A secure data outsourcing scheme based on Asmuth-Bloom secret sharing

Yusuf I. M.[1], Mustafa Kaiiali[1], Adib Habbal[2], Wazan A. S.[3], Auwal S. I.[1]

[1]*Department of Computer Engineering, Mevlana University, Konya, Turkey*
idrisawaa@gmail.com, mkaiiali@mevlana.edu.tr, engrausan@gmail.com

[2] *InterNetWorks Research Lab, School of Computing,*
Universiti Utara Malaysia, 06010 UUM Sintok, Kedah Malaysia
adib@uum.edu.my

[3] *Institut de Recherche en Informatique de Toulouse(irit), France*
ahmad-samer.wazan@irit.fr

A secure data outsourcing scheme based on Asmuth-Bloom secret sharing

Data outsourcing is an emerging paradigm for data management in which a database is provided as a service by third party service providers. One of the major benefits of offering database as a service is to provide organizations, which are unable to purchase expensive hardware and software to host their databases, with efficient data storage accessible online at a cheap rate. Despite that, several issues of data confidentiality, integrity, availability and efficient indexing of users' queries at the server side have to be addressed in the data outsourcing paradigm. Service providers have to guarantee that their clients' data is secured against internal (insider) and external attacks. This paper briefly analyses the existing indexing schemes in data outsourcing and highlights their advantages and disadvantages. Then it proposes a secure data outsourcing scheme based on Asmuth-Bloom secret sharing which tries to address the issues in data outsourcing such as data confidentiality, availability and order preservation for efficient indexing.

Keywords: data outsourcing; indexing; secret sharing.

Introduction

The rapid increase in the amount of data held by organizations has drastically increased the cost of in-house data management. Services providers that share resources across different organizations are therefore required in order to reduce the cost [1]. In-house data management cost was estimated to be 5-10 times higher than the initial acquisition cost [2].

Despite the economic advantage of data outsourcing, it faces a lot of challenges to get widespread acceptance [3]. Data owners are cautious to place their secret information under the control of a third party without the guarantee of having an efficient and robust service that manages the information in a secure and privacy preserving manner. Studies have shown that users' secret information stored at the premises of

service providers is susceptible to insider attacks; for example employees of online social network can know which profiles a user has visited [4]. Some may sell their users data to third party companies who can use them for commercial purposes [5]. In addition, governments may have a direct access to users' private information through collaboration with service providers, as revealed in 2013 by one of the NSA contractors Edward Snowden [6, 7]. For data outsourcing to take over in-house data management, these issues have to be adequately addressed.

The prominent techniques used in securing outsourced data are Data encryption [8], Homomorphic encryption [9], Secret Sharing algorithms and Private Information Retrieval (PIR) [10].

Encryption has been a traditional solution to the issue of confidentiality. However, performing database manipulations operations such as insert, update, delete and query on an outsourced encrypted data imposes an extra overhead due to the several encryption and decryption processes that need to take place at the client side [11-14].

In homomorphic encryption, data processing operations are performed on an encrypted outsourced data at the server side while the client only verifies the correctness of the answer without spending much computational effort [15]. This reduces the overhead of encryption and decryption processes that have to be performed at the client side. However, homomorphic encryption is considered to be computationally expensive due to its dependence on public key cryptosystems [10].

Apart from data confidentiality, another important issue is data availability. "Downtime can cripple work flow and impose substantial cost - from time and money spent on repairs to lapses in productivity to the inability to meet service levels and compliance mandates" [16]. Data availability is usually achieved by duplicating the data

and storing it at different servers [10]. Secret sharing schemes and Information Dispersal Algorithms (IDAs) [17] are used to provide data availability.

Secret sharing [18] refers to a method of distributing a secret to a group of participants each of whom is allocated a share of the secret [19]. There are several types of secret sharing schemes. The most basic types are the threshold schemes, where only the cardinality of the set of shares matters [19, 20]. In other words, given a secret *S*, and *n* shares, any set of *t* out of *n* shares is a set with the smallest cardinality from which the secret can be recovered [21], in the sense that any set of (*t-1*) shares is not enough to give *S*. This is known as a threshold access structure and is also known as (*t*, *n*) threshold secret sharing scheme [22]. Several secret sharing schemes have been employed in [2, 10, 23] to provide data availability for outsourced data.

In PIR protocols, the queries performed by the client on the database are hidden from the service provider. One trivial but inefficient way to achieve PIR is for the server to send the entire encrypted database to the client. Then the client has to decrypt and execute the query locally. This provides privacy but introduces communication latency and computational overhead on the client especially for large databases [24].

Due to this, many approaches were designed to enable an efficient execution of different types of queries on an encrypted data e.g. encryption based indexing [8], homomorphic encryption [9], partition based indexing [25], hash based indexing [26], B+ tree indexing [25], and order preservation encryption [27] (order preservation is a very important aspect in a data outsourcing scenario, in order to support comparison operations to be directly applied on encrypted/shared data at the server side without the need to retrieve and decrypt it at the client side).

Apart from providing data as a service, the data outsourcing model has been incorporated in many real-time applications to solve users' privacy issues. As an example, several data outsourcing paradigms have been proposed to address the privacy issue that exists in Online Social Networks [28, 29] such as flyByNight [5], faceCloak [30] and Cloud-based OSN projects [31].

This paper proposes a new data outsourcing scheme based on Asmuth-Bloom Secret Sharing (ABSS) [32]. ABSS is a threshold scheme based on the Chinese Remainder Theorem. It requires $O(r)$ modular operations to recover a shared data as compared with other secret sharing schemes such as Shamir Secret Sharing which is sensitive to error and uses an interpolation formula that requires $O(r\log^2 r)$ operations. ABSS is secure in the sense that no useful information can be recovered out of $(r - 1)$ shares. ABSS can be modified to include the option of checking the validity of the shared outsourced data before recovery [32]. For the previously mentioned advantages of ABSS, this paper proposes a new data outsourcing scheme based on ABSS that addresses several security challenges existing in data outsourcing, namely, availability, confidentiality and order preservation efficiently. There are other schemes that offer one [8, 9] or two [19, 21] of these characteristics, however, our proposed scheme supports all three characteristics in an efficient and computationally less expensive manner.

The rest of this paper is organized as follows: Section II discusses the data outsourcing model. Section III gives a detailed review of related work. Section IV discusses the secret sharing scheme using Chinese Remainder Theorem (CRT). Section V presents a direct implementation of the Asmuth-Bloom secret sharing scheme in data outsourcing. In Section VI, an enhanced data outsourcing model based on the Asmuth-Bloom secret sharing scheme is proposed. Then, Section VII presents a security analysis of the proposed scheme. Finally, the paper concludes in Section VIII.

Data outsourcing model

A data outsourcing model [34] consists of the following entities depicted in Figure 1.

- **Data owner:** the entity that owns the data and wishes to outsource it.
- **User:** an entity that needs to access the outsourced data published by the data owner.
- **Service provider:** an entity that stores and manages the outsourced data.
- **Client**: the only party trusted by all the entities involved in a data outsourcing scenario. The client transforms queries presented by users into equivalent queries that operate on the encrypted database stored on the server side.

The data owner and the clients rely upon the service provider for the availability of the outsourced data. However, the confidentiality of the outsourced data is not trusted with the service provider as it may contain sensitive information that the data owner needs to reveal only to authorized users. Therefore, it is necessary to prevent the service provider from having unauthorized access to the outsourced data.

