data Sharing* and *Answering queries*. In the first phase, all records are shared among data servers. In the second phase, upon receiving a query, the client translates the query for the index server and relevant DSs. Following potential interactions with the index server and DSs, the client provides the answer to the user's query.

B. Data Sharing

During this phase, the four entities exchange information or data among themselves. Data sharing contains three stages, I) Generating parameters, II) Generating shares, and III) Adding indexes to IS.

1) Generating parameters: Before sharing any record, the following steps should be taken. At the first step, a probabilistic polynomial algorithm $Setup(1^{\lambda}, n, k)$ is run where λ, n and k are security parameter, the number of DSs and threshold, respectively. This algorithm outputs the following items,

- Finite field $\mathbb{Z}_p$ in which all computations are done.
- Secret key sk.
- Secret vector $X = \{x_1, \ldots, x_n\}$
- Partition $D_p = \{D_1, D_2 \ldots, D_r\}$ of $\mathbb{Z}_p$ where,
  1) Order is preserved. In other words, for all $x \in D_i$ and $x' \in D_j$, $x \leq x'$ iff $i \leq j$. See Ex. 1.
  2) For each $i, j \in \{1, 2, \ldots, r\}$,

$$\sum_{x \in D_i} P(x) = \sum_{x \in D_j} P(x)$$

Example 1. *Suppose we have a normal distribution, and we want to divide its domain into four sections in a manner that maintains the relative order of values and ensures that each section contains an equal probability of a randomly selected value belonging to it. As it can be observed, this might result in unequal segment lengths.*

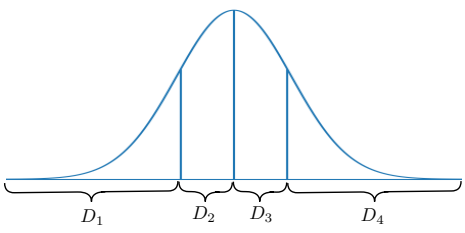

Fig. 2: Partitioning into four segments. Based on this partitioning into four segments, the index server will have four shards to store the indexes for strings.

The parameters are selected based on the security parameter and will remain constant once set.

#	Name	SIN Number	Age	Salary	Weight (gr.)
2	Jack	G556F35	48	52k	75938

2) Generating shares: To illustrate the process of generating shares, we will use a toy example. Let's consider the following record that is given to a client for sharing purposes,

Here, *Weight* is the secret data that is required to be kept hidden. The client runs the following algorithm to share this record,

Algorithm 1 Generating DSs' shares.

Require: The record
$X = \{x_1, \ldots, x_n\}$: Private vector.
n: The number of the data servers.
k: Threshold.

1: **Procedure:**
2: Chooses at random a polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree $k-1$ where $f(0) = Weight$.
3: Generates n shares $share_i = f(x_i)$, $i \in \{1, 2, \ldots, n\}$.

Upon generating the records, the client generates a random index $index_{rec}$ and sends the following information to the i^{th} DS.

#	Name	SIN Number	Age	Salary	share	index
2	Jack	G556F35	48	52k	$share_i$	$index_{rec}$

3) Adding indexes to IS: Prior to explaining the index addition process, we will introduce a data structure utilized by the IS for index storage. As previously mentioned, the confidential value domain D is divided into r segments $D_1, \ldots, D_r$. Therefore, IS has r shards, each of which holds multiple buckets where the indexes are stored. Each bucket consists of a header and body, as depicted in Fig. 3. The header serves as a label to aid the IS and CLs in managing and accessing the buckets as needed. The body is where the indexes are kept.

When share generation finishes, generated index $index_{rec}$ is stored on IS according to Alg. (2). Briefly, this algorithm traverses all the buckets in the corresponding shard to find the latest one which has empty space, and IS is asked to add the index to this bucket. Thus, if the $Weight \in D_j$, this index is added to j^{th} shard.

It should be noted that the headers of the buckets are produced using a pseudorandom generator (PRG), making them indistinguishable from random sequences. Hence, adversaries cannot uncover any correlation between the generated buckets. This enhances the security of the scheme, which will be discussed in further detail in the following sections.

