Privacy-Preserving Fine-grained Correctness Verification of Query Results in Outsourced Databases

Mohammad Ali Hadavi, Rasool Jalili*, and Javad Ghareh Chamani

Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
Email: {mhadavi@ce., jalili@, gharehchamani@ce.}sharif.edu
**corresponding author: Tel: +98 21 66166617 Fax: +98 21 66166665 (Ext. 103)*

Adoption of data outsourcing to cloud servers is hindered by data integrity issues due to the lack of trust to the servers. Existing solutions to deal with the problem often require costly verification processes to build a verification object (VO) at the server side and to verify it at the client side, especially when the verification is to be performed at a low-level of granularity. This paper proposes an efficient and privacy-preserving solution, which verifies both the integrity and completeness of query results at the finest level of granularity — an individual attribute value. Outsourced data confidentiality is also preserved by securely dividing attribute values into several pieces. As a key novelty, our solution does not require building VOs at the server side for query results. Consequently, 1) no computation overhead is imposed on the server to construct VOs, 2) no change is required to the existing DBMS engines due to calculating VOs and accompanying them with the results, and 3) no information leakage occurs due to building or observing VOs by the server. Our theoretical and empirical analyses indicate the effectiveness of our solution compared to the existing solutions in terms of communication, query execution, and verification overheads.

Keywords: Data Outsourcing; Correctness Verification; Verification Object; Multi-Secret Sharing

1. INTRODUCTION

Cloud computing is supposed to play a pivotal role in IT architecture of future enterprises due to its irrefutable advantages such as on-demand self-service, ubiquitous network access, resource elasticity, and pay-per-use pricing. In this regard, outsourcing data to cloud servers has increasingly attracted attentions as it relieves data owners from the burden of data storage and management. In spite of its clear advantages, data outsourcing faces security challenges in untrustworthy environments. Preserving the integrity of outsourced data and assuring the correctness of query results are among the major challenges when data is stored and managed out of the physical control of its owner. It is possible for a cloud server to perform unfaithfully towards the data owner. For example, it can intentionally remove some rarely accessed data from the database, or conceal some inadvertent data corruptions to maintain its reputation. In both cases, incomplete results for some queries are returned without the data owner being informed. Even a faithful cloud server may be compromised by external attackers or infected by viruses, through which the integrity of the outsourced data and the completeness of query results may be violated.

The correctness is usually verified at the client side relying on Verification Objects (VO). VOs are constructed by the server and sent together with the results. Existing work on the topic propose authentication data structures such as Merkle Hash Tree (MHT) [1, 2], chained aggregated signatures [3], and skip lists [4, 5] to address different aspects of query verification, including the integrity, completeness, and freshness of the results. However, we have identified several limitations in the existing proposals.

First, they usually suffer from extra ordinary computation overheads in their client-side verification process, server-side proof construction, or performing data updates. Particularly, from the granularity viewpoint, fine grained verification, e.g., at the level of attribute value, considerably increases the computation and storage overheads. Tuple level verification, which has been proposed in the majority of the related works, imposes high communication overhead since the whole tuple must be returned in response to a query, even if

a single attribute is requested in the query.

Second, the existing solutions typically focus on the integrity of outsourced data and neglect the data privacy issues. The majority of existing solutions cannot be directly applied to the outsourced data when data privacy is also a concern in addition to correctness assurance. They usually reveal some amount of information to the servers. The ordering relation between outsourced values is an example of such revealed information. This leakage may open doors of inference over the outsourced data.

Third, for query verification, the existing DBMS engines should be improved so that a VO is calculated and accompanied with the query result. Indeed, this hinders the practicality of the existing solutions.

Considering the aforementioned limitations, this paper is aimed at addressing data integrity and query result correctness issues in outsourced databases using *Multi-Secret Sharing (MSS)* schemes. In MSS schemes more than one secret can be divided into the same set of shares through a single secret sharing process. Although the redundancy of shares in threshold (t, n) secret sharing schemes has been already proposed as the basis of query result verification [6], it is limited to the integrity of shares. Moreover, it is applicable when data shares are stored on non-communicating and consequently, non-colluding servers, which cannot be guaranteed in practice.

In this paper, we extend the idea of redundant shares and propose a solution to verify both the integrity and completeness of query results at the finest level of granularity based on multi secret sharing schemes. Using such schemes enables us to split attribute values along with their VOs into the same set of shares. That is, in addition to providing the confidentiality of data through splitting attribute values, correctness verifiability is also achieved without requiring separate VOs to be stored on the servers. Furthermore, it is possible to verify the results of projection queries without the need to retrieve whole tuples containing extra attribute values. We have implemented our solution and performed a series of experimental evaluations to assess its effectiveness in terms of client and server computation, as well as client-server communication overheads.