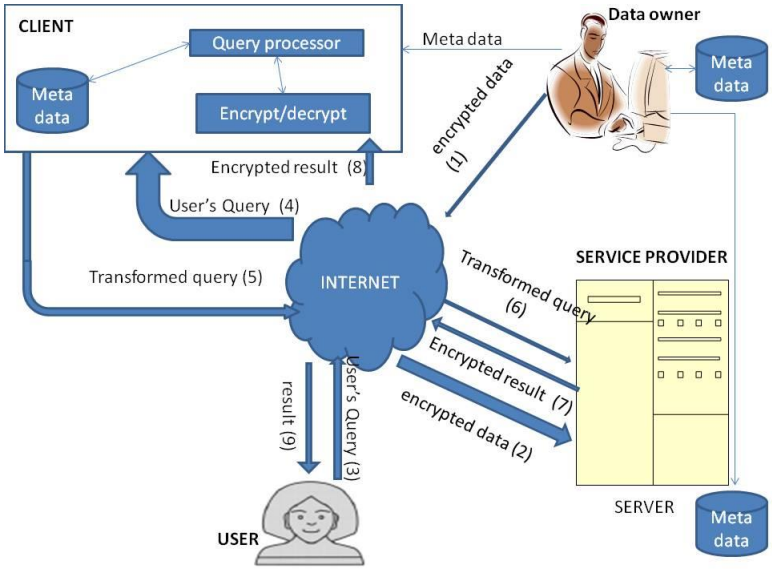


Figure 1: Data outsourcing model

The Figure above illustrates how the data outsourcing model works. The data owner encrypts his/her data and outsources it to the third party service provider where it will be stored and managed in an encrypted form. Whenever a user wants to access the outsourced data, he/she sends his/her query to the client where it will be transformed to a corresponding query that can work on the encrypted data at the server side. The server then responds to the request and sends the encrypted result back to the client, where it will be decrypted, further processed and finally sent to the user.

Related works

A lot of techniques were designed to support queries on encrypted relational databases and to protect the outsourced database against compromises in confidentiality, integrity and availability. However, some of these techniques have certain limitations which make them inefficient.

The table below illustrates a simple database example used throughout this paper. It is a database of employees' information. The data owner may outsource this database to multiple Data Service Providers (DSP_j).

Table 1: Employees' table

S/N	Fname	Salary
1	Alicia	10
2	Bob	18
3	Carol	21
4	Danna	36
5	Ebe	67
6	Frank	69
7	Gabza	71
8	Habu	81
9	Ibro	93
10	Joseph	238

Encryption based indexing approach

Database encryption refers to the use of encryption techniques to transform a plaintext database into an encrypted database, thus making it unreadable to anyone except those who possess the decryption key [8].

Encryption protects the exposure of sensitive information even if the server is compromised, and ensures its integrity since data tampering can be detected. Data decryption cannot be executed at the server side, therefore solutions have been developed that allow the server to execute queries directly on the encrypted data [12].

In [8], an encryption approach is employed in which the attributes of the relation are encrypted and used as an index. The index value $I_i[j]$ associated with the attribute value $a_i[j]$ in tuple j is computed as $E_k(a_i[j])$, where E_k is a conventional encryption function and k is the encryption key.

The advantages of this approach are:

- ✓ It supports the confidentiality service.
- ✓ It supports equality. As an example, searching for a tuple in an outsourced encrypted data whose attribute a_1 value in Table 1 is equal to *10* will be directly translated by the client into searching for a tuple in the encrypted relation whose I_1 value is equal to $E_k(10)$ [25].

The limitations of this approach are:

- × It does not support range queries because conventional encryption need not be order preserving [27].
- × It does not support availability.

Homomorphic encryption

Homomorphic encryption is a form of encryption that allows mathematical operations to be performed on encrypted data without compromising the encryption, and the result when decrypted should match the operations on the plaintext.

Homomorphic encryption is desirable in modern communication system architectures especially in Cloud Computing [8]. This is because it can allow the chaining of different services together without exposing the data to each of those services [35]. For example, a chain of different services from different companies could be: calculate the tax, the currency exchange rate and the shipping. All of these services are needed within a single transaction without exposing the unencrypted data to any of them [36]. This feature makes this scheme good in providing confidentiality of an outsourced data, though it does not preserve its order due to the encryption process.

In [9], a privacy homomorphic encryption that allows basic arithmetic operations to be performed over encrypted data is employed. In this scheme two functions α and β are defined over the domains of unencrypted and encrypted values respectively. If *A* is a domain of unencrypted values, E_k is an encryption function with key *k*, and D_k is the corresponding decryption function, then (E_k, D_k, α, β) can be defined as a privacy homomorphism if:

$$D_k(\beta_i(E_k(a_1), E_k(a_2), \ldots, E_k(a_n))) = \alpha_i(a_1, a_2, \ldots, a_m): 1 \leq i \leq n$$

To add two values a_1 and a_2 which are encrypted and stored at the server side the system goes through the following steps; assuming $E_k(a_i) = (a_i \bmod p, a_i \bmod q)$ while (p, q) are prime numbers representing the shared secret key *k* between owner and client):

- Compute $n = p.q$, where *n* is a public parameter.
- The server is instructed to compute $E_k(a_1) + Ek(a_2)$ and then return the result to the client
- The client decrypts the result using the function: $(d_1qq^{-1}+d_2pp^{-1}) \pmod n$, where $d_1 = a_1 \pmod p$, $d_2 = a_2 \pmod q$, $qq^{-1} = 1 \pmod p$ and $pp^{-1} = 1 \pmod q$.

As an example, if $k = (p = 5, q = 7)$ and $a_1 = 10, a_2 = 18$ are two salaries of two different employees (as shown in Table 1) ⇒ $E_k(a_1) = (10 \bmod 5, 10 \bmod 7) = (0, 3)$ and $E_k(a_2) = (18 \bmod 5, 18 \bmod 7) = (3, 4)$. To add a_1 and a_2 stored encrypted at the server:

- $n = p.q = 5 \times 7 = 35$
- $E(a_1)+E(a_2) = (0+3, 3+4) = (3, 7)$
- The client finally decrypt the result using the function $(d_1qq^{-1}+d_2pp^{-1}) \pmod n = (3\times7\times3 + 7\times5\times3) \bmod (35) = 168 \bmod 35 = 28$ which is the addition of a_1 to a_2.

The advantages of homomorphic encryption are:

- ✓ It supports confidentiality.
- ✓ It supports equality, aggregation and logical operations on the outsourced data [9].

The limitations of this approach are:

- × It is computationally expensive due to its dependence on PKI [10].
- × It does not preserve the order of the outsourced data so it cannot execute range queries.
- × It does not support the availability service.

Partitioned based indexing approach

In this approach the attribute domain D_i is partitioned into contiguous subsets of values of the same size without overlapping. Each partition is augmented with a label in an ordered or random manner in the domain D_i.

In [25], an architecture is proposed based on a partition based indexing approach. It comprises of three entities: a user, a server and a client (similar to Figure 1). A user poses a query to a client to transform it to an equivalent query that can run on encrypted data at the server side. The encrypted database is augmented with an index to make the

encrypted data queryable. Based on the index, a technique is developed to split the user's query over encrypted data into 2 queries; a server query runs on the encrypted data and a client query runs on the client side to post-process the results obtained from the server query. To split the query, three functions are introduced: a partitioning function, an identification function and a mapping function.

- The partition function splits the domain values of attributes into partitions of equal length using a histogram construction technique (e.g. MaxDiff [37] or equi-depth [38]) such that it covers the whole domain.
- The identification function assigns a unique identifier to each of these partitions. To avoid collisions, a collision free hash function can be used for this purpose.
- The mapping function maps each of the partitions to its identifier.

The advantages of this approach are:

✓ It supports the confidentiality service.

✓ It allows server evaluation of the equality condition [34].

✓ The partitioning of the domain values into contiguous subsets makes it possible to partially preserve the order of the outsourced data [34].

The limitations of this approach are:

× It can produce spurious tuples. Spurious tuples are tuples that satisfy the correspondent query condition over the indexes of the outsourced data but do not satisfy the query condition over the original plaintext [34]. These spurious tuples need to be further processed and eliminated at the client side.

× It partially supports range queries [34].

× It does not support availability.

Hash based indexing

The hash based indexing technique uses as an index the result of a secure hash function over the attribute values of a relation. The hash function can be adapted for different granularities of the presented data. As an example if the co-domain C of the hash function is small compared to the number of attribute values, the hash function distributes the tuples uniformly in |C| buckets.

With respect to direct encryption, hash based indexing provides more confidentiality, however, it does not preserve the order of the outsourced data. When hashing is used, different plaintext values are mapped onto the same index, so the query results often produce spurious tuples that need to be further processed by the client [26].

As an example, consider the relation in Table 2 of an encrypted employee table Employee^k. The indexes of the attributes Fname and Salary in relation Employee (shown in Table 1) are computed by applying the hash-based method. Three distinct values namely α, β and γ have been mapped to the values of the attribute Fname while four distinct values ϵ, θ, ϕ and σ have been mapped to the values of the attribute Salary.