C. Answering queries

Clients are responsible for the translation of the queries they receive before forwarding them to the DSs and IS. They then respond with the information they have stored in their databases. Finally, the client evaluates the answer and passes it to the user. The proposed scheme guarantees the security

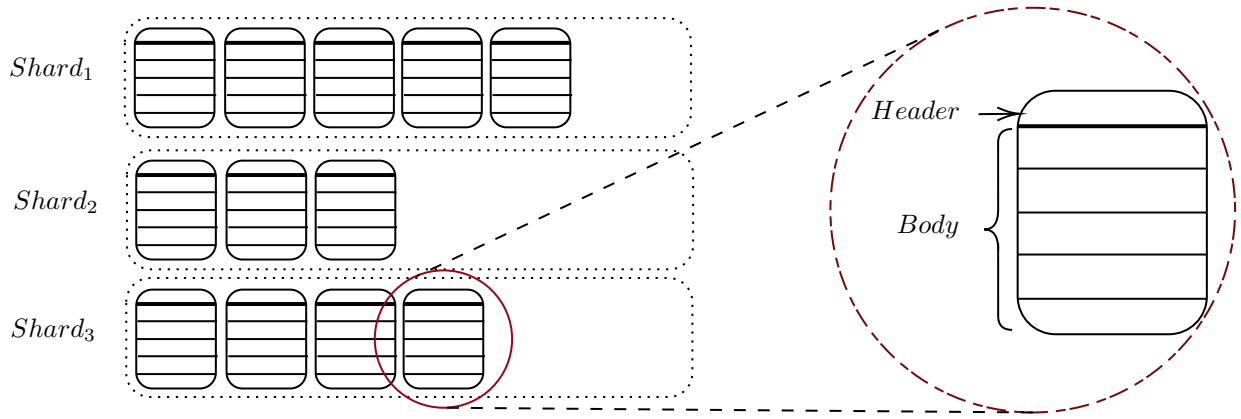

Fig. 3: Sharding data structure to store indexes inside IS. This data structure consists of three *Shard*s. In each shard, corresponding to a partition, indexes are stored in several buckets. Each bucket consists of two parts, *Header* and *Body*. The i^{th} bucket in the j^{th} shard is magnified.

Algorithm 2 Adding an index into index server.

Require: $index_{rec}$: chosen index;
G: a PRG;
t: bucket size;
sk: Secret key;

1: **Procedure:**
2: m=1; $\{Bucket_m$ represents the m^{th} bucket in this shard and its header is $G(sk|j|m).\}$
3: **while** $|Bucket_m| < t$ **do**
4: $m = m + 1$;
5: Ask IS to send $Bucket_m$;
6: **end while**
7: Ask IS to add $index_{rec}$ to $Bucket_m$;

of stored records while responding to queries. This process is secure as it has been demonstrated that even if fewer than k servers were to work together and have access to the IS, they would still not be able to gain any information about the shared records.

The proposed scheme can answer different types of queries. The first type is *Exact match* in which the customer tries to understand how many records satisfy a specific condition. *Range* is another type of query that can be answered. The next type of query that can be handled by the proposed scheme is *Aggregation* queries, such as *Sum*, *Max* etc. Finally, *Update* and *Projection* are other kinds of queries that can be easily performed by the scheme.

To make the answering process more comprehensible, we will demonstrate how the scheme operates when the input data is represented by the table (I). For the sake of simplicity, we will consider a scenario with only two DSs, S_1 and S_2, and a single client and *weight* will be the attribute that should be hidden.

a) Exact match: The objective of this query is to determine the number of records that meet a specific criterion. Assume that a user tries to understand the number of individuals with a weight of 76 kilograms. Hence, the query is submitted to the client as follows,

"SELECT * FROM $Table$ WHERE $Wei. = 67k$"

For simplicity, $Wei.$ is used instead of $Weight$. Upon receiving the request, CL should identify the shard that holds the indexes of records whose weight is 67 kilograms. After identifying the shard, CL requests the IS to provide all the indexes found within it. Subsequently, CL requests the servers to send the corresponding shares for the provided indexes, enabling the recovery of all secrets. Finally, CL will deliver the count of individuals weighing 67 kilograms to the user.

b) Range query: The next commonly used query is called *Range*. In cases where the data is discrete and the range of the data is confined, the *Range* query can be easily transformed into a series of *Exact Match* queries. However, this approach is significantly inefficient when the shared attribute is scattered, especially when it is a real number. Instead of this approach adopted by current schemes, we use another approach that reduces running time. Suppose a client requests the answer to the following query,

"SELECT * FROM $Table$ WHERE $67k \leq Wei. \leq 88k$"

User sends this query to CL. The client determines shards ($pShards$) which can possibly contain the indexes corresponding to the values in $(67k, 88k)$.

$$pShards = \{Shard_\alpha | D_\alpha \cap (67k, 88k) \neq \varnothing\}$$

The shards in $pShards$ can be divided into two groups,

- $pShards_{in} = \{shard_\alpha \in pShards | D_\alpha \subset (67k, 88k)\}$
- $pShards_{bound} = \{shard_\alpha \in pShards | D_\alpha \cap \{67k, 88k\} \neq \varnothing\}$

Obviously, we have,

$$pShards = pShards_{in} \cup pShards_{bound}$$

In addition, all records whose indexes are in the shards in $pShards_{in}$ satisfy the condition of the query. Therefore, the

client does not need to recover their value and only can count them and save it in a variable $result$. In contrast, records whose indexes are in the shards in $pShards_{bound}$ should be recovered because some of them might not meet the condition stated in the query. Therefore, first, they should be recovered, second, the number of them that satisfy the query's condition will be added to $result$ to achieve the final answer. Then, the $result$ is passed to the customer.

c) Aggregation query: Another type of query which can be answered by the scheme is aggregation query. *MIN*, *MAX* and *SUM* are three queries that lie in this type and can be answered by the proposed scheme. A *MAX* query is submitted by a user has the following form,

"SELECT Name FROM $Table$ WHERE WEIGHT = MAX(wei.)"

MAX and MIN: In order to answer this query as described in Alg. (3), CL asks for all data in the last non-empty shard that contains the maximum value. After recovering all the records whose index in this shard, client can identify the maximum value and send it to the user. The same routine can be applied for *MIN* query. The only difference is that the client asks for the all records whose indexes are in the first shard.

Algorithm 3 *Max* Query

1: **Procedure:** // Executed by the client.
2: Recover all records in the last Shard.
3: Find the maximum value.
4: Return the found maximum to the customer.

SUM: Shamir secret sharing is additive. In other words, DSs can add their shares to obtain the share of the summation of values and send these shares to the client to recover the summation of the values.

d) Update query: In the proposed scheme, the dynamic nature of databases is taken into account, allowing for the replacement, deletion, or modification of all or parts of the database. This flexibility is essential in ensuring the effectiveness and efficiency of the database system. In the proposed scheme, these operations can be performed.

Delete: First operation is deletion. Suppose the data owner tries to remove some part of the stored data. (S)he asks a client to remove that part and then client does the following steps to remove the determined parts,

- Find the shards that contain indexes of the part data owner wants to remove.
- Recover the records whose indexes are in these shards.
- Withdraw records that data owners try to remove.
- Share the remained records.

Modify: In order to modify the some records, a client recovers them, and after modifying, records are shared again.

Insert: Insertion is the easiest operation as it does not need recovering. According to Sec. IV-B new records are be added.

e) Selection query: In this kind of query, some columns and some conditions are given and users want to know which records satisfy the determined conditions. Following is an example of these queries,

"SELECT Name FROM $Table$
WHERE $32 \leq AGE \leq 52$ AND $Wei. = 77k$"

In order to answer the above query, a client follows the following steps,

- Finds the possible shards that contain records satisfying the conditions.
- Recovers corresponding records.
- Finds answers.
- Sends the results to the user.

V. Implementation

Multiple implementations were conducted to assess the efficiency of the scheme. The scheme has two stages, namely sharing and answering the queries.

A. Answering the queries

The duration of the process of answering the queries can be influenced by various factors,

- Database's Size.
- PCs' configurations.
- Buckets' size.
- Query's type.

To simplify the analysis, only two data servers were considered. Additionally, only queries that have longer response times were selected, while others were excluded. To minimize the impact of noise, each part of the implementation was repeated five times, and the average running time was computed. The findings of the analysis are presented in Fig.4.

Database's Size: It is observable that the running time increases as the size of the database increases. We can make the assertion that the running time will increase linearly with the number of records, and it will not surpass one second even for a database size of seven million records.

PCs' Configurations: Implementations have been executed over two different PCs, PC_1 and PC_2. PC_1 has Intel Core i5-7400 3 GHz CPU and 4 GB of RAM and PC_2 has Intel Core i7-7700 GHz CPU and 16 GB of RAM. The implementation is done via Python 3.7 on Windows 10 for both PCs. Overall, the findings indicate that the proposed scheme is highly scalable, as queries can be answered in under a second, even on a home computer.

Buckets' Size: Increasing the size of the buckets (t) has a minor impact on the running time. As can be observed in the results, increasing the size of the buckets slightly reduces the running time. This is mainly because a larger t reduces the number of possible buckets that need to be searched for a given query, as the number of indexes stored per bucket increases. As a result, decreasing the size of the buckets can increase communication costs, which in turn increases the running time. However, the change in running time resulting from adjusting the bucket size is relatively insignificant and can be disregarded.

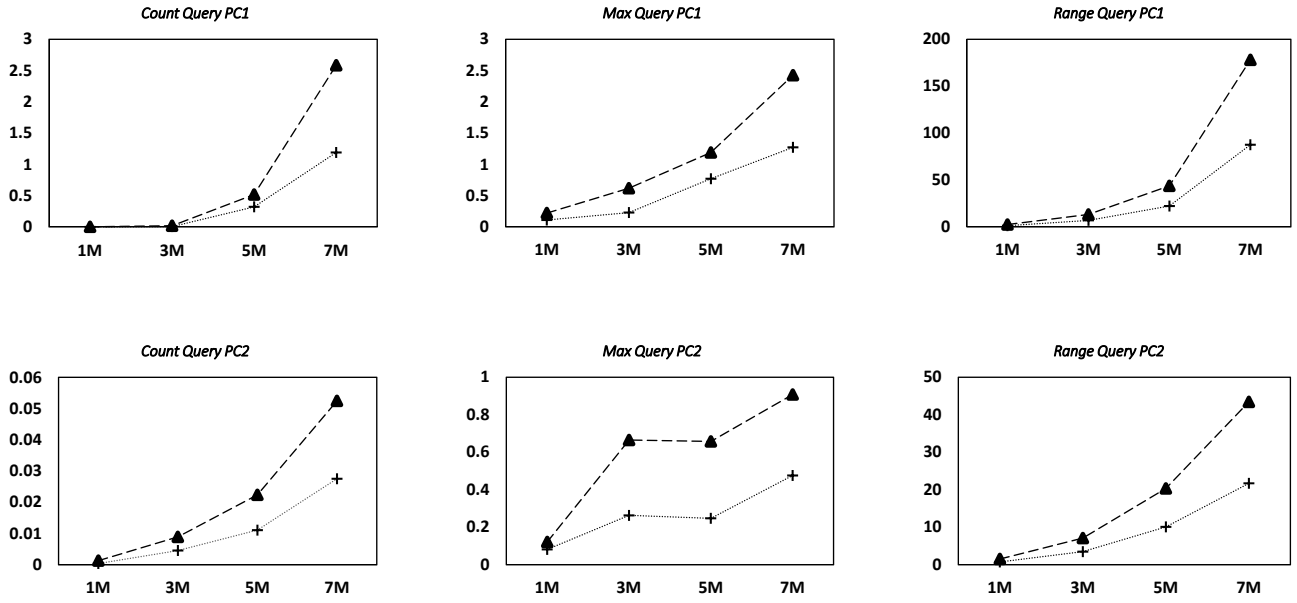

Fig. 4: Implementation of the scheme over two systems with different configurations ($PC1$: Intel Core i5-7400 3 GHz CPU and 4 GB of RAM, $PC2$: Intel Core i7-7700 GHz CPU and 16 GB of RAM). Y-Axis represents running time in milliseconds and X-Axis is the number of shared data among data servers. In these figures, ▲ and $+$ signs represent different size of the buckets, 10 and 20 respectively.

Query's Type: According to the results, *Range* query stays the highest among all possible queries. This roots in the fact that during the process of *Range* query, several shards should be assessed to reach the final answer and the client might have to recover irrelevant records, incurring higher communication and computation throughput. However, it can be clearly seen that the presented scheme is quite scalable and handle range queries securely.

B. Sharing process

The investigation will encompass an analysis of the sharing process in terms of the required time. The results obtained indicate that a typical computer is capable of sharing seven million records under one minute, given a scheme that utilizes two data servers. It should be noted that the sharing process is a one-time and offline operation that has negligible effects on the scheme's scalability.

C. Comprehension

To establish a benchmark for the scheme, a comprehensive comparison has been conducted based on the obtained results. The results are presented in a detailed table, outlining a comprehensive analysis of both the proposed scheme and current IDSSOs, with particular attention paid to factors of security and efficiency.

Schemes	Queries					False Hits	Attack resistance	Fake records	Scattered Values
	Range	Max	Count	Selection	Sum				
Hadavi [8]	7.9	130	NA	NA	NA	×	×	✓	×
Emekci [14]	3.5	100	NA	NA	NA	✓	×	×	×
Hacigumucs [15]	1.8	NA	NA	NA	NA	✓	✓	×	×
Bahrami [10]	0.139	138	1	13	0.115	×	✓	×	×
Proposed scheme	170	2.5	2.7	320	23	✓	✓	×	✓

TABLE II: Comparison between the proposed scheme and the current schemes. The running times are in m.s. All queries are calculated over a database with 5k Records.

The range of queries supported by the proposed scheme is wider than that of the current schemes, as demonstrated in Table II. In addition, the proposed scheme exhibits efficient and secure handling of scattered data, in contrast to other schemes that cannot provide a scalable solution for dispersed values. Ghasemi [9] presented an attack technique that enables an unauthorized individual to retrieve sensitive information from shares stored in DSs. The impact of this attack on various schemes and the presented scheme has been documented in the "Attack resistance" column of a table. According to the table, the proposed scheme resists this attack. Some schemes, add fake records to increase the security of their scheme. In contrast, in the presented scheme, there is no need for adding fake records and it has been proved that it is secure with respect to the security definition presented in [10]. Note that the in the proposed scheme, there are false hits in the process of queries, affecting running time. However, implementations demonstrate that this does not have a noticeable impact on the running time and the scheme still is able to handle queries efficiently.

VI. Conclusion

The current ISSDOs schemes are inefficient in handling scattered values due to their data structure. This paper proposes a new data structure that significantly reduces communication costs compared to current schemes when dealing with scattered values. Unlike the current schemes, this data structure utilizes shards to store indexes of values, resulting in a reduction in the number of buckets and communication costs. However, the use of sharding may result in false hits, which can increase the running time. Nonetheless, several implementations have demonstrated that false hits have negligible effects on running time, and all queries can be answered in a few milliseconds. Furthermore, the proposed scheme has been proven to be $k-1$-secure in terms of security. Therefore, any coalition consisting of no more than $k-1$ servers and the index server will be unable to extract meaningful information about shared data.

References

[1] "Aaron souppouris. linkedin investigating reports that 6.46 million hashed passwords have leaked online." http://www.theverge.com/2012/6/6/3067523/linkedin-password-leak-online., 2012.

[2] "Cambridge analytica and facebook: The scandal and the fallout so far." https://www.nytimes.com/2018/04/04/us/politics/cambridge-analytica-scandal-fallout.html, 2012.

[3] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 965–976.

[4] M. Bellare and P. Rogaway, "Optimal asymmetric encryption," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1994, pp. 92–111.

[5] G. J. Simmons, "Symmetric and asymmetric encryption," in *Secure Communications and Asymmetric Cryptosystems*. Routledge, 2019, pp. 241–298.

[6] Z. Tang, "Secret sharing-based iot text data outsourcing: A secure and efficient scheme," *IEEE Access*, vol. 9, pp. 76 908–76 920, 2021.

[7] J. L. Dautrich and C. V. Ravishankar, "Security limitations of using secret sharing for data outsourcing," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2012, pp. 145–160.

[8] M. A. Hadavi, R. Jalili, E. Damiani, and S. Cimato, "Security and searchability in secret sharing-based data outsourcing," *International Journal of Information Security*, vol. 14, no. 6, pp. 513–529, 2015.

[9] R. Ghasemi, "Resolving a common vulnerability in secret sharing scheme–based data outsourcing schemes," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 2, p. e5363, 2020.

[10] S. Bahrami and R. Ghasemi, "A new secure and searchable data outsourcing leveraging a bucket-chain index tree," *Journal of Information Security and Applications*, vol. 67, p. 103206, 2022.

[11] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[12] G. R. Blakley, "Safeguarding cryptographic keys," in *Managing Requirements Knowledge, International Workshop on*. IEEE Computer Society, 1979, pp. 313–313.

[13] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*, ser. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2014. [Online]. Available: https://books.google.com/books?id=OWZYBQAAQBAJ

[14] F. Emekci, A. Methwally, D. Agrawal, and A. El Abbadi, "Dividing secrets to secure data outsourcing," *Information Sciences*, vol. 263, pp. 198–210, 2014.

[15] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing sql over encrypted data in the database-service-provider model," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 2002, pp. 216–227.

Appendix

A detailed formal proof will be added to the paper in the final version later.