The rest of this paper is organized as follows. Section 2 introduces the general model of secret sharing based data outsourcing and describes some preliminaries, assumptions, and requirements of a query verification solution in such a model. Our solution is proposed in Section 3. Concentrating on active attacks, Section 4 theoretically analyzes the security of our solution. The results of our empirical study are reported in Section 5. Related works are reviewed in Section 6. The paper is finally summarized and concluded in Section 7.

2. PRELIMINARIES AND ASSUMPTIONS

In this section, we briefly review the use of secret sharing in the data outsourcing scenario. Considering such a scenario, we then express our assumptions of data and threat models as well as the requirements of a typical solution for query result verification.

2.1. Secret Sharing and Data Outsourcing

Secret sharing schemes refer to methods through which a secret is divided into shares. The shares are then distributed among a group of participants. The secret can be reconstructed when authorized subsets of participants pool their shares. Shamir [7] proposed a threshold (t, n) secret sharing scheme in which a subset A of n participants can reconstruct the secret if $|A| \geq t$.

The notion of secret sharing has attracted attentions to provide a privacy preserving data outsourcing solution [8, 9, 10, 11, 12, 13, 14, 15]. In a typical (t, n) secret sharing based data outsourcing solution, the data owner becomes the distributor of attribute values among n servers $S_1, S_2, \ldots, S_n$. Each server S_i stores a share $share_i(v)$ of an attribute value v $(1 \leq i \leq n)$. $share_i(v)$ is the result of computing $f(x) = a_{t-1}x^{t-1} + a_{t-2}x^{t-2} + \ldots + a_1x + a_0$ on the input x_i where $a_0 = v$ and $a_1, \ldots, a_{t-1}$ are chosen randomly by the data owner. The input values $x_1, x_2, \ldots, x_n$, which are known to the owner (and also to the authorized users), form an n-ary key vector $X = (x_1, x_2, \ldots, x_n)$ upon which attribute value are spilt and reconstructed.

When a user who knows X, receives at least t shares of a secret v, the secret can be reconstructed because t points are enough to define a $t-1$ degree polynomial.

Agrawal *et al.* [12, 13], Hadavi *et al.* [11, 9], Tian *et al.* [15], and Emekci *et al.* [6] customized Shamir's scheme and proposed some searchable secret sharing schemes in which users' queries can be transformed into queries over randomly generated shares stored on the servers. The transformation technique depends on the method of dividing secrets into their shares. Figure 1 depicts sharing a sample relation $Employee(ID, Name, Age, Salary)$ among n data servers.

In regard to the integrity of outsourced data, redundant shares in a threshold (t, n) secret sharing can be utilized as a potential source of verifying the integrity of shares, stored on the untrusted servers. This general idea has previously been taken into consideration in [6, 16] relying on comparing the results of extra interpolations over different t-ary sets of shares. When secrets reconstructed over different t-out-of-n groups of shares are not equal, there is at least a manipulated share on the servers. However, this simple idea has some limitations for query result verification. First, it is limited to share integrity and cannot verify the completeness of query results. Second, it assumes non-communicating (non-colluding) servers. If the servers can communicate, they can collude and manipulate their shares such that a fixed wrong secret is

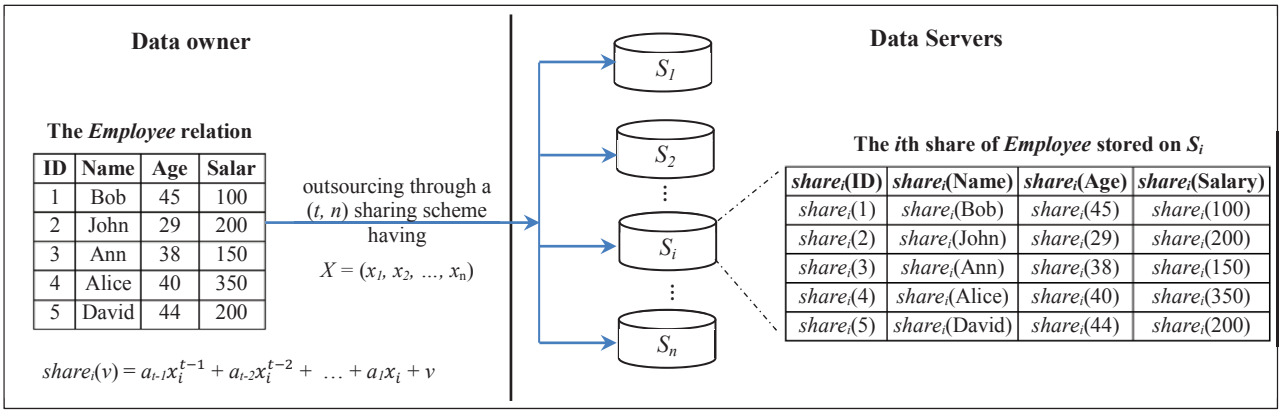

FIGURE 1. The secret sharing based data outsourcing model

reconstructed by different interpolations. For instance, if all servers add a constant value c to their shares, the wrong secret $s + c$ is interpolated instead of the real secret s and the cheating remains undetected. The servers can also agree on removing a specific tuple from a query result without being detected. A problem with the assumption of non-communicating servers, in addition to have it unguaranteed in practice, is that it enforces separate cloud providers be chosen for outsourced data management. Whereas it can be possible to store all data splits on a single cloud server.

Considering the above limitations and also those identified in Section 1, this paper is aimed at proposing a solution for the verification of both integrity and completeness of query results when outsourced data confidentiality is preserved through data splitting by Shamir based secret sharing schemes. In the proposed solution, attribute values as secrets are split into several pieces so that the integrity and completeness of query results can be verified, after secret reconstruction.

2.2. Assumptions and Threat Model

We assume relational data is outsourced by a data owner either to a single server or to multi servers. The server(s) is/are not trusted neither for data confidentiality nor for correctness of query results. That is, the servers are curious to obtain confidential information about the outsourced data and also may manipulate query results. While the servers might have *a priori* knowledge about the outsourced data, they are not aware of the query workload and the database access pattern.

The owner as well as authorized users can submit queries via a query interface to access and retrieve the outsourced data. We assume that users have already passed an authentication step and are authorized to submit queries to the system as well as the data owner. We consider selection and projection queries with equality or range conditions over searchable attributes of the outsourced relation. To this purpose, the values of searchable attributes are supposed to be split and shared through a searchable secret sharing scheme [6, 9, 11, 13, 15]. Each query, according to its conditions, returns a set of field values as the query result for which the querier user must be able to verify its integrity and completeness. We assume numeric values for attributes while the approach can immediately be extended to other data types, including character data over which exact match queries are executed.

2.3. Solution Requirements

Security and practicality issues are two sides of the same coin. From the security viewpoint, data confidentiality must be preserved and unauthorized manipulations of query results must be detected when the cloud server is trusted neither for query correctness nor for data confidentiality. In other words, a solution for query result verification should be applicable when data confidentiality must be preserved. For example, the servers must be able to construct the proof of query results having no information about the actual outsourced data, e.g., the ordering relation. Further, the verification object itself must not reveal any information about the outsourced data such as the original data distribution.

From the practicality point of view, solutions are preferred that, besides adequate query support, adopt existing DBMSs without changes into their engines. Moreover, the efficiency of query execution and verification and also the efficiency of data updates are major concerns. The size of VOs, which affects the communication cost, should depend on the size of the query result rather than the size of the outsourced database. From the computational perspective as well, the client-side verification cost should be proportional to the result size not to the database size. Moreover, the granularity of verification is important as it has a direct impact on both communication and processing overheads. For instance, correctness verification at tuple level implies that entire tuple should be returned

even if the user requests just one attribute of a relation through a projection query.

With respect to the above requirements, we propose our solution in the next section and analyze it further, theoretically and empirically, in Sections 4 and 5.

3. OUR PROPOSED SOLUTION

In Shamir's (t, n) secret sharing scheme, $t-1$ shares do not reveal any information about the secret. In other words, among t shares through which a secret is reconstructed, $t-1$ shares can be random elements. We suggest replacing these random elements with the information, which can be further used for query result verification. This can be considered as a Multi Secret Sharing (MSS) scheme in which more than one secret can be split during a single secret sharing process [17]. In general, consider a set of secrets $s_1, \ldots, s_m$ are to share among a set of participants $U = \{u_1, \ldots, u_n\}$ through a single sharing process. In a threshold MSS scheme (m, t, n), any subset A of participants, has no information about the secret s_j $(1 \leq j \leq m)$ if $|A| < t$ and all participants in $A \subseteq U$ can reconstruct all secrets s_j $(1 \leq j \leq m)$ if $|A| \geq t$.

So far, various MSS schemes have been proposed with different performances in terms of their required storage for public information, the number of secrets to share per each sharing process, efficiency of sharing and reconstruction processes, and so on [18, 19, 20, 21, 22]. Considering such parameters and especially its efficient sharing and reconstruction processes, we recognized Yang *et al.*'s scheme [18] appropriate for our purpose and customized it to provide the verifiability of query results in the secret sharing based data outsourcing scenario. However, it might be possible to use other MSS schemes or design new ones in order to have a more efficient verification of query results.

In the following subsections, we show how to use an MSS scheme for query result verification in the data outsourcing scenario.

3.1. MSS Based Query Result Verification

Consider a relation r with schema $R(attr_1, \ldots, attr_k)$ for which the correctness of query results for some attributes is expected. According to the general model of secret sharing based data outsourcing (See Section 2.1), r is split into n relations with schema $R_i(share_i(attr_1), \ldots, share_i(attr_k))$ for $1 \leq i \leq n$ through a secret sharing scheme where the value $share_i(a)$ from the relation r_i indicates the ith share of the value a in the original relation r (Figure 1).

All values of verifiable attributes in r must be split into n shares such that the correctness verification is provided for the querier clients. For the sake of simplicity, we assume that there is an attribute *attr* in the relation schema whose values are returned as query results. Queries are posed over the values of a searchable attribute $attr'$. Let Dom_{attr} be the domain of the attribute *attr*. Disregarding attributes other than *attr* and $attr'$ in the relation schema except a $TupleID$, which uniquely identifies database tuples, a database relation is modeled by $r = \{(id, v, v') | id \in Dom_{TupleID}, v \in Dom_{attr}, v' \in Dom_{attr'}\}$. A query Q selects *attr* values of r if their corresponding $attr'$ values fall in the range $[l..u]$. We model the query by $Q_r(attr, attr'[l..u])$ and the query result by $Res_Q = \{(id, v) | (id, v, v') \in r$ and $l \leq v' \leq u\}$. While the actual result of the query contains v values, we add their corresponding $TupleID$s into the result to distinguish extra (repetitive) v values. We can ignore having such id values in the result if *attr* values be unique in the relation. The correctness of Res_Q is the result of its precision and recall with respect to the query Q_r and is formally defined as below:

Definition 1- Res_Q is correct with respect to $Q_r(attr, attr'[l..u])$ if the three following conditions hold:

1. $\forall (id, v) \in Res_Q \ : \exists (id, v, v') \in r, l \leq v' \leq u$
2. $\forall (id, v, v') \in r \ : l \leq v' \leq u \Rightarrow (id, v) \in Res_Q$
3. $\forall (id, v_1), (id', v_2) \in Res_Q \ : v_1 = v_2 \Rightarrow id \neq id'$

□

The first condition guarantees that the result is precise. That is, all return values a are authentic with respect to Q_r in the sense that they satisfy the query condition. The second condition guarantees the completeness with respect to Q_r, i.e., no value is removed from the result while it satisfies query condition. The third condition denotes that no repetitive value is permitted in the result.

To verify the above conditions, we propose using an MSS scheme such that an attribute value, as the main secret, together with its VO are simultaneously divided into several pieces. While reconstructing the attribute value, its VO is also reconstructed.

For the verification purpose, the VO must reveal the satisfaction of the query condition for each member of the result. Moreover, it must show that a value is neither extra in nor removed from the result set. Therefore, the VO of a verifiable value v, which appears in a tuple $(id, v, v') \in r$, is a triple $VO_v = (v', id_{prv}, id_{nxt})$ where id_{prv} and id_{nxt} denote $TupleID$ values of respectively the predecessor and successor tuples with respect to the searchable dimension. While reconstructing secret value v, we check whether the query condition is truly satisfied by simultaneous reconstruction of v'. Moreover, the ordering information id, id_{prv}, and id_{nxt} let us verify whether the chain of values in the result set is a valid fragment of the original chain of values, outsourced previously by the data owner. Table 1 shows an example of VOs associated with *Salary* values from the *Employee* relation. In the table, *Salary* and *Age* are supposed to be the verifiable and searchable attributes, respectively. The values of searchable attributes are split through a searchable secret sharing scheme [6, 9, 13, 15]. Attributes of a relation which are neither

TABLE 1. Values of *Employee* relation (Figure 1), *Age* is searchable and *Salary* is verifiable

Id	*Age*	*Salary* (to be shared)		VO of *Salary* values
1	45	100	$\longrightarrow$	(45, 5, $+\infty$)
2	29	200	$\longrightarrow$	(29, $-\infty$, 3)
3	38	150	$\longrightarrow$	(38, 2, 4)
4	40	350	$\longrightarrow$	(40, 3, 5)
5	44	200	$\longrightarrow$	(44, 4, 1)

searchable nor verifiable can be shared through the basic scheme of Shamir.

We need an MSS scheme (m, t, n) through which four secrets, an attribute value plus its triple VO, are split into n pieces. Therefore, a (4, t, n) MSS scheme where $t \geq 4$ and $n \geq t$ is required to verify the correctness of query results. The next subsections describe such a (4, t, n) MSS scheme.

3.1.1. System Initialization

The owner specifies attributes of a relation schema whose values in query results require verification. A searchable attribute is also specified by the owner as the searchable dimension. We require splitting four secrets by a single process. To this end, we use a (4, t, n) MSS scheme as a relation $\mathcal{S} : V^4 \times \mathcal{X} \rightarrow V^n$, which maps a quaternary set of values, a secret plus its triple VO, onto n shares, having an n-ary vector $X \in \mathcal{X}$ as the private key.

To reconstruct each quaternary set of secrets, a set of at least t shares are required. Secret reconstruction can be considered as a function $\mathcal{R} : V^t \times \mathcal{X} \rightarrow V^4$, which outputs a secret value and its corresponding VOs by inputting t shares of the value, having the key vector $X \in \mathcal{X}$. In a (4, t, n) MSS scheme, the splitting and reconstruction keys are the same.

To use a (4, t, n) MSS scheme, the data owner initially determines the system parameters including t, n (where $t \geq 4$ and $n \geq t$), a large prime P regarding that all calculations are performed in GF(P), and the n-ary key vector X containing n distinct random integers x_1, x_2, ..., x_n in GF(P). x_is, usually known as secret shadows in MSS schemes, are kept private from the untrusted servers. We assume that only authorized clients, in addition to the data owner, have access to the key vector in order to use it for secret reconstruction.

Finally, the data owner sorts database tuples with respect to the searchable dimension to build the VO corresponding to each attribute value, similar to the one shown in Table 1. Now, attribute values and their corresponding VOs are ready to split using our (4, t, n) MSS scheme.

3.1.2. Secret Splitting

Now, we describe the relation $\mathcal{S} : V^4 \times \mathcal{X} \rightarrow V^n$, which splits an attribute value v with its $VO_v = (v', id_{prv}, id_{nxt})$ into n shares.

For each attribute value, the owner constructs a $t-1$ degree polynomial $f(x) = id_{nxt} + id_{prv}.x + v'.x^2 + v.x^3 + a_1.x^4 + \ldots + a_{t-4}.x^{t-1} \mod P$ where a_1, ..., a_{t-4} are random numbers from GF(P). Shares of v are then calculated by $share_i(v) = f(x_i)$ for $1 \leq i \leq n$ and stored on the server.

Example 1- We want to share *Salary* values of the *Employee* relation (Figure 1) in a verifiable manner. Consider the *Salary* value 150 and its VO_{150} = (38, 2, 4) in Table 1 (the third tuple of the relation). Assume that for a (4, 5, 5) MSS scheme, P = 1009, X = (x_1 = 301, x_2 = 849, x_3 = 16, x_4 = 11, and x_5 = 90). To split v = 150 into five shares we need a random value in GF(1009) such as 511 as the coefficient of x_4. It is worth mentioning that P = 1009 means that no value in the domain of attributes (*Salary* and $TupleID$ in this example) are no more than 1009. Moreover, the random coefficient of x_4 must be generated per attribute value. Now, we construct the polynomial $f(x)$ and compute five share values of 150.

$f(x) = 4 + 2x + 38x^2 + 150x^3 + 511x^4 \mod 1009$
$share_1(150) = f(301) = 699$,
$share_2(150) = f(849) = 493$,
$share_3(150) = f(16) = 909$,
$share_4(150) = f(11) = 73$, and
$share_5(150) = f(90) = 953$

□

3.1.3. Secret Reconstruction

The function $\mathcal{R} : V^t \times \mathcal{X} \rightarrow V^4$ reconstructs a secret value and its corresponding VO, having the private key X plus t distinct shares. The owner as well as authorized users who know X can uniquely reconstruct the $t-1$ degree polynomial $f(x)$ using the Lagrange interpolation in (1), provided that at least t shares of a secret value v are returned back by the server. In (1), $share_1(v)$, ..., $share_t(v)$ are the t returned shares from the server.

$$f(x) = \sum_{i=1}^{t} share_i(v) \prod_{j=1, j\neq i}^{t} \frac{x - x_j}{x_i - x_j} \mod P =$$
$$p_1 + p_2 x + p_3 x^2 + p_4 x^3 + a_1 x^4 + \cdots + a_{t-4} x^{t-1} \mod P \quad (1)$$

Having the polynomial $f(x)$, the secret v and its VO_v = (v', id_{prv}, id_{nxt}) can be obtained as follows:
$p_1 = id_{nxt}$, $p_2 = id_{prv}$, $p_3 = v'$, and $p_4 = v$.

Example 2- Four share values are enough to reconstruct the *Salary* value 150 and its corresponding VO in Example 1. Consider four shares $share_2(150)$ = 493, $share_3(150)$ = 909, $share_4(150)$ = 73, and $share_5(150)$ = 953 as the input of secret reconstruction process, having the reconstruction key X = (301, 849,

16, 11, 90). Putting the given shares and x_is in (1), $f(x) = p_1 + p_2x + p_3x^2 + p_4x^3 + a_1x^4$ is interpolated so the secret value and its VO are reconstructed in parallel. That it, no extra computation is required to compute verification information. The result of interpolation outputs $p_1 = 4$, $p_2 = 2$, $p_3 = 38$, $p_4 = 150$, and $a_1 = 511$ which in turn results in $v = 150$ and $VO_v = (38, 2, 4)$. □

Having VO of each reconstructed secret value, the client can verify the integrity and completeness of the query result.

3.1.4. Result Verification

For a query $Q_r(attr, attr'[l..u])$, the actual result consists of pairs (id, v) where $id \in Dom_{TupleID}$ and $v \in Dom_{attr}$. For each value v of the result, its constituent shares must be returned from the server. After the reconstruction process for all secrets, the client obtains a set of secret values v_1 to v_k and their corresponding verification information $VO_{v_1}= (v'_1, id_{prv_1}, id_{nxt_1})$ to $VO_{v_k}= (v'_k, id_{prv_k}, id_{nxt_k})$. Having such information, the querier user performs the following actions to verify the result:

1. Checks that for each v_i $(1 \leq i \leq k)$, its corresponding v'_i satisfies the query condition, i.e., $l \leq v'_i \leq u$.
2. Sorts returned VOs based on v'. Assuming the set of triples $(v'_1, id_{prv_1}, id_{nxt_1})$, $(v'_2, id_{prv_2}, id_{nxt_2})$, $\ldots$, $(v'_k, id_{prv_k}, id_{nxt_k})$ is a sorted list of VOs associated with the result set (id_1, v_1), (id_2, v_2), $\ldots$, (id_k, v_k), then it is necessary to have the following conditions for the result completeness:
   - $v'_1 \leq v'_2 \leq \cdots \leq v'_{k-1} \leq v'_k$
   - $\forall\ i, 1 \leq i \leq k-1 : id_{i+1} = id_{nxt_i}\ ,\quad id_i = id_{prv_{i+1}}$

The second condition says that the original chain of outsourced values has not been broken, neither in forward nor in backward paths by adding fake values to or removing legitimate values from the result set.

Completeness verification in result boundaries: Having the above verification process, the malicious server can remove legitimate values from the lower and upper borders of the result set. This is not always a possible manipulation because the server does not know about the ordering of shares and consequently, cannot find the borders of the result set. However, it is plausible for small sets of results, e.g., below five elements in the result, that the server tries removing a share, which is either an upper or a lower border of the result set. In this situation, we require extra process to guarantee the completeness. To this purpose, the querier user requests a query $Q_r(attr, TupleID = \{id_{prv_1}, id_{nxt_k}\})$ to retrieve share values from the servers corresponding to the predecessor of the first and the successor of the last element in the result. Having these shares, the querier can verify that whether a value is removed from the upper or lower borders of the result.

Algorithm 1 shows the pseudo code of query result verification which outputs *true* provided that it verifies the correctness of the result.

Remark 1. One may claim that *cheating-immune* secret sharing schemes are more appropriate for query result verification in secret sharing based scenarios of data outsourcing compared to our MSS based solution. A secret sharing scheme is *cheating-immune* if cheater participants of the scheme do not have more advantage than honest participants on submitting incorrect shares in determining the true secret [23]. Although these schemes have been devised to detect integrity violations of shares, they cannot detect completeness violations in query results. Furthermore, they are more computationally demanding in secret splitting as well as in secret reconstruction, nevertheless, cannot be perfect schemes [23]. We refer readers to [23] for more information about properties and constraints of this family of secret sharing schemes.

3.2. Data Update

Our solution supports efficient data updates, including DELETE, UPDATE, and INSERT operations. Typically, in addition to the inserted/updated tuple, the VOs of affected elements in the chain of values are also updated in these operations. The following specific procedures are followed for each operation.

INSERT: To insert a new tuple t_{new} into the relation r, the value of searchable attribute is shared through a searchable secret sharing scheme. The following 7-step procedure is performed by the owner (or an authorized user) to split the verifiable attribute values of t_{new}.

Step 1- Share values belonging to the previous tuple t_{prv} and the next tuple t_{nxt} are retrieved through a selection query.

Step 2- Attribute values, pertaining to t_{nxt} and t_{prv}, and their corresponding VOs are reconstructed.

Step 3- The correctness of reconstructed values is verified.

Step 4- Considering the new tuple t_{new}, the VO of verifiable values in t_{nxt} and t_{prv} is updated.

Step 5- The VO of the verifiable value in t_{new} is built.

Step 6- Verifiable values in t_{nxt} and t_{prv} regarding their new VOs are re-shared through our MSS scheme and updated in the database.

Step 7- The verifiable values in t_{new} are split regarding their VOs and inserted into the database.

As discussed in Section 3.1.2, building VO of a verifiable value is a prerequisite to split the value into its shares. To this aim, the owner is involved in an interaction with the server to retrieve share values of t_{new}'s predecessor and successor tuples with respect to the searchable attribute (*Step 1*). Then, attribute values and their VOs are reconstructed from their shares (*Step 2*) and the verification process is performed for assuring correctness of the retrieved values (*Step 3*). Similar to inserting a new element into a doubly-linked

Algorithm 1 Verification process

Inputs:

r // the relation, $attr$: verifiable, $attr'$ searchable
l, u // lower and upper bounds of query condition
$Res[k, 2]$ // array of the result containing k pairs (id, v)
$VO[k, 3]$ // array of VOs containing k triples (v', id_{prv}, id_{nxt})
X // key vector
t, n // parameters of a (4, t, n) MSS scheme
$\mathcal{R}$ // reconstruction function

Outputs:

true OR **false** // whether the result is verified or not

Variables:

int: $VO_{srt}[k, 3], v, VO_v[3], a[t]$

Initialization:

$VO_{srt} = \text{Sort}(VO)$ // sorts VO triples w.r.t $attr'$

Verification:

/* checking query satisfaction */
for $i = 1$ **to** k **do**
  if $(l > VO_{srt}[i, 1]$ OR $VO_{srt}[i, 1] > u)$ **then**
    return false
  end if
end for
/* checking backward and forward links in the chain */
for $i = 2$ **to** $k - 1$ **do**
  if $(VO_{srt}[i, 2] \neq Res[i - 1, 1])$ **then**
    return false
  end if
  if $(VO_{srt}[i, 3] \neq Res[i + 1, 1])$ **then**
    return false
  end if
end for
/* checking the first element of the chain */
if $(VO_{srt}[1, 3] \neq Res[2, 1])$ **then**
  return false
end if
/* checking the last element of the chain */
if $(VO_{srt}[k, 2] \neq Res[k - 1, 1])$ **then**
  return false
end if
/* retrieving t shares pertaining to the lower bound value */
for $i = 1$ **to** t **do**
  $a[i] =$ SELECT $share_i(attr)$ FROM r_i WHERE $TupleID = VO_{srt}[1, 2]$
end for
/* reconstructing the lower bound value and its VO */
$(v, VO_v) = \mathcal{R}(\{a[1], \ldots, a[t]\}, X)$
/* checking the completeness at lower bound */
if $(l \leq VO_v[1] \leq u$ OR $VO_v[3] \neq Res[1, 1])$ **then**
  return false
end if
/* retrieving t shares pertaining to the upper bound value */
for $i = 1$ **to** t **do**
  $a[i] =$ SELECT $share_i(attr)$ FROM r_i WHERE $TupleID = VO_{srt}[k, 3]$
end for
/* reconstructing the upper bound value and its VO */
$(v, VO_v) = \mathcal{R}(\{a[1], \ldots, a[t]\}, X)$
/* checking the completeness at upper bound */
if $(l \leq VO_v[1] \leq u$ OR $VO_v[2] \neq Res[k, 1])$ **then**
  return false
end if
return true

list, share values of the verifiable attribute for t_{new}'s predecessor and successor tuples is recomputed such that the chain of values includes the new inserted value (*Step 4*). Subsequently, the VO of the verifiable value pertaining to t_{new} is built with respect to its retrieved adjacent tuples (*Step 5*). Having new VOs, share values of t_{nxt} and t_{prv} are then computed and stored on the server by an UPDATE operation (*Step 6*). Finally, share values of t_{new} are computed and sent to the server(s) by an INSERT command (*Step 7*).

DELETE: Performing a DELETE operation, which removes a tuple t from the relation, is similar to the INSERT in the sense that the chain of verifiable values is modified regarding the removed tuple. The following 5-step procedure describes the process of a DELETE operation issued by the data owner or an authorized client.

Step 1- Share values belonging to t's previous tuple t_{prv} and t's next tuple t_{nxt} are retrieved through a selection query.

Step 2- Attribute values of t_{nxt} and t_{prv} and their corresponding VOs are reconstructed.

Step 3- The reconstructed values are verified.

Step 4- The VO of verifiable values in t_{nxt} and t_{prv} are updated after deleting t.

Step 5- Verifiable values of t_{nxt} and t_{prv} considering their new VOs are re-shared and updated in the database.

The first three steps of the above procedure are similar to the INSERT case. In the fourth step, the VO of verifiable attribute values in t_{nxt} and t_{prv} are updated so that t be removed from the chain of values, similar to removing an element from a doubly-linked list. Finally, the values of t_{nxt} and t_{prv} are re-shared and sent to the server by an update query, updating shares of the verifiable attribute values (*Step 5*).

UPDATE: The protocol of performing an update operation of the form "UPDATE r SET $attr = a$ WHERE $attr' = b$" depends on whether the updated attribute $attr$ in r is searchable or verifiable. The simplest case is when it is neither searchable nor verifiable. In this case, affected tuples based on the update condition are sent back to the user who has issued the update. For each affected tuple, shares of the new value are computed according to the utilized secret sharing scheme. An update operation over the server-side relations, storing share values, is posed to modify share values with respect to the update condition.

There is almost the same procedure when the updated attribute $attr$ is a verifiable attribute. Again, affected tuples based on the update condition are sent back to the owner. Then, for each tuple, new shares of the verifiable attribute value are computed through our MSS scheme. Finally, new share values of the verifiable attribute are modified on the servers through an update query over the server-side relation of shares. In this case, although verifiable attribute is updated, the VOs are fixed as there is no change to the values of

TABLE 2. Values of the *Employee* relation (Figure 1), *Age* and *Salary* are searchable and *Name* is verifiable

Id	*Age*	*Salary*	*Name*	VO of *Name* values
1	45	100	Bob →	(45, 5,+∞),(100,-∞,3)
2	29	200	John →	(29,−∞, 3),(200, 3, 5)
3	38	150	Ann →	(38, 2, 4),(150, 1, 2)
4	40	350	Alice →	(40, 3, 5),(350, 5,+∞)
5	44	200	David →	(44, 4, 1),(200, 2, 4)

searchable attributes. Therefore, verifiable values are updated while the chain of values remains unchanged.

When the update operation updates the value of a searchable attribute, it affects the chain of verifiable values. This case is more complicated but can be considered as a combination of DELETE and INSERT operations.

3.3. The Case of More Searchable Attributes

In Section 3.1 we showed how to use an MSS scheme in order to verify the correctness of query results through checking the chain of returned values. For the sake of simplicity, however, we assumed one searchable dimension. In this section, we go through a more realistic assumption and show that how the core idea is extended to support more searchable attributes. One idea is to include the information of all searchable dimensions into the VO of outsourced values.

Example 3- Consider the *Employee* relation in Figure 1 and assume that both *Age* and *Salary* are searchable dimensions for the verifiable attribute Name. Table 2 shows the verification information corresponding to each *Name* value. In Table 2, the last column consists of six pieces of information divided into two groups of triples, each of which associated to a searchable dimension.

□

In a (p, t, n) MSS scheme, this simple extension implies that the number of hidden secrets p, used as VO, increases proportional to $3 \times N_{sa}$ when N_{sa} is the number of searchable attributes. Having the constraint of $p \leq t$ in our approach, increasing p raises the amount of t, which in turn has a direct impact on the server side storage and server-client communication costs. Consider a relation having eight searchable attributes. To share a verifiable attribute using our (p, t, n) MSS scheme, we need $p = 8 \times 3 + 1 = 25$ values (one secret plus 24 values as VO associated to eight searchable dimensions) to be shared through a single splitting process. That is, each attribute value should be split into at least 25 shares so it imposes a considerable communication cost (due to transferring 25 shares for reconstructing a value) in addition to storage cost on the server.

While the above extension can be acceptable just for few searchable attributes, another extension is to keep $p = 4$ as in the case of one searchable attribute, and re-share each verifiable attribute per searchable dimension.

Example 4 - Consider again the *Employee* relation in Figure 1 and assume that both *Age* and *Salary* are searchable dimensions for the verifiable attribute *Name*. Figure 2 depicts the ith share of the *Employee* relation, having two searchable attributes *Age* and *Salary* and one verifiable attribute *Name*. In the figure, there exists two columns for the ith share of the *Employee* relation; $share_i(Name_{Age})$, which is the ith share of *Name* values spilt with respect to their VO built upon the searchable attribute *Age*, and $share_i(Name_{Salary})$, which is the ith share of *Name* values spilt with respect to their VO built upon the searchable attribute *Salary*. