Received March 13, 2021, accepted March 16, 2021, date of publication April 23, 2021, date of current version June 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3075282

Secret Sharing-Based IoT Text Data Outsourcing: A Secure and Efficient Scheme

ZHAOHUI TANG, (Member, IEEE)
University of Southern Queensland, Toowoomba, QLD 4350, Australia
e-mail: zhaohui.tang@usq.edu.au

ABSTRACT Secret Sharing has been recently used as an alternative approach to solve privacy-preserving issues in cloud-based data outsourcing, for overcoming the challenges faced when encryption-based methods are adopted. In this work we revisit secret sharing-based text data outsourcing schemes and focus on their applications into an Internet-of-Things (IoT) system with resource constrained IoT devices as clients. We propose a new method which is secure against common attacks to secret sharing-based text data outsourcing schemes. Compared with the existing works under the same assumption that the cloud servers are possibly colluded, our scheme is more efficient and supports multiplication based operations.

INDEX TERMS Data outsourcing, privacy-preserving, secret sharing, Internet-of-Things (IoT).

I. INTRODUCTION

A. PROBLEM STATEMENT

We consider a typical data outsourcing system as shown in FIGURE 1 below, where user data are uploaded to a group of cloud servers via Internet-of-Things (IoT) devices, for recording, sharing and research purposes [1]. Leveraging cloud services not only provides an increased storage capacity, but also offers an enhanced processing power and thus computational capabilities for the underlying IoT system. Apart from storage services, we consider two minimum functionalities that are required to perform at the cloud servers: SQL queries [2], and computations such as machine learning as a service [11], [12].

Without loss of generality, we consider two cloud servers used to host customers' data: Server 1, Server 2. And we assume these two servers have a priori knowledge of the data, including the distribution or frequency of their values. Furthermore, we assume the servers can communicate and exchange with each other the information they possess.

There are well-known security breaches to data outsourcing systems on various security aspects including privacy-preserving which is of particular interest to this paper.

One approach of solving data privacy-preserving issues in data outsourcing is through traditional encryption [2]–[6] which is computationally expensive and faces a big challenge if further complex processing is required because decryption is needed prior to further processing. Homomorphic encryption [9], [10] has therefore been introduced to enable intermediaries and end users to further process encrypted data without decrypting them. Nevertheless, the practicality of homomorphic encryption is still arguable because an efficient homomorphic encryption does not always exist for all applications.

An alternative approach, secret sharing-based data outsourcing, has been recently proposed to overcome the aforementioned challenges faced by encryption-based schemes.

Under the secret sharing-based approach, each cloud server is given a random share of the secret (i.e., a data owner's confidential information). A group of servers are able to reconstruct the secret if and only if they possess a sufficient amount of shares (instead of privately-owned keys in encryption-based schemes). Further processing is possible based on the shares without the need of decryption.

Among the existing secret sharing schemes, Shamir's (k, N)-threshold secret sharing scheme (Section III-A1) has been widely used to preserve the privacy of outsourced text data for various organizations including hospitals and medical research institutes [15], smart grid companies [16], intelligent transportation solution providers [17] and others [1], [18], [19].

Nevertheless, it is noteworthy to mention that Shamir's (k, N)-threshold secret sharing scheme requires a share size of N times the size of the secret. Different from text data outsourcing, applications with massive amounts of images have faced memory space issues with Shamir's scecret sharing scheme due to the N times of share size (of the images) required. For efficiency consideration, another secret sharing scheme, called ramp secret sharing (or multi-secret sharing) has therefore been utilized to replace Shamir's secret sharing scheme for outsourcing image datasets [20], [21].

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Zakarya.

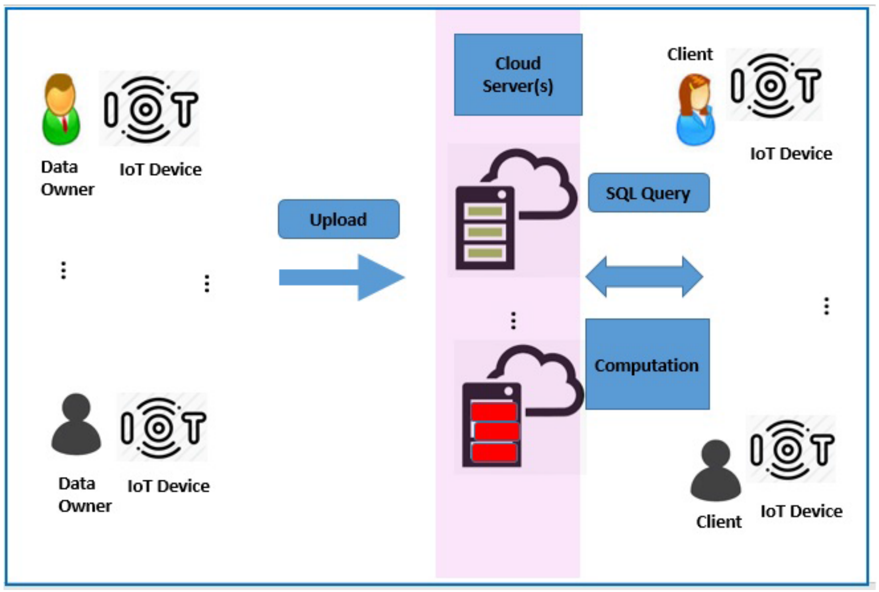

FIGURE 1. An IoT data outsourcing system.

More precisely, as compared with Shamir's (k, N)-threshold secret sharing scheme, an (ℓ, k, N) ramp secret sharing scheme reduces the share size by $\frac{1}{\ell}$ times at the sacrifice of security level though (which can be improved by incorporating an additional security solution such as permutation, stenography, matrix projection, etc..) [21].

In this work we concentrate on text data outsourcing which can be efficiently protected by Shamir's secre sharing scheme without the need of any additional security mechanism. As such, for convenience of discussion in this paper, we refer to secret sharing as Shamir's secret sharing scheme and data type as text format, unless otherwise stated.

As observed in previous works [22]–[24], potentially there are two types of attacks against secret sharing-based outsourcing schemes.

The first type of attacks, called *inference attacks*, will be explained in detail in Section III-C1. An inference attack was discovered by Dautrich and Ravishankar [22] who exploited an inherent vulnerability in Shamir's secret sharing scheme. Hadavi *et al.* [23] mitigated inference attacks through a method called *secure partitioning*. However, as admitted by the authors in the literature [23], their approach incurs a high storage overhead at the client side who is required to securely store the mapping function that is generated by a data owner and must be securely kept by the client for recovering the original secret. The security of their secure partitioning approach relies on the assumption that the mapping function is out of the access of any attacker including the outsourced cloud servers.

The second type of attacks, termed as *gcd-based attacks*, will be described in Section III-C2. A gcd-based attack has been discovered by Ghasemi in the literature [24] where a countermeasure was also proposed by inserting 10% extra records as fake records in order to protect the real records. We refer to this approach as *fake record approach* in the later sections.

Apart from the security concerns, there is another issue that should have been investigated in the secret sharing based publications mentioned above. While these literatures support additions and addition related operations thanks to the additive homomorphism property in their proposed schemes, none of them has discussed how to perform multiplication operations in their schemes which lack the multiplicative homomorphism property.

There is another line of related research which built on secure multi-party computation with an assumption that the servers are not allowed to communicate with each other [11], [12]. Although this series of publications have supported multiplication operations, they are vulnerable to both inference attacks and gcd-based attacks mentioned above.

B. MOTIVATION

This paper worked on secret sharing-based approach, with an assumption that the various cloud servers can collude to poll their shares for retrieving data owners' sensitive inputs.

We aimed to devise a scheme to achieve the following two goals.

- Our first goal was set to overcome the challenges faced when applying existing schemes into a large domain database.
  - In terms of the efficiency against inference attacks, the authors of secure partitioning approach have acknowledged that the client storage size needs to be reduced for a large domain database, due to their demanding size of order $\mathcal{O}(|D_v|)$ (D_v stands for the domain of unique values in the database).
  - For mitigating gcd-based attacks, a minimum amount of 10% fake records are required for the

fake record approach to protect real records, which is impractical for a large domain database.
Motivated by these facts, the first goal of our current work is to propose a scheme which is not only secure against both inference attacks and gcd-based attacks, but also more efficient than [23], [24] especially in terms of the storage overhead at clients (which are resource constrained IoT devices).

- Secondly, as mentioned in Section I-A, none of these literatures has realized multiplication operations, which are essential and commonly used in real-life applications. Motivated by this, we aim to support multiplication based queries and computations in this paper.

It's worth to mention that it's not the purpose of this paper to compare our scheme with the line of secure multiparty computation (SMC)-basaed works which also support multiplication operations. This is due to the different assumptions we made in our work. More specifically, our proposed scheme tolerates possible collusion between honest-but-curious servers, which is prohibited in the line of SMC-based literatures.

C. ORGANIZATION

The paper is organized as follows. We give full details of assumptions and threat modeling in Section II. Preliminaries including secret sharing related concepts and a brief introduction of secret sharing-based data outsourcing schemes can be found in Section III. Our detailed solution is presented in Section IV followed by its security analysis in Section V. The experimental results and comparisons with existing works are shown in Section VI. We conclude the paper in Section VII.

II. ASSUMPTIONS AND THREAT MODELING

A. ENTITIES

We assume there're three types of entities in the data outsourcing system illustrated in FIGURE 1:

- **Data owners** who own the original data.
- **Servers** who receive, store the data from data owners, as well as responding to query and computation requests from clients.
- **Clients** who send query and computation requests.

B. ASSUMPTIONS AND THREAT MODELING

1) ASSUMPTIONS

We made the following assumptions about the data outsourcing system:

- Servers are passive attackers, i.e., they are *honest-but-curious* but not malicious.
- Servers have a prior knowledge of the *statistical* information about the original data, including the distribution of all values of the data.
- Servers can possibly collude by pooling their shares to recover sensitive information belonging to a data owner.

2) THREAT MODELING

We assume data owners and clients are trusted, while servers can be compromised to reconstruct the data owners' confidential information based on what's stored at their premises.

3) NOTATIONS

We use the notation $\{A\}$ frequently in this paper where A is a variable. By the notation of $\{A\}$ we refer to the collection of all possible A values. By a more specific notation of $\{A_i\}$ we refer it to the collection of values A_i for all possible i's.

III. SECRET SHARING-BASED DATA OUTSOURCING

A. INTRODUCTION ON SECRET SHARING

The idea behind secret sharing is to share a secret among multiple participants. Each participant is given one or more shares, and whether a group of participants are able to reconstruct the secret is determined by what shares they possess.

A common type of secret sharing is called threshold secret sharing scheme where participants are able to reconstruct the secret if and only if the number of shares they hold reaches a threshold number. A classical implementation of threshold secret sharing scheme is the (k, N) *Shamir Secret Sharing*.

1) SHAMIR's (k, N)-THRESHOLD SECRET SHARING

Shamir's (k, N)-threshold secret sharing scheme is designed to share a secret, denoted as v, among N participants such that at least k shares are required for successfully reconstructing v.

There are two steps involved in Shamir's (k, N)-threshold secret sharing, which are presented in *Section III-A1.a: Share Generation* and *Section III-A1.b: Secret Reconstruction* respectively.

a: SHARE GENERATION

To share a secret v, the sender chooses a number $p > v$, and randomly selects $k-1$ coefficients $c_1, \cdots, c_{k-1}$ from a finite field $\mathbb{F}_p$. A unique polynomial is therefore formed and can be represented as follows:

$$q(x) = v + \sum_{h=1}^{k-1} c_h x^h \ mod\ p \tag{1}$$

The sender also generates a distribution vector

$$X = (x_1, \cdots, x_N) \tag{2}$$

of distinct elements from $\mathbb{F}_p$.

Finally, for Participant $1 \le i \le N$, the sender computes and sends it a share $y_i = q(x_i)$.

b: SECRET RECONSTRUCTION

A group of k participants are able to reconstruct the original secret v, by utilizing their shares $y_{j,1}, \cdots, y_{j,k}$ and unique elements $x_{j,1}, \cdots, x_{j,k}$ (see Equation (2) above). More specifically, they reconstruct v through the following calculation:

$$v = \sum_{i=1}^{k} y_{j,i} \ell_{j,i}(0) \ mod\ p. \tag{3}$$

The notation $\ell_{j,i}(0)$ here refers to the following Lagrange basis polynomial $\ell_{j,i}(x)$ evaluated at $x = 0$:

$$\ell_{j,i}(x) = \prod_{1 \le m \le k, m \ne i} \frac{x - x_{j,m}}{x_{j,i} - x_{j,m}} \; mod \; p \tag{4}$$

The security of Shamir's (k, N)-threshold secret sharing rests on the fact that at least k points are needed to reconstruct a unique polynomial of degree $k - 1$.

B. INTRODUCTION: SECRET SHARING-BASED DATA OUTSOURCING

The basic idea of secret sharing-based data outsourcing is to treat the multiple servers as participants who receive and store shares that are generated by data owners, through the traditional Sharmir's secret sharing scheme as depicted in Section III-A.

However, there exists a significant difference between a conventional secret sharing scheme and applying it into our data outsourcing setting. The receivers of the shares in our scenario, i.e., the cloud servers are not authorized to gain knowledge of any secret that is originated from a data owner and recoverable at clients. To meet this security goal, we have to prevent the servers from accessing the distribution vector X which is needed for recovering the secret v (see Section III-A1.b above). The vector X should be only accessible to the authorized clients in addition to the respective data owner.

In this paper, for simplicity we assume $k = N = 2$ and thus a Shamir's (2, 2)-threshold secret sharing scheme is adopted. This means an authorized client has to receive the shares from both the two servers in order to obtain an accurate query and computation result. More formally, Polynomial (1) can be formatted as:

$$q(x) = v + c_1 x \; mod \; p.$$

For convenience of discussion, we use the notation c_v to denote the coefficient that is associated with the secret v. As a result the polynomial $q(x)$ can be re-written as:

$$q(x) = v + c_v x \; mod \; p.$$

We assume a data owner outsources his sensitive value v by generating and storing the resultant shares at cloud servers. Furthermore, all subsequent SQL query and computation operations requested by clients on v are performed based on v's shares that are stored at cloud servers.

C. POTENTIAL ATTACKS TO SECRET SHARING-BASED APPROACH

1) INFERENCE ATTACKS

In an inference attack, an attacker conducts statistical analysis by exploiting a prior knowledge about the victim's original data distribution [7], [22], [23].

If there exists an association between the frequency of a secret and the frequency of its shares, it has been demonstrated in [22] that the attacker is able to recover all the remaining secrets provided that k secrets are known beforehand.

Existing works mitigated inference attacks by perturbating the original data distribution in their shares, more specifically, by distributing shares across a well-defined domain such that the association between a secret and its shares does not exist any more. An exact implementations of data perturbation to remove the aforementioned association has been presented in a recent work, i.e., Hadavi *et al.* [23]. In their work, Hadavi *et al.* tackled inference attacks through secure partitioning which divides the domain of shares in such a way that equal secret values are more likely to be mapped onto different shares.

2) GCD-BASED ATTACKS

Gcd-based attacks were first discovered in [24] whereas compromised cloud servers are able to retrieve the distribution vector X, by recovering each of the individual reconstruction element x_i based on the shares they received.

It has been claimed in [24] that, subtracting any two shares that are associated with the same secret will result in a multiple of the reconstruction element x_i. To see this, readers can refer to Equation (5) below, where $(c_{vn} * x_i + v)$ is the share generated from the secret v for an n-th time and $(c_{vm} * x_i + v)$ is the share for an m-th time.

$$(c_{vn} * x_i + v) - (c_{vm} * x_i + v) = (c_{vn} - c_{vm}) * x_i \tag{5}$$

One can easily see that the right hand side of Equation (5) is essentially a multiple of x_i.

The value of x_i has been successfully retrieved in [24] by observing that x_i turns out to be the greatest common divisor (gcd) appearing most frequently in a set of data called *Med*. The dataset *Med* was newly generated by subtracting all two adjacent values from a sorted set of shares that are stored in a server.

With $x_i (1 \le i \le k)$ recovered by the servers, it's easy to check from Equation (3) and Section III-B that these servers are able to reconstruct the secret value v, given the fact that these servers are holding the shares as well, i.e., the $\{y_{j,i}, 1 \le i \le k\}$ in Equation (3).

D. OUR CONTRIBUTION

Our contribution lies in proposing a scheme to achieve the two goals as defined in Section I-B:

- Firstly, we proposed one solution to mitigate inference attacks and gcd-based attacks with reduced costs, as compared with the two existing approaches.
  - As compared with the secure partitioning approach [23], our work is more cost-effective in mitigating inference attacks. More importantly, our scheme thwarted gcd-based attacks while [23] didn't:
    - Similar to [23], we perturbed the original data distribution in their shares and thus thwarted inference attacks. Our solution requires a storage overhead of order $\mathcal{O}(1)$, which is a reduced

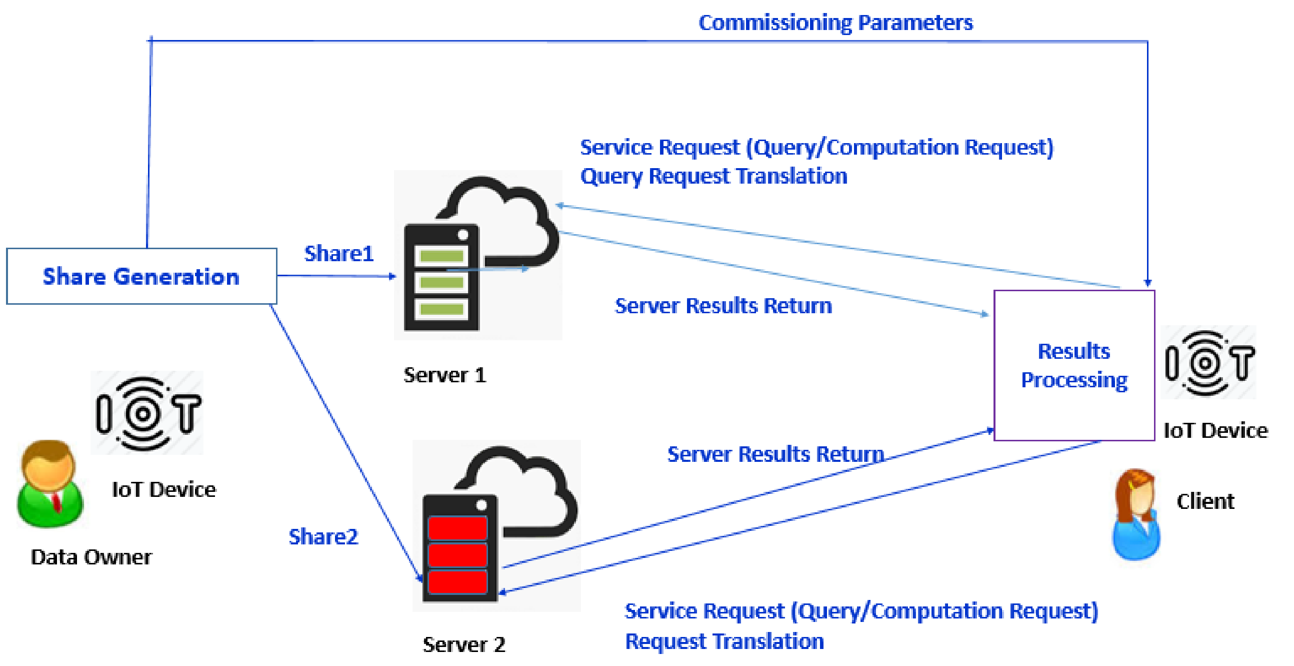

FIGURE 2. **Our proposed secret sharing-based IoT outsourcing scheme.**

cost as compared with the overhead of order $\mathcal{O}(|D_v|)$ as required in the secure partitioninging approach.

In other words, we only need a constant size of storage overhead while the secure partitioning approach requires an extra storage which grows linearly in the domain size of secret values (readers can recall that D_v is the domain of unique values originated from data owners.).

* When compared with the fake record approach [24] which tackled gcd-based attacks by introducing an additional 10% overhead, our scheme is more efficient thanks to the constant size storage overhead achieved.

Our idea of mitigating gcd-based attacks is to mask the original secret v with a random variable prior to share generation. Furthermore, a newly generated random variable was utilized when an equal value was outsourced for another time. That is, we used $r_{v_m} \neq r_{v_n}$ for the m-th and n-th times when v was outsourced ($m \neq n$). With our design, as can be seen below, subtracting two adjacent values will generate some random value rather than a multiple of x_i:

$$c_v x_i + v + r_{v_m} - (v_v x_i + v + r_{v_n}) = r_{v_m} - r_{v_n}.$$

This thwarts gcd-based attacks.

Our first contribution, i.e., reducing storage overhead without sacrificing security, helps to maximize the benefits that cloud-based technologies can offer to a resource-constrained IoT system.

- Our second contribution is that we supported multiplication operations on outsourced data, which has never been considered in any of the existing secret sharing-based outsourcing schemes where servers are allowed to collude.

IV. OUR APPROACH

A. ARCHITECTURE

As mentioned in Section III-B, we assume there are two servers: Server 1 and Server 2 in the system, which can be easily extended to a scenario of more than two servers.

As can be seen from FIGURE 2, our scheme consists of five algorithms: **Share Generation, Service Request, Query Request Translation, Server Results Return, Results Processing**.

The algorithm **Service Request** allows a client to send query and computation requests without any security consideration, which is straightforward and thus omitted here. As follows we describe each of the remaining four algorithms.

1) GENERAL IDEA

Before describing our scheme and algorithms in great detail, we show a general idea on how to accomplish the two goals listed in Section I-B:

- Similar to the secure partitioning approach which successfully tackled inference attacks, we divide the value domain D_v into segments by following Algorithm 2, which results in a random variable domain R_v for each value v. The domain R_v will be used later on when a data owner picks a coefficient (c_v in Equation (6) below) or random variable (r_{v_m} in Equation (6)) for the value v.

For mitigating inference attacks, it is critical to generate differen shares when an equal value is outsourced for multiple times. Distinct from the secure partitioning approach where a different coefficient is used, we use the same coefficient (c_v in Equation (6)) but introduce a mask (r_{v_m} in Equation (6)) prior to sharing the value v. Furthermore, a new mask is used whenever the same value v is outsourced for another time, that is, we use $r_{v_n} \neq r_{v_m}$ when the value v is shared for an n-th time.
Equation (6) below can be obtained from Algorithm 1 which generates shares for each of the i-th server when the value v is outsourced for the m-th time.
The significance of our approach of perturbating the secret share distribution is twofold: (1) our method achieves the same security level against inference attacks without asking a client to store mapping functions as the secure partitioning approach, and (2) our scheme is secure against gcd-based attacks by avoiding resulting in a multiple of x_i when subtracting two shares associated with v.
The only storage overhead required by our approach is a pair of vectors that are needed for reconstructing the mask, which is of constant size of $2\log(p)$ bits when 2 servers are utilized. For a general scenario with N servers, our required storage overhead is $2N\log(p)$ bits, which is constant and independent of the domain of values or the number of records in the database.

$$Share_i(v + r_{v_m}) = c_v * x_i + v + r_{v_m} \tag{6}$$

- Considering the adopted Shamir's (k, N)-threshold secret sharing scheme is additive homomorphic, all addition related functions can be realized by asking servers to locally add all the shares stored. However, due to the lack of multiplicative homomorphism property, it is challenging to accomplish a multiplication operation by this scheme (to be discussed in Section IV-E0.c).

B. SHARE GENERATION

1) SHARE GENERATION

Every time a data owner intends to outsource a value v, there are two steps and two algorithms involved for generating and distributing shares to the servers:

- Step 1: *Random Variable Domain Generation*. For each value v the function *RVDomainGeneration* shown in Algorithm 2 returns v's random variable domain (denoted as R_v) based on its frequency (denoted as freq_v). R_v is the domain of random values, from where a data owner can choose coefficients and random values for generating v's shares, as shown in Step 2 below.
- Step 2: *Share Generation*, as shown in Algorithm 1. For the value v, a data owner randomly chooses $c_v \in R_v$ as the coefficient associated with v. In addition to c_v, another random value r_{v_m} is chosen from R_v when the m-th time v is outsourced. Together with the secret element x_i, we generate the share for Server i as: $share(v)_i = c_v x_i + v + r_{v_m}$ where x_i is from the distribution vector (see Equation (2)).

C. SQL QUERY REQUEST TRANSLATION

It's important to recall that the data stored in the servers are shares of the original values while conventional SQL queries are meant for the original values. Therefore, the client is required to translate the original SQL queries into new queries which are meant for the shares (rather than the original values).

We provide the SQL query request algorithm below, followed by examples of translating two simple types of SQL query requests: equality query and range query. Other types of SQL query requests can be translated in a similar way or derived from our translation algorithm, and thus omitted in this paper.

Algorithm 1 ShareGeneration($\{\text{Server}_i\}$, $\{v\}$, $\{\text{freq}_v\}$,$\mathbb{F}_p$, $\{\text{x}_i\}$,$\{\text{x}_{ri}\}$)

```
$R_v$ =RVDomainGeneration($\{v\}$, $\{\text{freq}_v\}$, $\mathbb{F}_p$)

**for all** $v$ **do**
    Randomly choose $c_v \in R_v$.

    Randomly choose $r_v \in R_v$, with a guarantee that $r_v$ is
    newly generated every time an equal $v$ is outsourced.

    $\{Share_i(v+r_v) = c_v x_i + v + r_v\}$ {Comments: Computing
    shares for servers on $v + r_v$.}

    $\{Share_i(r_v) = c_v x_{ri} + r_v\}$ {Comments: Computing
    shares for servers on $r_v$.}

**end for**

**return** $\{Share_i(v + r_v)\}$ and $\{Share_i(r_v)\}$
```

1) EXAMPLE (Equality Query)

As follows is one example on equality query translation.

Example 1: We assume this is the original SQL query intended for Server i:

$$\text{Select } * \text{ from Table1 where age } = 30.$$

The client has to translate this query into:

$$Min(R_{30})(x_i + 1) + 30 \;\; mod\ p \leq share(S_i) \leq Max(R_{30})(x_i + 1) + 30 \;\; mod\ p$$

and sends to Server i.

2) EXAMPLE (Range Query)

Here is one example of how to translate a range query.

Example 2: We assume this is the original SQL query intended for Server i:

$$\text{Select } * \text{ from Table1 where } 20 \leq Age \leq 70.$$

Algorithm 2 RVDomainGeneration($\{v\}$, $\{\text{freq}_v\}$, $\mathbb{F}_p$)

```
Sort $\{v\}$ in an ascending order and form the new set of values $\{v\}$.

Compute $|D_v|$ as the number of unique values in $\{v\}$.
{Comments: $D_v$ refers to the domain of values in the value set $\{v\}$}

Divide $\mathbb{F}_p$ into $|D_v|$ segments, where Segment$_1$ = [ 1, $\lceil p * \text{freq}_1 / \sum_{i=1}^{|D_v|} \text{freq}_i \rceil$ ]; for 2 $\le i \le |D_v|$, Segment$_i$ = [ 1 + $\lceil p * \sum_{j=1}^{i-1} \text{freq}_j / \sum_{i=1}^{|D_v|} \text{freq}_i \rceil$, $\lceil p * \sum_{j=1}^{i} \text{freq}_j / \sum_{i=1}^{|D_v|} \text{freq}_i \rceil$ ].
for all $v$ do
  $R_v$ = Segment$_{\text{ord}(v)}${Comments: $R_v$ stands for the domain of random variables for $v$, while ord($v$) is defined as the order of $v$ in the set $\{v\}$.}
end for

return $\{R_v\}$
```

Algorithm 3 Query Request Translation(Request)

```
if (request == equality query with condition as "where attrName = $v$") then
  return
    $Min(R_v)(x_i + 1) + v \mod p \le share(S_i) \le Max(R_v)(x_i + 1) + v \mod p$
  {Comments: This returns a translated equality query to Server $i$. By "equality query" it means the condition in the query is attrName = $v$ where attrName is the attribute being queried and $v$ is the value.}

else if (request== range query with condition as "where $v_l \le$ attrName $\le v_h$") then
  return
    $Min(R_{v_l})(x_i + 1) + v_l \mod p \le share(S_i) \le Max(R_{v_h})(x_i + 1) + v_h \mod p$
  {Comments: This returns a translated range query to Server $i$. By "range query" it means the condition in the query is $v_l \le$ attrName $\le v_h$ where attrName is the attribute being queried. $[v_l, v_h]$ is the range of values under search.}
end if
```

The client has to translate this query into:

$$Min(R_{20})(x_i + 1) + 20 \mod p \le share(S_i) \le Max(R_{70})(x_i + 1) + 70 \mod p$$

and sends to Server i.

D. SERVER RESULTS RETURN

As shown in Algorithm 4 below, the returned results are aggregated shares if the request is an aggregate SQL query or computation, otherwise individual shares of each value.

The parameter *rqType* stands for the request type, while *agrtFunc* refers to the aggregate function provided by the client.

Algorithm 4 Server Results Return ($\{Share_i(v + r_v)\}$, $\{Share_i(u+r_u)\}$, $\{Share_i(r_v)\}$, $\{Share_i(r_u)\}$, $rqType$, $agrtFunc$)

```
if ($rqType$ == $aggregate$) then
  return { $agrtFunc$ $\{Share_i(v + r_v), Share_i(u + r_u)\}$, $agrtFunc(\{Share_i(r_v)\}, \{Share_i(r_u)\})$ }

else
  return { $\{Share_i(v + r_v)\}$, $\{Share_i(u + r_u)\}$, $\{Share_i(r_v)\}$, $\{Share_i(r_u)\}$ }

end if
```

E. RESULTS PROCESSING

Algorithm 5 below demonstrates how the results are processed at a client.

The parameter $\{Share_i(v+r_v)\}$ stands for the collection of shares from all servers. Each share, denoted as $Share_i(v+r_v)$ was received by Server i ($1 \le i \le 2$) on the secret $v + r_v$. The parameter $\{\ell_i(0)\}$ is the collection of values held by the client. Each value, denoted as $\ell_i(0)$, is the following Lagrange basis polynomial $\ell_i(x)$ evaluated at $x = 0$ for Server i (derived from Equation (4) in Section III-A1.b):

$$\ell_i(x) = \prod_{1 \le m \le 2, m \ne i} \frac{x - x_m}{x_i - x_m} \mod p \tag{7}$$

Similarly, $\{Share_i(r_v)\}$ is the collection of shares received by Server $i(1 \le i \le 2)$ for the secret r_v. The parameter $\{\ell_{r_i}(0)\}$ refers to the set of values possessed by the client. Each value, denoted as $\ell_{r_i}(0)$, is an evaluation of Polynomial (8) at $x = 0$:

$$\ell_{r_i}(x) = \prod_{1 \le m \le 2, m \ne i} \frac{x - x_{r_m}}{x_{r_i} - x_{r_m}} \mod p \tag{8}$$

a: AGGREGATE REQUEST

Our scheme supports aggregate requests, which can be implemented by specifying the parameter *agrtFun* in Algorithm 7 below.

In this paper we focus on basic aggregate functions which can be reduced to additions and multiplications, and more complex aggregate functions can be discussed in our future work.

While Algorithm 7 takes in a general *agrtFun* as input, we provide more details on addition and multiplication implementations in Section IV-E0.b and Section IV-E0.c respectively.

Algorithm 5 Results Processing ($\{Share_i(v + r_v)\}$, $\{Share_i(u+r_u)\}$, $\{Share_i(r_v)\}$, $\{Share_i(r_u)\}$, $\{\ell_i(0)\}$, $\{\ell_{r_i}(0)\}$, $rqType$, $agrtFunc$)

```
if ($rqType == aggregate$) then
    ResultsProcess4Aggregate($\{\{Share_i(v + r_v) + Share_i(u + r_u)\}, \{Share_i(r_v) + Share_i(r_u)\}\}, \{\ell_i(0)\}, \{\ell_{r_i}(0)\}, agrtFun$)

else if ($valRequested == v$) then
    ResultsProcess4nonAgg($\{\{Share_i(v + r_v)\}, \{Share_i(r_v)\}\}, \{\ell_i(0)\}, \{\ell_{r_i}(0)\}$)
    {Comments: $valRequested$ stands for the value that is requested by the client}

else if ($valRequested == u$) then
    ResultsProcess4nonAgg($\{\{Share_i(u + r_u)\}, \{Share_i(r_u)\}\}, \{\ell_i(0)\}, \{\ell_{r_i}(0)\}$)
    {Comments: $valRequested$ stands for the value that is requested by the client}

end if
```

The function *SecretReconstruction* in Algorithm 6 and Algorithm 7 refers to the secret sharing reconstruction functionality as depicted in *Section III-A1.b: Secret Reconstruction*.

Algorithm 6 ResultsProcess4nonAgg($\{\{Share_i(v + r_v)\}, \{Share_i(r_v)\}\}, \{\ell_i(0)\}, \{\ell_{r_i}(0)\}$)

```
$v + r_v = SecretReconstruction(\{Share_i(v + r_v)\}, \{\ell_i(0)\})$

$r_v = SecretReconstruction(\{Share_i(r_v)\}, \{\ell_{r_i}(0)\})$

return $v = (v + r_v) - r_v$
```

Algorithm 7 ResultsProcess4Aggregate($\{\{Share_i(v + r_v) + Share_i(u + r_u)\}, \{Share_i(r_v) + Share_i(r_u)\}\}, \{\ell_i(0)\}, \{\ell_{r_v}(0)\}, agrtFun$)

```
$agrtFunc(v + r_v, u + r_u) = SecretReconstruction(agrtFunc(\{Share_i(v + r_v)\}, \{Share_i(u + r_u)\}), \{\ell_i(0)\})$

$agrtFunc(r_v + r_u) = SecretReconstruction(agrtFunc(\{Share_i(r_v + r_u)\}), \{\ell_{r_i}(0)\})$

return $agrtFunc(v, u) = agrtFunc(v + r_v, u + r_u) - agrtFunc(r_v + r_u)$
```

b: ADDITIONS

A simple example for demonstrating an addition-based query or computation can be the SQL SUM function.

Without loss of generality, we assume a SUM query based on two values v and u is requested by the client who expects a summation of v and u. In this section we discuss how to process a SQL SUM query.

Recall from Algorithm 1 that Server i received the following shares from the data owner ($x_{i,r_v} = x_{i,r_u}$):

- With regard to the value v, Server i received the following two shares:
  - $c_v x_i + v + r_v \ mod\ p$, which is the share corresponding to the message $v + r_v$.
  - $c_v x_{i,r_v} + r_v \ mod\ p$, which is the share corresponding to the message r_v.
- With regard to the value u, Server i received the following two shares:
  - $c_u x_i + u + r_u \ mod\ p$, which is the share corresponding to the message $u + r_u$.
  - $c_u x_{i,r_u} + r_u \ mod\ p$, which is the share corresponding to the message r_u.

Apart from the aforementioned shares that a client can receive from Server i as a response to his SQL SUM query, the client can utilize the following information he received from the data owner:

- $\ell_i(0)$ which can be used for reconstructing the message $v + r_v$ as well as the message $u + r_u$.
- $\ell_{i,r_u}(0) \neq \ell_i(0)$ for reconstructing r_u.
- $\ell_{i,r_v}(0) = \ell_{i,r_u}(0)$ for reconstructing r_v.

With all the information listed above, the summation of v and u can be calculated via the following two steps:

- Step 1: Calculating $r_v + r_u$:
  - After receiving $((c_v x_{i,r_v} + r_v) + (c_u x_{i,r_u} + r_u)) \ mod\ p$ from Server i ($1 \leq i \leq 2$), the client reconstructs $r_v + r_u$ by:

$$r_v + r_u = \sum_{i=1}^{2} \ell_{i,r_v}(0)((c_v x_{i,r_v} + r_v) + (c_u x_{i,r_u} + r_u)) \ mod\ p.$$

  The correctness can be proved by:

$$\begin{aligned} r_v + r_u &= \sum_{i=1}^{2} \ell_{i,r_v}(0)(c_v x_{i,r_v} + r_v) \ mod\ p \\ &\quad + \sum_{i=1}^{2} \ell_{i,r_u}(0)(c_u x_{i,r_u} + r_u) \ mod\ p \\ &= \sum_{i=1}^{2} \ell_{i,r_v}(0)((c_v x_{i,r_v} + r_v) + (c_u x_{i,r_u} + r_u)) \ mod\ p \end{aligned} \tag{9}$$

- Step 2: Calculating the final result $v + u$:
  - After receiving $((c_v x_i + v + r_v) + (c_u x_i + u + r_u)) \ mod\ p$ from Server i ($1 \leq i \leq 2$), the client reconstructs $u + v + r_u + r_v$ by:

$$v + u + r_v + r_u$$

$$= \sum_{i=1}^{2} \ell_i(0)((c_v x_i + v + r_v) + (c_u x_i + u + r_u)) \ mod\ p \tag{10}$$

- Finally the client calculates $v + u$ by (recall $r_v + r_u$ is obtained in Step 1 above):

$$v + u = ((v + u + r_v + r_u) - (r_v + r_u)) \ mod\ p$$

c: MULTIPLICATIONS

Our multiplication implementation was inspired by the following expressions:

$$u = g^{\bar{u}} \tag{11}$$
$$v = g^{\bar{v}}, \tag{12}$$

where g is a primitive element of the finite field $\mathbb{F}_p$.

Based on Equations (11) and (12) above, we can obtain the multiplication of u and v by:

$$uv = g^{\bar{u}} g^{\bar{v}} = g^{\bar{u}+\bar{v}} \tag{13}$$

As a result, we can reduce the complicated multiplication of u and v to an addition of $\bar{u}$ and $\bar{v}$, which can be accomplished by following the descriptions in Section IV-E0.b.

More formally, our multiplication operation involves three steps:

- The data owner chooses a primitive element g of $\mathbb{F}_p$, based on which he calculates $\bar{u}$ for u satisfying $u = g^{\bar{u}}$, and $\bar{v}$ for v satisfying $v = g^{\bar{v}}$.
- The client runs an algorithm *Additions* as depicted in Section IV-E0.b, and calculates the summation of $\bar{u}$ and $\bar{v}$.
- Finally the client calculates the multiplication of u and v as $uv = g^{\bar{u}+\bar{v}}$.

V. SECURITY ANALYSIS

In this section we show that our proposed scheme is secure against both inference attacks and gcd-based attacks.

A. SECURITY ANALYSIS: SECURITY AGAINST INFERENCE ATTACKS

Lemma 1: Our scheme is secure against inference attacks, with the same security level as the secure partitioning approach [23]. However, our scheme requires less storage overhead than the secure partitioning approach.

Proof 1: We start the proof of achieving the same security level as the secure partitioning approach, followed by the proof on storage overhead reduction.

- Proof on security achievement. We use s to represent the share stored at some server, whereas the goal of an inference attack is to retrieve the original value v that corresponds to s, that is, $s = share(v)$. More specifically, the following equation holds for some i and j:

$$s = c_v x_i + v + r_{v_j} \tag{14}$$

We define $Pr(s = share(v))$ as the probability of success in an inference attack, i.e., the probability of Equation (14) holding for some i and j. We prove that

$$Pr(s = share(v)) \approx \frac{1}{|D_v|} \tag{15}$$

where $|D_v|$ is the number of unique values from the data owner. This in turn implies that our scheme attains the same security level as the secure partitioning approach. Our detailed proof is presented below:

$$\begin{aligned} & Pr(s = share(v)) \\ & = Pr(s = c_v x_i + v + r_{v_j}) \times Pr(r_v = r_{v_j}) \\ & = Pr(s = c_v x_i + v + r_{v_j}) \times \frac{1}{|R_v|} \\ & = \sum_{i=1}^{|R_v|} Pr(s = c_{v_m} x_i + r_{v_j} + v | c_v = c_{v_m}) \times \frac{1}{|R_v|} \\ & \approx \sum_{i=1}^{|R_v|} \frac{1}{|D_v|} \times \frac{1}{|R_v|} \\ & = |R_v| \times \frac{1}{|D_v|} \times \frac{1}{|R_v|} \\ & = \frac{1}{|D_v|} \qquad (*) \end{aligned}$$

The equation (*) holds due to Theorem 2 from the publication [23].

- Storage overhead reduction. Storage overhead in the secure partitioning approach is $\mathcal{O}(|D_v|)$. In our scheme, the only extra storage overhead is the cost for storing the values of $\{\ell_i(0), \ell_{r_i}(0)\}_{i=1}^{N}$, which is of $2N \log(p)$ bits and constant size since we assume the number of servers (i.e., N) is fixed. The value of $2N \log(p)$ is independent from the domain size $|D_v|$, which implies that our scheme is more efficient than the secure partitioning approach in terms of storage overhead at the client.

B. SECURITY ANALYSIS: SECURITY AGAINST GCD-BASED ATTACKS

Lemma 2: Our scheme is secure from gcd-based attacks and more efficient than the fake record approach [24] which was the only existing work having addressed gcd-based attacks.

Proof 2:

- Proof on security achievement. The security against gcd-based attacks is proved from the observation that the probability of yielding a multiple of x_i when subtracting any two shares is very low.

  For the same v we choose a different random value r_v when v is outsourced for a different time, that is, $r_{v_m} \neq r_{v_n}$ in Statement (16) below:

$$c_v x_i + v + r_{v_m} - (c_v x_i + v + r_{v_n}) = r_{v_m} - r_{v_n} \neq 0. \tag{16}$$

From the statement above, one can clearly see that the subtraction of two adjacent shares in our scheme turns out to be a subtraction of two distinct random numbers instead of a multiple of x_i (as in existing secret sharing-based schemes).

- Proof on efficiency gain. The fake record approach requires at least 10% extra storage and overhead, which grows linearly with the original database size. The only extra storage overhead required in our scheme is the cost for storing the values of $\left\{\ell_i(0), \ell_{r_i}(0)\right\}_{i=1}^{N}$, which is of $2N \log(p)$ bits and constant size for a fixed N number of servers. The value of $2N \log(p)$ is independent from the original database size, which proves that our scheme is more efficient than the fake record approach.

VI. EVALUATIONS AND COMPARISONS WITH RELATED WORK

A. EVALUATIONS AND COMPARISONS WITH WORKS UNDER A SAME ASSUMPTION (i.e., SERVERS CAN BE COLLUDED)

In this section we evaluate and compare our scheme with the existing works which were proposed based on the same assumption, that is, servers are colluded. More specifically, our scheme is compared with the secure partitioning approach, the fake record approach, as well as the encryption and bucket-based method [25] for a more comprehensive benchmarking.

It's important to point out that the server-side capacity is not of a big concern in general, because of its high storage and processing capacities. Henceforth, the costs required from clients became our focus when evaluating the performance of our proposed scheme. More specifically, we evaluated both the computational and storage costs at the client side, with various functionalities performed including equality queries, additions, and multiplications.

1) THEORETICAL COMPLEXITY ANALYSIS

The theoretical complexity analysis and comparison with existing schemes are presented in TABLE 1 (for computational costs at a client) and TABLE 2 (for storage costs at a client), where n denotes the number of records in the database and $|D_v|$ represents the number of unique values for a given database.

One can see that, as compared with the secure partitioning and fake record approaches, the storage size required by our scheme is significantly reduced when the number of unique values or records increases. Our scheme outperforms the encryption and bucket-based approach in terms of computational cost.

2) EXPERIMENTS

a: TESTING ENVIRONMENT

For an experimental comparison, we have prototyped our scheme where a data owner outsourced his sensitive text data to a server equipped with MS SQL Server 2016. To simulate a general cloud server configuration, we experimented with a configuration of Windows 10, Intel Core i5 1.6GHz processor and 16GB of memory.

TABLE 1. Complexity comparison of client side processing time.

Approach	Complexity
Secure Partitioning Approach	$\mathcal{O}(n)$
Fake Record Approach	$\mathcal{O}(n)$
Encryption and bucket-based approach	$\mathcal{O}(2^n)$
Our Proposed Scheme	$\mathcal{O}(n)$

TABLE 2. Complexity comparison of client side storage.

Approach	Complexity
Secure Partitioning Approach	$\mathcal{O}(\lvert D_v\rvert)$
Fake Record Approach	$\mathcal{O}(n)$
Encryption and bucket-based approach	$\mathcal{O}(1)$
Our Proposed Scheme	$\mathcal{O}(1)$

TABLE 3. Comparisons with existing schemes based on assumptions made (colluding or non-colluding servers).

Approach	Secure against Reference Attacks	Secure against Gcd-based Attacks	Additions Supported	Multiplications Supported	Assumption Made
Secure MPC Approach	No	No	Yes	Yes	Non-colluding Servers
Secure Partitioning Approach	Yes	No	Yes	No	Colluding Servers
Fake Record Approach	Yes	Yes	Yes	No	
Our Proposed Scheme	Yes	Yes	Yes	Yes	

For simulating an IoT device (i.e., a client) in our prototyping, we used a Raspberry Pi device which was equipped with 1.2 GHz 64-bit quad core processor and 1GB RAM.

We downloaded a dataset from IPUMS 2010 ACS data [26], with approximately 1 million tuples in the database containing sensitive attributes including Age, Income etc..

While our testing data were mainly numeric values, our scheme can be easily extended to character values and thus works with all text data.

b: EXPERIMENTAL RESULTS AND DISCUSSIONS

The experimental results of our scheme and comparisons with other approaches are presented in FIGURE 3, FIGURE 4, FIGURE 5 and FIGURE 6 below. A comparison on computational costs can be found in FIGURE 3 and Fig. 4. One can see clearly that the secret sharing-based approach outperforms the encryption-based approach in both SQL equality queries and addition operations. Within the category of secret sharing-based approach, our computational cost is slightly

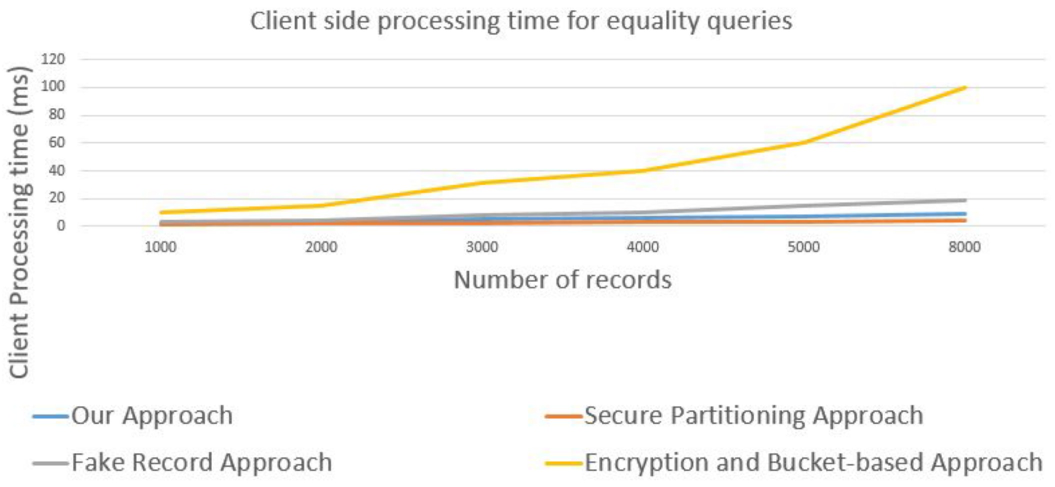

FIGURE 3. Experimental results on equality queries.

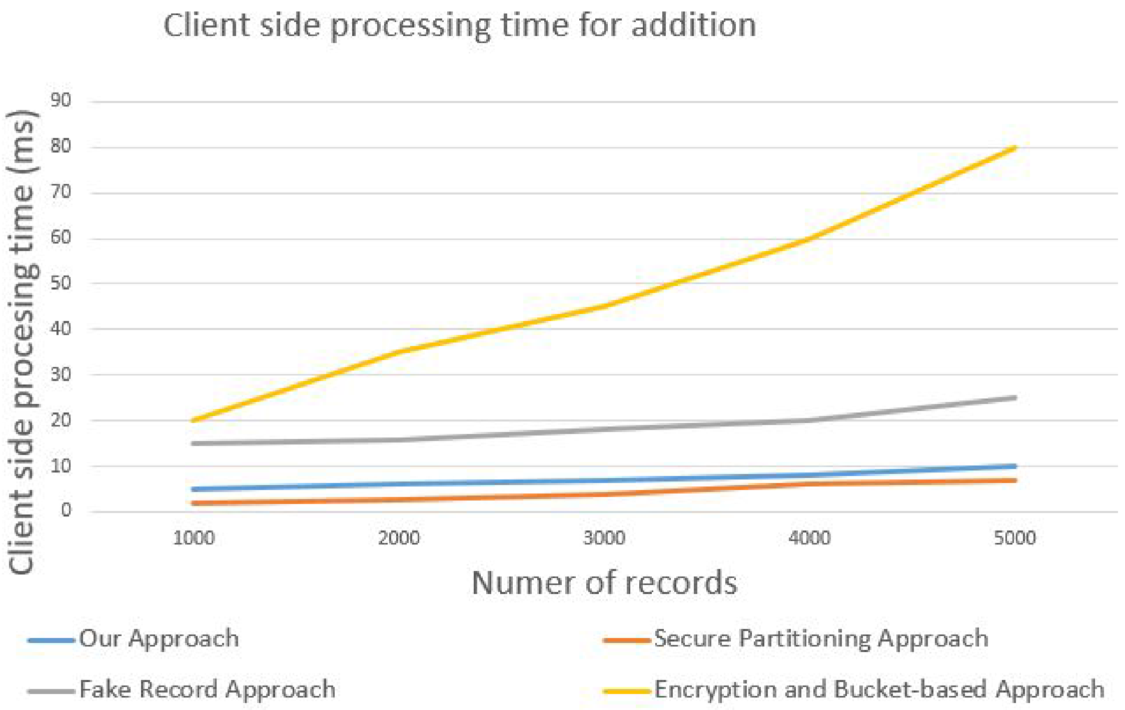

FIGURE 4. Experimental results on additions.

heavier than the secure partitioning approach, but significantly lower than the fake record approach.

The experimental results on storage overhead are presented in FIGURE 5. With two servers experimented, our scheme requires a client to store 4 elements from $\mathbb{F}_p$, i.e., the values of $\{\ell_i(0), \ell_{r_i}(0)\}_{i=1}^{2}$. We chose $p = 2^{64}$ which is sufficient for most of the database sizes. This implies that $4 * 8 = 32$ bytes of storage overhead are required in our scheme, which is slightly bigger than the 16 bytes storage size required in the encryption-based approach using an AES-128 encryption algorithm. However, the advantage of requiring constant size of storage can be seen very clearly in FIGURE 5 that the storage overheads in the secure partitioning approach and fake record approach increased rapidly when the number of unique values (denoted as $|D_v|$) increased in the database.

We also experimented with multiplication operations on two secret values v and u. The processing time required at the client for a multiplication operation is shown in FIGURE 6. It took 120 milliseconds for a client to retrieve the multiplication result of vu based on the shares stored at servers, when a database of 800 records was given. This is considered as acceptable given the low computation capacity of the Raspberry Pi device we used.

From both theoretical analysis and experimental results presented above, one can verify that that our scheme has achieved the goals set in Section I-B.

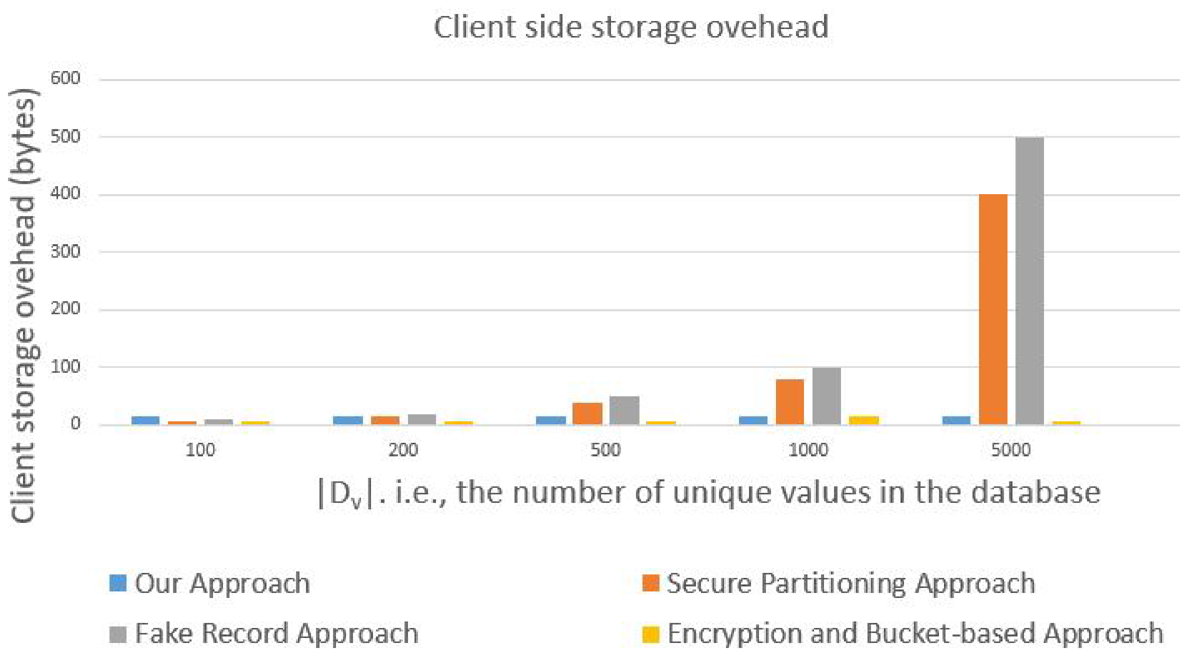

FIGURE 5. Experimental results on client storage overhead.

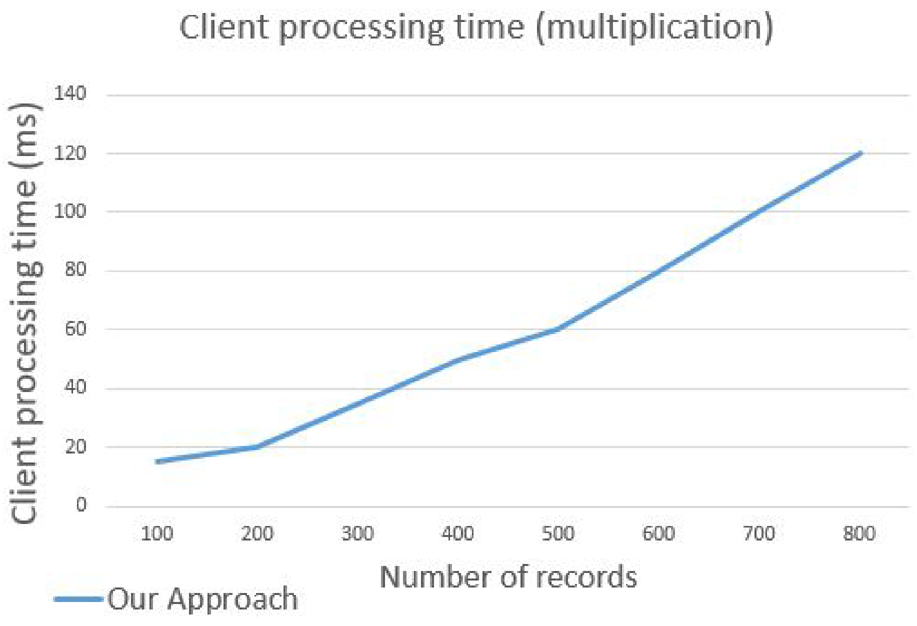

FIGURE 6. Experimental results on multiplications.

B. COMPARISONS WITH WORKS UNDER A DIFFERENT ASSUMPTION (i.e., SERVERS CANNOT BE COLLUDED)

While our solution tolerates possible collusion between servers, the secure multiparty computation (SMC)-based schemes were proposed based on a different assumption. Due to their assumption that servers cannot be concluded, the SMC-based schemes are inherently vulnerable to both inference and gcd-based attacks, as can be seen from TABLE 3 where schemes with different assumptions are compared in terms of security and functionality.

One can easily see from TABLE 3 that our proposed scheme is the only one that has met both security and functionality requirements. More specifically, our scheme is the only solution that not only defends against the two attacks (inference and gcd-based attacks), but also supports the two types of operations (additions and multiplications).

Combing the evaluation and comparison results from Section VI-A and VI-B, we can conclude that our scheme has achieved a best balance between functionality, security and efficiency among all the existing works.

VII. CONCLUSION AND FUTURE WORK

We revisited the secret sharing-based text data outsourcing scheme in the context of an IoT system where clients are resource constrained IoT devices. We designed a new method that is more efficient than existing works, particularly in terms of storage overhead at clients. To our best knowledge, our scheme is the first secret-sharing based scheme that has realized multiplication based queries and computations, under the assumption that the servers are colluded. Our work was further compared with the secure MPC-based approach which were designed for non-colluding servers, and was shown to have achieved a best trade-off between functionality, security and efficiency among existing works.

While our scheme achieves perfect secrecy (i.e., zero information about the secret text is disclosed to any combination of less than k shares), our scheme demands a considerable amount of share size for outsourcing a large amount of image data. Although protecting an individual pixel value of a single image works with the same efficiency as shown in our experimental results, the efficiency of our scheme needs to be improved for protecting image datasets which consist of a massive number of pixel values. Ramp secret sharing-based schemes perform more efficiently than our Shamir secret sharing-based scheme in terms of large image privacy-preservation protection, at an expense of a reduced security level.

Part of our future work is to see whether possible to make multiplication-based computations more efficient, as well as investigate how more complicated computations can be efficiently realized by our scheme. Another future work is to protect the integrity of the outsourced IoT text data since we only discussed the privacy-preservation issues in this paper.

REFERENCES

[1] A. H. M. Aman, E. Yadegaridehkordi, Z. S. Attarbashi, R. Hassan, and Y.-J. Park, "A survey on trend and classification of Internet of Things reviews," *IEEE Access*, vol. 8, pp. 111763–111782, 2020.

[2] D. Agrawal, A. E. Abbadi, F. Emekci, and A. Metwally, "Database management as a service: Challenges and opportunities," presented at the IEEE 25th Int. Conf. Data Eng., Shanghai, China, Mar. 2009.

[3] S. Wang, D. Agrawal, and A. El Abbadi, "A comprehensive framework for secure query processing on relational data in the cloud," presented at the Workshop Secure Data Manage., Seattle, WA, USA, 2011.

[4] E. Damiani, D. S. DeCapitani, S. Paraboschi, and P. Samarati, "Computing range queries on obfuscated data," in *Proc. Inf. Process. Manage. Uncertainty Knowl.-Based Syst.* Washington, DC, USA: IEEE Computer Society, 2004, pp. 1333–1340.

[5] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati, "Efficient and private access to outsourced data," in *Proc. 31st Int. Conf. Distrib. Comput. Syst.* Washington, DC, USA: IEEE Computer Society, Jun. 2011, pp. 710–719.

[6] F. Emekci, A. Methwally, D. Agrawal, and A. E. Abbadi, "Dividing secrets to secure data outsourcing," *Inf. Sci.*, vol. 263, pp. 198–210, Apr. 2014.

[7] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Boca Raton, FL, USA: CRC Press, 2014.

[8] L. Ferretti, M. Colajanni, and M. Marchetti, "Distributed, concurrent, and independent access to encrypted cloud databases," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 2, pp. 437–446, Feb. 2014.

[9] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford Univ., Stanford, CA, USA, 2009.

[10] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu, "Private database queries using somewhat homomorphic encryption," presented at the Int. Conf. Appl. Cryptogr. Netw. Secur., Banff, AB, Canada, 2013.

[11] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.

[12] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*. New York, NY, USA: ACM, 2018, pp. 35–52.

[13] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[14] Z. Tang, H. W. Lim, and H. Wang, "Revisiting a secret sharing approach to network codes," in *Provable Security* (Lecture Notes in Computer Science), vol. 7496. 2012, pp. 300–317.

[15] S. M. Pournaghi, M. Bayat, and Y. Farjami, "MedSBA: A novel and secure scheme to share medical data based on blockchain technology and attribute-based encryption," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 11, pp. 4613–4641, Nov. 2020.

[16] G. S. Wagh, S. Gupta, and S. Mishra, "A distributed privacy preserving framework for the smart grid," in *Proc. IEEE Power Energy Soc. Innov. Smart Grid Technol. Conf. (ISGT)*, Washington, DC, USA, Feb. 2020, pp. 1–5.

[17] D. Wang and X. Zhang, "Secure data sharing and customized services for intelligent transportation based on a consortium blockchain," *IEEE Access*, vol. 8, pp. 56045–56059, 2020, doi: 10.1109/ACCESS.2020.2981945.

[18] L. Liu, H. Wang, and Y. Zhang, "Secure IoT data outsourcing with aggregate statistics and fine-grained access control," *IEEE Access*, vol. 8, pp. 95057–95067, 2020, doi: 10.1109/ACCESS.2019.2961413.

[19] F. El Mahdi, A. Habbani, Z. Kartit, and B. Bouamoud, "Optimized scheme to secure IoT systems based on sharing secret in multipath protocol," *Wireless Commun. Mobile Comput.*, vol. 2020, pp. 1–9, Apr. 2020.

[20] M. Mohanty, "Secret sharing approach for securing cloud-based image processing," Ph.D. dissertation, Nat. Univ. Singapore, Singapore, 2013.

[21] P. Sarosh, S. A. Parah, and G. M. Bhat, "Utilization of secret sharing technology for secure communication: A state-of-the-art review," *Multimedia Tools Appl.*, vol. 80, no. 1, pp. 517–541, Jan. 2021, doi: 10.1007/s11042-020-09723-7.

[22] J. L. Dautrich and C. V. Ravishanka, "Security limitations of using secret sharing for data outsourcing," in *Data and Applications Security and Privacy XXVI* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2012, pp. 145–160.

[23] M. A. Hadavi, R. Jalili, E. Damiani, and S. Cimato, "Security and searchability in secret sharing-based data outsourcing," *Int. J. Inf. Secur.*, vol. 14, no. 6, pp. 513–529, Nov. 2015.

[24] R. Ghasemi, "Resolving a common vulnerability in secret sharing scheme–based data outsourcing schemes," *Concurrency Comput., Pract. Exper.*, vol. 32, no. 2, p. e5363, Jan. 2020.

[25] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2002, pp. 216–227.

[26] S. Ruggles, J. T. Alexander, K. Genadek, R. Goeken, M. B. Schroeder, and M. Sobek, "Integrated PublicUse microdata series: Version 5.0 [Machine-readable database]," Univ. Minnesota, Minneapolis, MN, USA, Tech. Rep., 2010.

ZHAOHUI TANG (Member, IEEE) received the Ph.D. degree in information security. He is currently with the University of Southern Queensland, Australia. His research interests include information and cyber security, particularly, in applied cryptography and the Internet-of-Things (IoT) security areas.