Table 2: Employee^k

S/N	Encrypted tuple	Index$_1$	Index$_2$
1	tyuR34+pcn	α	ϵ
2	Kljm$=4tye	β	σ
3	Frgshgajk2	γ	σ
4	Njwklp345y	α	θ
5	Rnozh12p*0	β	ϵ
6	Yajkie409@	γ	ϕ
7	WblpAeNR%7	α	ϕ
8	yusbr)@jkl	γ	θ
9	pls-2wema>	β	θ
10	Rtcbrtmlas	α	σ

The advantages of hash based indexing are:

- ✓ It is more confidential than direct encryption based indexing [5].
- ✓ It allows server side evaluation of the equality condition. As an example each condition $a_{ij} = v$, where v is a constant value is transformed into a condition $I_{ij} = h(v)$, where I_{ij} is the index corresponding to a_{ij} in the encrypted relation [5].

The limitations of hash based indexing are:

- × It produces spurious tuples in cases where the hash function is not collision free, which adds a burden to the client side [34].
- × It is not order preserving, therefore it does not support range queries [34].
- × It does not support the availability service.

B + tree indexing approach

A B+ tree can be viewed as a B-tree [39] in which each node contains only keys (not key-value pairs), and to which an additional level is added at the bottom with linked leaves. It consists of leaves, internal nodes and a root. The root may be either a leaf or a node with two or more children. The main essence of a B+ tree is in storing data for efficient data processing operations [40].

Damiani et al. [8], propose an indexing method based on the B+ tree that can be attached to an encrypted database, which can be used by the server to select the data to be returned in response to a query without the need for disclosing the database content.

Given an internal vertex storing f key values $k_1, \ldots, k_f$ with $f \leq n-1$, each key value k_i is followed by a pointer p_i, and k_1 is preceded by a pointer p_0. Pointer p_0 points to the subtree that contains keys with values lower than k_1, p_f points to the sub-tree that contains keys with values greater than or equal to k_f, and each p_i points to the sub-tree that contains

keys with values included in the interval [k_i, k_{i+1}]. Internal vertices do not directly refer to tuples in the database, but just point to other vertices in the structure. Leaf vertices directly refer to the tuples in the database that have a specific value for the indexed attribute. Leaf vertices are linked in a chain that allows the efficient execution of range queries.

A B+ tree is built over the most queried attribute a_{ij} of a database D_i. It is then represented as a relational table, encrypted and stored at the server side along with the encrypted database [34]. The relational table consists of two attributes:

- Id - represents the identifier of a vertex, and
- Vertexcontent - represents the content of a vertex.

Each row in the relation represents a vertex of the tree, while the pointers are represented through cross references from the vertex content to other vertex identifiers in the relation. Encryption is applied at the vertex level in order to protect the order relationship among plaintext, index values and the mapping between the two domains [41].

To search for a tuple with key k in B+ tree, the client starts searching from the root vertex, which is a tuple with $Id = 1$, retrieves it and decrypts it to see whether it satisfies the given condition. If not, it then traverses through the nodes, and at each node retrieves the content of the vertex and decrypts it until the leaf containing the key k is found.

This kind of indexing approach provides both confidentiality and order preservation, as the outsourced data is stored encrypted with a B+ tree built over the most queried attribute for efficient data processing operation. However, it does not support availability.

Figure 2 illustrates how this scheme is applied to the employee table (Figure 2 (a)) encrypted (Figure 2 (b)), the B+ tree is built over the most queried attribute of the database (Figure 2 (c)), it is then represented in a relational table (Figure 2 (d)), encrypted (Figure 2 (e)) and stored along with the encrypted database at the server side.

The advantages of the B+ tree indexing approach [34] are:

- ✓ It supports confidentially.
- ✓ It is secure against inference attacks, since the B+ tree content itself is encrypted.
- ✓ It supports equality queries.
- ✓ It is order preserving so it can support range queries.
- ✓ It does not produce spurious tuples.
- ✓ It allows the evaluation of ORDER-BY and GROUP-BY clauses of SQL queries.

The limitations of the B+ tree indexing approach are:

- × B+ tree indexing is expensive for the client compared to bucket and hash based. Several trips (as many as the levels we have in the B+ tree) between the client and the server are needed to retrieve the required tuples [8].
- × It does not support the availability service.

S/N	Roll Number	Name	Department	Salary
1	9992345	Alice	Electrical Engr.	10
2	9992378	Bob	Computer Engr.	18
3	9992367	Donna	Computer Sci.	21
4	9992980	Elvis	Political Sci.	67
5	99923567	Frank	Electrical Engr.	69
6	99922567	Gary	Human Resources	71
7	99921070	John	Human Resources	81
8	99921071	Kamal	Political Sci.	216
9	99921080	Lewis	Political Sci.	228
10	99921081	N	Computer Sci.	248

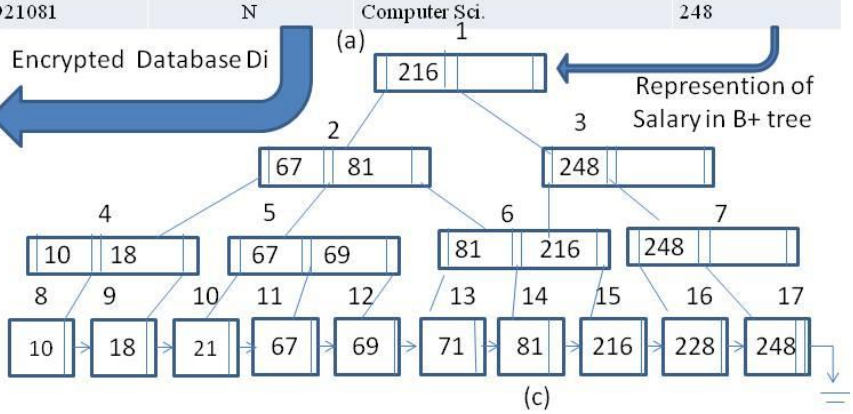
(a)

S/N	Etuple
1	+Z7kaVaDI1
2	FzCs4HjK"w
3	dehZ0hcCQ#
4	2sVOl/l2Fyr
5	498PWtEXw
6	JO5cSX7nRa
7	6SE6zB3aeV
8	5rtklm$op0w
9	dkelo0@jlmk
10	P0nm%p0df

(b)

Encrypted Database Di

Represention of Salary in B+ tree



(c)

Id	Vertexcontent
1	2, 216,3
2	4,67,5,81,6
3	6,248,7
4	10,8,18,9
5	10,67,11,69,12
6	13,81,14,216,15
7	16,248,17
8	10,9,t_1
9	18,10,t_2
10	21,11,t_3
11	67,12,t_4
12	69,13,t_5
13	71,14,t_6
14	81,15,t_7
15	216,16,t_8
16	228,17,t_9
17	248,NIL,t_{10}

(d)

Encryption of The Vertexes

Id	E(Vertexcontent)
1	+Z7kaVaDI1
2	FzCs4HjK"w
3	dehZ0hcCQ#
4	2sVOl/l2Fyr
5	498PWtEXw
6	JO5cSX7nRa
7	6SE6zB3aeV
8	aaHhgf900''
9	SDFV7kaNM=
10	AKDBNGHu
11	SA'Kubbns^k
12	3UYFs89kl*
13	Tyun68.op0
14	Jsht4{{nmw
15	7nsytimws)oi
16	Hshs4?lpqp
17	Fsha45?,[]p

(e)

Figure 2. Example of the B+ tree indexing approach

Order preservation encryption scheme (OPES)

The idea behind OPES is that unique values are generated from a user specified target distribution and sorted in a table *T*. Then the i^{th} plaintext (p_i) value in the sorted list of |*P*| plaintexts is encrypted into the i^{th} value in the sorted list of |*P*| values obtained from the user's specified target distribution. This makes the scheme good in preserving the order of the encrypted data and also provides confidentiality. To decrypt any of the ciphertext a lookup into a reverse map is required [27]. In this case the table *T* is the encryption key that must be kept secret.

OPES uses three stages to encrypt a database:

- Model Stage
- Flatten Stage and
- Transform stage.

Model stage:

In this stage the input and the target distributions are modeled as piece wise linear splines using either of the following techniques.

- Histogram technique: this technique captures statistical information about the data value distributions using counters for a specified number of data value buckets [42]
- Parametric technique: this technique approximates data value distributions by fitting the parameters of a given type of function (e.g. polynomial of a given maximum degree) [42].

In [43], the two modeling techniques were used to first of all partition the values into buckets using the histogram-based technique then model the distribution within each

bucket as a linear spline (a linear spline of a bucket is a line connecting the densities at the two end points of a bucket) using the parametric technique.

Flatten stage:

In this stage the plaintext database *P* is transformed into a flat database *f* such that values in *f* are uniformly distributed. In cases where a bucket has dense plaintext, the bucket will be stretched so that the density of the plaintext in the flattened bucket will be uniform.

Transform stage:

In this stage the flattened database is transformed into a cipher database *C* such that the values in *C* are distributed according to the target distribution.

As an example, if we have p_i and p_j where $p_i < p_j$, this will be transformed to $f_i < f_j$ in the flatten stage, and finally the flattened buckets will be transformed to a uniformly distributed cipher database $c_i < c_j$. (i.e. $P_i < P_j \Rightarrow f_i < f_j \Rightarrow C_i < C_j$)

In [44], an order preserving indexing scheme over encrypted data is proposed based on a simple mathematical expression ($av + b + noise$). The result of the expression is made public while the coefficients $\boldsymbol{a}$ and $\boldsymbol{b}$ are the secret keys known only to the data owner and the client. $\boldsymbol{v}$ is the value to be indexed and *noise* is randomly chosen from the range $\{0, \ldots, a - 1\}$ to preserve the order of the index. One of the limitations of this scheme is that the secret keys $\boldsymbol{a}$ and $\boldsymbol{b}$ are reused throughout the system, which makes the scheme insecure. Adding noise randomly means that even the same encrypted values may not have the same noise, which may give false queried information. Another limitation is when an attacker obtains the ciphertext c_1 for plaintext zero (i.e. $c_1 = b + n_1$) and ciphertext c_2 for large plaintext k (i.e. $c_2 = ak + b + n_2$), then by computing the difference between the two ciphertexts, the attacker obtains ($c_2 - c_1 = ak + n_2 - n_1$). Since the randomly selected

noises n_1 and n_2 are between zero and a (i.e. $0 \leq n_1, n_2 < a$) then the attacker learns that the secret a is in the interval $I_1 = \frac{c_2 - c_1}{k+1} \leq a \leq I_2 = \frac{c_2 - c_1}{k-1}$ [46, 47].

The advantages of the OPES indexing approach are [27]:

- ✓ It supports the confidentiality service.
- ✓ It is order preserving.
- ✓ It produces exact results for the query without any spurious tuples.
- ✓ It can handle updates, without the need for re-encryption.

The limitations of the OPES indexing approach are:

- × It is vulnerable to tight estimation and statistical attacks [45].
- × It is vulnerable to chosen plaintext attacks [45].
- × It does not support availability.

Information dispersal algorithms (IDAs)

IDAs were first proposed by Michael O. Rabin in 1989 to protect data sent over the network or sitting in storage arrays and not to be accessed except by the right user [48].

In IDAs, a file is split into *n* pieces where a minimum *m* pieces (*n* > *m*) are required for reconstructing the original file. A transform matrix of *n* rows and *m* columns is used to transform the original file into *n* pieces [47].

In a salted IDA, confidentiality is achieved by using a modified IDA scheme. A salted IDA depends on randomness to improve data confidentiality. While it relies on the original IDA scheme to provide data availability [50].

A transform matrix TM of (*n* x *m*) is maintained by the client as the information dispersal matrix and the keys for encoding and decoding a data matrix *D*. *n* and *m* are

determined by the client based on the number of servers that are planned to be used and the estimated number of non-faulty (available) servers. Also the client keeps a secret *ss* (a shared secret between the data owner and client) and a deterministic function *fs* for producing random factors (called salts) based on the address of the data entries on *D*. Function *fs* feeds *ss* and the address of the data entry into a pseudorandom number generator (PRNG) before encoding and dispersing *D* onto the servers, for each column *i* of *D* the client calls the *PRNG* procedure *i* times, sets the last generated random number (the output of the i^{th} round) as the salt to each data entry of column *i*. The data matrix after adding the salt is called a tuple matrix TD. The essence of the generated random number was to improve the confidentiality of the original IDA.

It is difficult to search for data in an encoded matrix based on plaintext input. To solve this issue, a B+ tree index was built on the key attribute and kept secured at the client side. The leaf nodes of the index tree contain pointers to the columns of the tuple matrix (TD) where the tuple with its key are stored.

The Index matrix (ID) and TD are encoded into IE = (ID × TM) and TE = (TD × TM) respectively and then dispersed onto *n* servers, S_1, S_2, ..., S_n, by the salted IDA as shown in Figure 3. Queries on the index key attribute can be efficiently processed by locating the columns of ID (tree nodes) that store the query keys and then retrieving the corresponding tuples from the columns of TD.

After decoding the encoded data retrieved from *m* non-faulty servers, the client reconstructs the salts by calling f_s and then deducts these salts from the decoded data entries, recovering *D*.

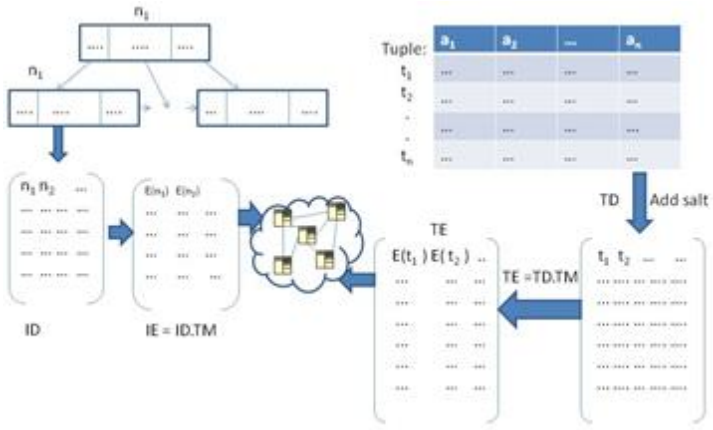


Figure 3: Framework for dispersing data in a cloud using salts [50]

The advantages of the IDA are [50]:

- ✓ It provides a confidentiality service.
- ✓ It supports equality queries.
- ✓ It is order preserving, and as such it supports range queries.
- ✓ It supports availability.
- ✓ It does not produce spurious tuples.
- ✓ It is secured against statistical analysis of the encrypted data.

The limitations of the IDA are [50]:

- × For every query the whole column where the candidate answer tuple is located has to be retrieved.
- × It is expensive for the client.

Table 3 compares the reviewed approaches in the literature based on six criteria: support for equality queries and/or range queries, the cost of performing operations at the client side, returning spurious tuples, data availability and data confidentiality. It is obvious that none of the aforementioned approaches are able to adequately address all the data outsourcing issues. Every approach has some limitations. This motivates us to propose a new data outsourcing scheme that tries to address all the issues.

Table 3: Comparison of the existing approaches

Approaches	Equality Queries	Range Queries	Client Burden	Spurious Tuples	Availability	Confidentiality
Encryption based indexing	Fully	No	High	Yes	No	Yes
Homomorphic encryption	Fully	No	High	Yes	No	Yes
Partitioned based indexing	Fully	Partially	Partially	Yes	No	Yes
Hash based indexing	Fully	No	High	Yes	No	Yes
B+ tree indexing	Fully	Yes	High	No	No	Yes
Order preservation encryption	Fully	Fully	High	No	No	Yes
IDAs	Fully	Fully	High	No	Yes	Yes

A secret sharing scheme using Chinese Remainder Theorem (CRT)

A secret sharing algorithm is one of the prominent techniques widely used in data outsourcing [3, 15, 29, 55]. It is an ideal scheme for storing sensitive and highly important information. Traditional encryption methods, as discussed earlier, are not efficient for achieving a high level of confidentiality and availability at the same time.

The aim of this paper is to explore a secret sharing algorithm based on the Chinese Reminder Theorem (CRT) to provide data availability, confidentiality and order preservation for outsourced data.

The secret sharing scheme based on CRT produces shares presented in congruence equations and the secret can be recovered by solving the system of congruencies to get a unique solution, which is the original outsourced data. Secret sharing schemes that are based on CRT are proved by Quisquater et el [49] to be asymptotically perfect schemes [51].

CRT Theorem:

Suppose that $m_1, m_2, \ldots, m_k$ are pairwise relatively prime positive integers. Then for any given integers $a_1, a_2, \ldots, a_k$, the following system of simultaneous congruencies

$$x \equiv a_i \,(mod\ m_i) \text{ for } 1 \leq i \leq k$$

has a unique solution x modulo the product $M = m_1 \ldots m_k$.

This is given by:

$$x \equiv a_1 M_1 y_1 + a_2 M_2 y_2 + \ldots + a_k M_k y_k (mod M)$$

where $M_i = \frac{M}{m_i}$ and $y_i \equiv (M_i)^{-1} (\text{mod } m_i)$

Asmuth-Bloom threshold scheme

Asmuth-Bloom secret sharing is a threshold scheme that is based on CRT. It was proposed with the intent of safeguarding a key [34]. In this scheme, a key K is decomposed into n shares I_i (congruence classes of a number associated with the original key), in such a way that the key is only recoverable from any $r \leq n$ of the shares, and no useful information can be recovered from $(r - 1)$ shares.

The value of this scheme depends on the following features [34]:

- the efficiency of decomposition and recoverability of the key,
- the sensitivity of the method to random error or deliberate tempering,
- the relationship between the number of shares (n), the minimum number of shares required to recover the secret (r) and the number of shares that is not enough to recover the secret ($r - 1$).

The Asmuth-Bloom threshold scheme consists of two phases [52]:

- Environmental formation phase
- Secret recovery phase.

Environmental formation phase:

This phase consists of two stages:

Initialization stage: In this stage, a set of integers $\{p, m_1, m_2, \ldots, m_n\}$ are chosen subject to the following conditions:

- $m_1 < m_2 < \ldots < m_n$
- $gcd(m_i, m_j) = 1$ for $i \neq j$
- $gcd(p, m_i) = 1$ for all i
- $\prod_{i=1}^{r} m_i > p \prod_{i=1}^{r-1} m_{n-i+1}$
- $p > K$

Decomposition stage: In this stage, the decomposition process goes as follows:

- compute $M = \prod_{i=1}^{r} m_i$
- compute the parameter $I = K + \beta.p$, where $0 \leq I < M$ and β is an arbitrary integer chosen from the range $\left[0, \frac{M}{P} - 1\right]$
- then decompose the Key using $Ii \equiv K(mod\ m_i)$ and share it to the users with the i^{th} user having the I_i share where $1 \leq i \leq n$

Secret recovery phase:

Suppose we have a system of congruencies $I_i \equiv K\ (mod\ m_i)$. The secret K can be uniquely recovered using a constructive algorithm as follows [53]:

- compute the product of the r primes that are required to recover the secret as $M=m_1 m_2 \ldots m_r$
- for each prime compute $M_i = \frac{M}{m_i}$
- use an Extended Euclidean Algorithm to find the integers α_i and γ_i such that:

$$\alpha_i.m_i + \gamma_i\,(M/m_i) = 1$$

$$\text{Let } e_i = \gamma_i\,(M / m_i) \Rightarrow \alpha_i.m_i + e_i = 1$$

e_i in the above equation guarantees that its remainder when divided by m_i must be 1, i.e. $e_i \equiv 1 (mod\ m_i)$

Hence, the system of congruencies has one solution that is $K = \sum_{i=0}^{r} I_i e_i$ based on CRT. Shares construction in the Asmuth-Bloom secret sharing scheme has a complexity of *Olog(r)* [49].

The Asmuth-Bloom scheme can be modified to provide a validity check and a deliberate tempering of the shares. To make the scheme perform a validity check to the shares of the key, the second condition of the initialization stage has to be weakened to allow $gcd(m_i, m_j) = q_{ij}$ where $q_{ij} \neq 1$. As an example if I_i and I_j are known and $gcd(m_i, m_j) = q_{ij}$, then for the correct shares $I_i \equiv I_j\ (mod\ q_{ij})$.

If there is an error in I_i, this will alter its congruencies class modulo all of the q_{ij}, therefore the error free shares would be in general agreement with each other and those in error can be discarded [32].

Direct implementation of Asmuth-Bloom secret sharing schema in data outsourcing

This section proposes a new data outsourcing scheme based on Asmuth-Bloom threshold secret sharing. It consists of three stages:

Setup:

In this stage the data owner performs the following steps:

- choose a set of prime integers $\{p, m_1, m_2, \ldots, m_n\}$ based on the following conditions:

- $p > d_i$: d_i represents a single attribute value
- $m_1 < m_2 < \ldots < m_n$, where n is the number of DSP
- $\gcd(m_i, m_j) = 1$ for $i \neq j$
- $\gcd(p, m_j) = 1$ for all j
- $\prod_{j=1}^{r} m_j > p \prod_{j=1}^{r-1} m_{n-j+1}$

- choose a random integer $\beta \in [0, \frac{M}{P} - 1]$ and compute $I_i = d_i + \beta.p$ for each data value (d_i) of the attribute to be used as an index for the outsourced database.

Outsourcing:

To distribute the database to multiple DSP servers, the data owner performs the following step:

- compute I_j' values for each I_i as the following $I_j' \equiv I_i (\boldsymbol{mod}\ m_j)$ and then distribute them to n service providers along with the encrypted database.

  The essence of each I_j' value is to serve as an order-preserving index for the encrypted database outsourced to the j^{th} server.

Retrieving:

To retrieve the outsourced encrypted data, the server uses the modulus as an index to retrieve the encrypted data and decrypt it at the client side.

The example below illustrates how this scheme can be used to build an index on the salary attribute of the employee database (Table 1). Assume the data owner needs to outsource the database of his/her employees to five different service providers (DSP_j).

Let $m_1=101$, $m_2=103$, $m_3=107$, $m_4=109$, $m_5=111$, $p=45$, $\beta=3$.

- Compute $I_i = d_i + \beta.p$ as shown in Table 4 column 3

- Then the salary can be shared using $I_j' \equiv I_i (\boldsymbol{mod}\, m_j)$ as shown in Table 4 columns (4-8)

Table 4: Indexes of the employee's Database to each DSP.

S/N	Salary (d_i)	I_i	Server 1 I_i (mod 101)	Server 2 I_i (mod 103)	Server 3 I_i (mod 107)	Server 4 I_i (mod 109)	Server 5 I_i (mod 111)
1	10	145	44	42	38	36	34
2	18	153	52	50	46	44	42
3	21	156	55	53	49	47	45
4	36	171	70	68	64	62	60
5	67	202	0	99	95	93	91
6	69	204	2	101	97	95	93
7	71	206	4	0	99	97	95
8	81	216	14	10	97	107	105
9	93	228	26	22	2	10	6
10	238	373	70	64	52	46	40

In the previous works, we can see that outsourcing by encrypting the whole data without building order-preserving indexes is prohibitive in terms of performance and does not solve the problem of availability. In contrast, ABSS can be used to create *n* shares and distribute it to different service providers as shown in Table 4. We can use these shares as indexes in various DSPs in order to achieve the following advantages:

- ✓ Partially preserve the order of the outsourced data (when $d_i < p$) to support range queries.
- ✓ Support equality queries.
- ✓ Reduce the computational cost of encryption.
- ✓ Reduce the burden of encrypting and decrypting on the client when making a query.

From Figure 4, we can extract the limitations of the direct implementation to this scheme as follows:

- × from an employee with *id = 1* to that with *id = 4*, the order of the index was preserved because the salaries are less than the chosen prime $p = 45$. But when $Max(d_i) > P$ the order cannot be preserved over all the attribute values. Thus

queries can produce spurious tuples. As an example, employees with *id = 4* and *id = 10* have the same index.

- By statistical analysis of the outsourced modulus values, an adversary can obtain the primes. For example, in Server 2 and Server 3, the highest moduli numbers are 101 and 107 respectively before it wraps around. Those values are very close to ($m_2 = 103$) and ($m_3 = 109$) values.

id	Salary
1	10
2	18
3	21
4	36
5	67
6	69
7	71
8	81
9	93
10	238

DSP1

id	Index	E(Salary)
1	44	#1anv;yuw
2	52	$89b5dja%
3	55	*s(1mfk3}
4	70	=s(1gkk3}
5	0	[79b5sjq%
6	2	q@medh6
7	4	s?dcms#th
8	14	@12hz+ef
9	26	Xbs&whw
10	70	Zxc!u!inm0

Figure 4: Index of employees' database in DSP_1.

Due to the issues raised in the direct implementation of ABSS in data outsourcing, the next section proposed an enhanced data outsourcing model based on ABSS that mitigates the aforementioned issues of direct implementation.

Enhanced data outsourcing model based on Asmuth-Bloom scheme

In this section, we have enhanced the direct implementation of Asmuth Bloom secret sharing to counteract its security threats. The enhanced scheme involves the following stages:

Setup:

In this stage the data owner initializes the data outsourcing environment through the following steps:

- choose an increasing polynomial function $f(d_i)$ (a function applied on the attribute values (d_i)) in which the variable is the attribute values to be outsourced and the coefficients are secrets to be selected by the data owner and shared with the client. Without loss of generality let us assume a degree 2 polynomial function of the following form:

$$f(d_i) = ad_i^2 + bd_i^1 + c \quad where\ a, b, c > 0$$

The above polynomial equation can have one positive and one negative root as there is single change of sign ($f(d_i) - ad_i^2 - bd_i^1 - c = 0$) which determines the number of positive roots that exist [54]. If the outsourced data contains single signed values (either positive or negative), the client can easily choose the correct root value when it solves the equation. However, in case the outsourced data contains both positive and negative values, we need to apply a scaling factor to make sure all the data values are of the same sign. The scale factor serves as an additional secret shared between the client and data owner. As an example if the expected data range is [-10, 25] then we can add a scaling factor ≥ 10 to have a new range of values that are all positive. If the scaling factor is selected as 10 then the new range will be [0, 35].

- choose a set of prime integers {p, m_1, m_2, …, m_n} such that :
  - $m_j < min(f(d_i))$
  - $m_1 < m_2 < \ldots < m_n$, where n is the number of DSP
  - $p << m_1$
  - $gcd(m_i, m_j) = 1$ for $i \neq j$
  - $gcd(p, m_j) = 1$ for all j
  - $\prod_{j=1}^{r} m_j > p \prod_{j=1}^{r-1} m_{n-j+1}$

- choose a random integer $\beta \in [0, \frac{M}{P} - 1]$ and compute $I_i = f(d_i) + \beta.p$ for each attribute value d_i, where $M = \prod_{j=1}^{r} m_j$

Outsourcing:

To distribute the database to multiple DSP servers, the data owner performs the following steps:

- compute the quotients $q_i = I_i \, div \, m_j$ and the modulus $I_j' \equiv I_i (\boldsymbol{mod}\, m_j)$ for all the attribute values,
- encrypt the modulus I_j' and store it with the unencrypted quotient q_i.

The essence of the introduced polynomial function *f* is to enhance system security as the coefficients are extra secrets known only to the data owner and the client. It is also used to enlarge the data to give a chance of selecting large co-prime integers m_j.

The quotient of the data is used as an index to maintain the order of the stored information at the server side, which will consequently help in supporting spurious-less range queries. The modulus is encrypted to keep the data secured as (m_j)s are selected big enough to make the quotient alone not sufficient to estimate the actual data.

Selecting large co-primes also helps in enhancing security because it increases the range that the modulus can take before it wraps around and thus reduces the chance of the adversary obtaining the modulus even in an unencrypted form.

Retrieving:

To retrieve the outsourced data the client performs the following steps:

- convert the queried data value into *n* quotient based query

- retrieve the correspondent tuples from at least *r* servers (*r* is the minimum number of servers out of which we can reconstruct the shared data) and decrypt the retrieved modulus
- use the following constructive algorithm to obtain I_i

$$I_i = \sum_{j=1}^{r} I_j'.e_j$$

where $e_j = \gamma_j\left(\frac{M}{m_j}\right)$ is computed by the Extended Euclidean Algorithm out of $\alpha_j.m_j + e_i = 1$

- compute the value I_i in $mod \prod_{j=1}^{r} m_j$
- subtract the value $\beta.p$ from I_i to obtain $f(d_i)$
- solve the polynomial equation to obtain d_i.

Let us illustrate how the scheme can be applied on the Employees table (Table 1).

Setup:

- let $f(d_i) = ad_i^2 + bd_i + c$: the coefficients $a=2$, $b=1$ and $c=1$ as shown in Table 5
- let $p = 45$, $m_1 = 101$, $m_2 = 103$, $m_3 = 107$, $m_4 = 109$, $m_5 = 111$
- let $\beta = 3$
- compute $I_i = f(d_i) + \beta.p$ for each attribute value d_i as shown in Table 5

Table 5: Environmental Set up

S/N	Salary (d_i)	$f(d_i) = ad_i2 + bd_i + c$	$I_i = f(d_i)+\beta.p$
1	10	211	346
2	18	667	802
3	21	904	1039
4	36	2629	2764
5	67	9046	9181
6	69	9592	9727
7	71	10154	10289
8	81	13204	13339
9	93	17392	17527
10	238	113527	113662

Outsourcing:

- compute the quotients q_i and modulus for each d_i
- encrypt the modulus and disperse it to the servers as shown in Table 6.

Table 6: Indexing scheme using a quotient of the outsourced data

Server 1		**Server 2**		**Server 3**		**Server 4**		**Server 5**	
Index	$E(I_i \bmod 101)$	Index	$E(I_i \bmod 103)$	Index	$E(I_i \bmod 107)$	Index	$E(I_i \bmod 109)$	Index	$E(I_i \bmod 111)$
3	#1anv,yuw	3	%1anghyuw	3	L?]ojub	3	Poknv$9	3	byyttt7
7	$89b5dja%	7	239b5dja%	7	P,mvs3#	7	Pomnbgfd3	7	Bbyughik
10	*s(1mfk3}	10	lkj6mfkdf	9	“;kubdu	9	“lkkmhgtui	9	hyt}x#4
27	=s(1gkk3}	26	hdgkdgdg	25	Pomngfa	2	(mkkjj”,?.	24	Buu46”p
90	[79b5sjq%	89	ty9fgsjqy	85	“lkjhf5	84	{dfrt5789	82	Yyggggg#
96	q@medh6	94	m.pmedhp	90	Piknhg6	89	q@medh6	87	Niiinioii*
101	s?dcms#th	99	=rtcmsm.,	96	‘p”hggff	94	,mngfswaw	92	yhyygg@g
132	@12hz+ef	129	asd2hzlm	124	Yunhzkjh	122	,kn gerxwxt	120	Ij%hhii”
173	Xbs&whw	170	SDbs&mne	163	ybsl./op	160	SF43455n	157	5nhjhdd
1125	Zxc!u!inm0	1103	RTN!u908u	1062	lkhu!ipoi	1042	Mhgfdc4k’	1023	Kuy6789

Retrieving:

To illustrate how this scheme can be used to retrieve outsourced data from the server, an example query to Table 6 for the salary $d_1 = 10$ is used.

- the client converts the query into an index corresponding to each server. Assume that out of the five servers only Server 1 ($m_1 = 101$), Server 2 ($m_2 = 103$) and Server 3 ($m_3 = 107$) respond to the request. The client therefore:
- retrieves the encrypted moduli and decrypts them,
- uses a constructive algorithm to compute $I_1 = \sum_{j=1}^{3} I_j'.e_j$ where e_1, e_2 *and* e_3 are obtained using the Extended Euclidean Algorithm such that:

  $gcd(m_1, (m_2 \times m_3)) = 4583(101) - 42(11021) \Rightarrow e_1 = -42(11021)$

  $gcd(m_2, (m_1 \times m_3)) = 1364(103) - 13(10807) \Rightarrow e_2 = -13(10807)$

  $gcd(m_3, (m_1 \times m_2)) = 4764(107) - 49(10403) \Rightarrow e_3 = -49(10403)$

  this yields $I_1 = 43.e_1 + 37.e_2 + 25.e_3 = -37845768$
- computes I_1 in $mod \prod_{j=1}^{3} m_j \Rightarrow I_1 = -37845768 \; mod \; (101 \times 103 \times 107) = 346$

- $I_1 = f(d_1) + \beta.p \Rightarrow f(d_1) = I_1 - \beta.p \Rightarrow f(d_1) = 346 - 3 \times 45 = 211$.
- solves the equation ($2d_1^2 + d_1 + 1 = 211$) to obtain $d_1 \Rightarrow d_1 = 10$ or $d_1 = -10.5$

Table 7 shows the same criteria as Table 3. It can be seen that the proposed scheme satisfies all criteria. It is more efficient compared to the previously reviewed ones in that it can jointly support equality and range queries, guarantees less burden on the client compared to some of previous approaches such as B+ tree indexing, produces no spurious tuples and also provides data availability and confidentiality.

Although IDA addresses many of the data outsourcing issues as shown in Table 3, however, the shares of IDA contain part of the original data that can be of use to a third party [33]. By snooping, an eavesdropper who obtains fewer than the threshold shares may reconstruct some part of the original data explicitly resulting in partial data leakage [56]. In contrast, no single share in the proposed approach can reveal information about the original data. Moreover, IDA places extra burden on the client side since it uses the B+ tree, which demands a lot of trips between the server and the client. This makes it less efficient compared to the proposed approach. Thus, the proposed Asmuth-Bloom secret sharing data outsourcing scheme provides more confidentiality and better performance when compared to IDA.

Table 7: Analysis parameters of the proposed scheme

	Equality Queries	**Range Queries**	**Client Burden**	**Spurious Tuples**	**Availability**	**Confidentiality**
Secure Data Outsourcing Scheme Based On Asmuth-Bloom Secret Sharing	Fully	Fully	Low	No	Yes	Yes

Security analysis

This section discusses different attack scenarios that can be launched successfully against some of the reviewed schemes. Then it shows how the proposed scheme is secured against such attacks.

Scenario 1:

An attacker succeeds in launching a Denial of Service attack against one or two of the servers involved in the outsourcing service. He/she cannot cripple the overall system, as only *r* out of the *n* servers are required to retrieve the outsourced data, and not all of them. Thus, the proposed scheme is secure against a DoS attack to a few of the servers and can maintain service availability.

Scenario 2:

An internal/external attacker might try to gain knowledge of the secrets (primes) by launching an inference attack, as the modulus values may expose the range of the secrets. However, the proposed scheme is secured against such attacks, since the modulus values are stored encrypted at the server side to preserve the confidentiality of the outsourced data.

Scenario 3:

An attacker succeeds in obtaining the order-preserved indexes of the outsourced data. Even so, he/she cannot gain any kind of knowledge about the original outsourced data since these indexes were obtained as a result of a function applied on the original attribute values. The coefficients of this function are shared secrets between the data owner and the client only. Thus, the proposed scheme preserves the order of the outsourced data without revealing information about the original data.

Scenario 4:

An adversary might try to obtain the indexes and the modulus values of different tuples from the same server, or the modulus values of the same tuple from different servers, in order to form linear equations that can be solved simultaneously so as to reveal the outsourced data. This kind of attack is not feasible in the proposed scheme as the modulus values are stored encrypted.

Scenario 5:

If an adversary, in scenario 4, has further succeeded in obtaining the unencrypted modulus values, he/she is still going to have more unknowns (the outsourced data and the scheme's primes) than the number of equations that he/she can form. Therefore, it will be computationally infeasible to get a deterministic solution.

Conclusion

Data outsourcing is a new paradigm that has gained a high popularity recently but it is hindered by some security challenges from gaining wide spread acceptance. The security challenges include: data confidentiality, availability and order preservation of the outsourced data. Data confidentiality is mostly achieved using a traditional encryption scheme, but traditional encryption is not good in preserving the order of the outsourced data, which is an important property that is needed in data outsourcing. Availability is another issue as the data loss can affect the business of the data owner. Data availability is usually achieved by duplicating the data and storing it at different servers.

This paper analyses some of the previous work in data outsourcing, and shows the advantages and limitations of each. It then proposes a secure data outsourcing scheme based on Asmuth-Bloom secret sharing, which uses congruence classes and the Chinese Remainder Theorem to solve the aforementioned issues. Firstly, we proposed a direct

implementation of this scheme that consists of three stages: setup, outsourcing and retrieving. In the setup stage the data owner chooses a set of prime integers known only to him/her and his/her client; in the data outsourcing stage the data owner outsources the encrypted attribute values and its modulus, where the modulus is used as an index. In the retrieving stage the client uses the index to retrieve the encrypted data and decrypt it at the client side. However, the direct implementation is expensive for the client as it produces spurious tuples. It is also vulnerable to statistical analysis attacks that can expose the used secret primes.

Finally, an enhanced scheme is proposed that also consists of three stages. In the setup stage, an increasing polynomial function is introduced to enlarge the attribute values. This allows larger co-prime integers to be selected so as to increase the modulus range before it wraps around, thus reducing the chance of an adversary obtaining the modulus value even in an unencrypted form. In the outsourcing stage, the data owner outsources the encrypted modulus and the quotient of the attribute values, to various servers for availability. The quotient is used as an index to keep the order of the outsourced encrypted modulus. To retrieve any value, the client retrieves the encrypted values, decrypts them and constructs the modulus using the CRT constructive algorithm.

The proposed scheme is computationally efficient as it uses simple modular arithmetic. It is more secure as the index is generated after wrapping the outsourced data by a polynomial function whose coefficients are kept secret. It also produces no spurious tuples and provides availability and confidentiality. In addition, it is an order-preserving scheme, which further reduces the burden of encryption and decryption of every query. Because of all its properties, the proposed scheme is strongly recommended as an efficient tool for real data outsourcing scenarios.

Acknowledgments

The authors would like to thank Prof. David W Chadwick from the University of Kent for proof reading this paper.

References

[1] Pope, J. A., Key, K. A., and Saigal, A. 2015. "Nonprofit Outsourcing Patterns: Why Don't Small NPOs Outsource More?." Journal of Nonprofit & Public Sector Marketing, 27(1): p. 99-116.

[2] Agrawal, D., El Abbadi, A., Emekci, F., and Metwally, 2009. "A. Database management as a service: Challenges and opportunities." Paper presented at the IEEE 25th International Conference on Data Engineering ICDE'09, Shanghai, March, 1709-1716.

[3] Li, Q., Wang, Z., Li, W., Li, J., Wang, C. and Du, R. 2013. "Applications integration in a hybrid cloud computing environment: modelling and platform." Enterprise Information Systems, 7(3): p. 237-271.

[4] Douglas, N. 2007. "Facebook Employees Know Whose Profiles You Look At." Valleywag, http://gawker.com/315901/facebook-employees-know-what-profiles-you-look-at.

[5] Lucas, M. M., and Borisov, N. 2008. "Flybynight: mitigating the privacy risks of social networking". Paper presented at the 7th ACM workshop on Privacy in the electronic society, Alexandria, Virginia, USA, October, 1-8.

[6] APUZZO, M. 2013. "What is the problem with prism?." http://news.yahoo.com/whats-problem-prism-203441280.html.

[7] Bajaj, K. 2014. "Cyberspace: Post-Snowden." Strategic Analysis, 38(4): p. 582-587.

[8] Damiani, E., Vimercati, S. D. C., Jajodia, S., Paraboschi, S. and Samarati, P. 2003. "Balancing confidentiality and efficiency in untrusted relational DBMSs." Paper presented at the 10th ACM conference on Computer and communications security, Washington, DC, October, 93-102.

[9] Rivest, R. L., Adleman, L., and Dertouzos, M. L. 1978. "On data banks and privacy homomorphisms." Foundations of secure computation, 4(11): p. 169-180.

[10] Nirmala, S. J., Bhanu, S. M. S., and Patel, A. A. 2012. "A Comparative study of the secret sharing algorithms for secure data in the cloud." International Journal on Cloud Computing: Services and Architecture (IJCCSA), 2(4): p. 63-71.

[11] Agrawal, R., Evfimievski, A., and Srikant, R. 2003, "Information sharing across private databases." Paper presented at the 2003 ACM SIGMOD international conference on Management of data, San Diego, June, 86-97.

[12] Samarati, P., and Vimercati, S. D. C. 2010. "Data protection in outsourcing scenarios: issues and directions." Paper presented at the 5th ACM Symposium on Information, Computer and Communications Security, Beijing, April, 1-14.

[13] Wang, C., Ren, K., Yu, S., and Urs, K. M. R. 2012. "Achieving usable and privacy-assured similarity search over outsourced cloud data". Paper presented at IEEE 2012 INFOCOM, March, 451-459.

[14] Selvam, L., Kumar, P. M., and Renjith, J. A. 2014. "Encryption-Based Secure Sharing of Data with Fine-Grained Access Control in Public Clouds." Journal of Applied Security Research, 9(2): p. 172-184.

[15] Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. 2011. "CryptDB: protecting confidentiality with encrypted query processing." Paper presented at the 23rd ACM Symposium on Operating Systems Principles, Cascais, Portugal, October, 85-100.

[16] Stratus Technologies. 2009. "Continuous Uptime for SQL Server: The best way to protect critical databases against downtime and loss.", http://www.platformmodernization.org/stratus/Lists/news/Attachments/1/Continuous-Uptime-for-SQL-Server.pdf.

[17] Bowers, K. D., Juels, A., and Oprea, A. 2009. "HAIL: a high-availability and integrity layer for cloud storage." Paper presented at the 16th ACM conference on Computer and communications security, Chicago, November, 187-198.

[18] Shamir, A. 1979. “How to share a secret.” Communications of the ACM, 22(11): p. 612-613.
[19] Stadler, M. 1996. “Publicly verifiable secret sharing.” Paper presented at the 15th annual international conference on Theory and application of cryptographic techniques - EUROCRYPT’96 – LNCS, Berlin, 190-199.
[20] Benaloh, J., and Leichter, J. 1990. “Generalized secret sharing and monotone functions.” Paper presented at the Advances in cryptology – LNCS, California, USA, 27-35.
[21] Chandramowliswaran, N., Srinivasan, S., and Muralikrishna, P. 2015. “Authenticated key distribution using given set of primes for secret sharing.” Systems Science & Control Engineering: An Open Access Journal, 3(1): p. 106-112.
[22] Tassa, T. 2007. “Hierarchical threshold secret sharing.” Journal of Cryptology, 20(2): p. 237-264
[23] Emekci, F., Agrawal, D., and El Abbadi, A. 2005. “Abacus: A distributed middleware for privacy preserving data sharing across private data warehouses.” Paper presented at the ACM/IFIP/USENIX 6th International Middleware Conference, France, November, 21-41.
[24] Hore, B., Mehrotra, S., and Tsudik, G. 2004. “A privacy-preserving index for range queries.” Paper presented at the Thirtieth international conference on Very large data bases, Canada, 720-731.
[25] Hacigümüş, H., Iyer, B., Li C., and Mehrotra, S. 2002. “Executing SQL over encrypted data in the database-service-provider model.” Paper presented at the 2002 ACM SIGMOD international conference on Management of data, Madison, Wisconsin, June, 216-227.
[26] Damiani, E., Vimercati, S. D. C., Foresti, S., Samarati, P., and Viviani, M. 2006. “Measuring inference exposure in outsourced encrypted databases.” Paper presented at Quality of Protection - Advances in Information Security, Springer, Feb, 185-195.
[27] Agrawal, R., Kiernan, J., Srikant, R. and Xu, Y. 2004. “Order preserving encryption for numeric data.” Paper presented at the 2004 ACM SIGMOD international conference on Management of data, Paris, 563-574.
[28] Devmane, M. A., and Rana, N. K. 2012. “Privacy Issues in Online Social Networks.” International Journal of Computer Applications, 41(13): p. 5-8.
[29] Gross, R., and Acquisti, A. 2005. “Information revelation and privacy in online social networks.” Paper presented at the 2005 ACM workshop on Privacy in the electronic society, Alexandria, 71-80.
[30] Luo, W., Xie, Q., and Hengartner, U. 2009. “Facecloak: An architecture for user privacy on social networking sites.” Paper presented at the International Conference on Computational Science and Engineering 2009 - CSE'09, Vancouver, BC, Aug. 26-33.
[31] Auwal, S. I., Faisal, S. I., Yusuf, I. M., Altun, H., Kaiiali, M., and Wazan, A. S. 2013. “Cloud-based online social network.” Paper presented at the 2013 International Conference on Electronics, Computer and Computation (ICECCO), Ankara, Turkey, November. 289-292.
[32] Asmuth, C., and John B. 1983. “A modular approach to key safeguarding.” IEEE transactions on information theory, 30(2): p. 208-210.
[33] Tse, D. W. K., Chen, D., Liu, Q., Wang, F., and Wei, Z. 2014. “Emerging Issues in Cloud Storage Security: Encryption, Key Management, Data Redundancy, Trust Mechanism.” Paper presented at the International Conference of Multidisciplinary Social Networks Research, Kaohsiung, Taiwan, September, 297-310.
[34] Foresti, S. 2010. “Preserving privacy in data outsourcing.” Springer Science & Business Media.
[35] Kaosar, M., and Quazi M. 2014. “Privacy-preserving interest group formation in online social networks (OSNs) using fully homomorphic encryption.” Journal of Information Privacy and Security, 10(1): 44-52.
[36] “Homomorphic Encryption”, https://en.wikipedia.org/wiki/Homomorphic_encryption.
[37] Poosala, V., Haas, P. J., Ioannidis, Y. E. and Shekita, E. J. 1996. “Improved histograms for selectivity estimation of range predicates.” Paper presented at ACM SIGMOD international conference on Management of data, Montreal, Quebec, Canada, June, 294-305.
[38] Mousavi, H., and Zaniolo, C. 2011. “Fast and accurate computation of equi-depth histograms over data streams.” Paper presented at the 14th International Conference on Extending Database Technology, Uppsala, Sweden, 69-80.
[39] B- tree, https://en.wikipedia.org/wiki/B-tree.
[40] Graefe, G. 2012. “A survey of B-tree logging and recovery techniques.” ACM Transactions on Database Systems (TODS), 37(1): 1-35.
[41] Stuntz, C. 2010. “What is Homomorphic Encryption, and Why Should I Care?”, http://blogs.teamb.com/craigstuntz/2010/04/08/38577.

[42] König, A. C., and Weikum, G. 1999. “Combining histograms and parametric curve fitting for feedback-driven query result-size estimation.” Paper presented at the 25th International Conference on Very Large Data Bases, San Francisco, CA, USA, 423-434.
[43] Wang, S., Agrawal, D., and El Abbadi, A. 2012. “Is Homomorphic Encryption the Holy Grail for Database Queries on Encrypted Data?” Technical report, University of California, USA.
[44] Liu, D., and Wang, S. 2012. “Programmable order-preserving secure index for encrypted database query.” Paper presented at the IEEE 5th International Conference on Cloud Computing (CLOUD), June, 502-509.
[45] Kadhem, H., Amagasa, T., and Kitagawa, H. 2010. “An encryption scheme to prevent statistical attacks in the das model.” Paper presented at DEIM Forum.
[46] Popa, R. A., Li, F. H., and Zeldovich, N. 2013. “An ideal-security protocol for order-preserving encoding.” Paper presented at IEEE Symposium on Security and Privacy (SP), May, 463-477.
[47] Mar, K. K. 2011. “Secured virtual diffused file system for the cloud.” Paper presented at International Conference for Internet Technology and Secured Transactions (ICITST), Dec, 116-121.
[48] Tech Target. 2010. “Information Dispersal Algorithms: Data-parsing for network security.”, http://searchnetworking.techtarget.com/Information-dispersal-algorithms-Data-parsing-for-network-security.
[49] Quisquater, M., Preneel, B., and Vandewalle, J. 2002. “On the security of the threshold scheme based on the Chinese remainder theorem.” Paper presented at Public Key Cryptography – LNCS, 199-210.
[50] Wang, S., Agrawal, D., and El Abbadi, A. 2011. “A comprehensive framework for secure query processing on relational data in the cloud.” Paper presented at Secure Data Management - LNCS, 52-69.
[51] Dragan, C. C., and Tiplea, F. L. “On the Asymptotic Idealness of the Asmuth-Bloom Threshold Secret Sharing Scheme.” International Journal on Designs, Codes and Cryptography, submitted for publication in 2013.
[52] Kuo, W. C., Fu, C. J., and Laih, C. S. 2006. “Design a Secure and Practical Metering Scheme.” Paper presented at the International Conference on Internet Computing, 443-447.
[53] Secret sharing using the Chinese remainder theorem, https://en.wikipedia.org/wiki/Secret_sharing_using_the_Chinese_remainder_theorem.
[54] MATHs is FUN advanced, “Polynomials: The Rule of Signs.”, http://www.mathsisfun.com/algebra/polynomials-rule-signs.html.
[55] Maohua, S., Shoushan, L., Lei, P., Zhe, J., and Yang, X. 2012. “Research on Secret Sharing and Privacy-preserving Secret Sharing with SMC.” International Journal of Advancements in Computing Technology, 4(4): 115-123.
[56] Li, M. 2013. “On the confidentiality of information dispersal algorithms and their erasure codes.” Journal of Computing Research Repository - arXiv:1206.4123.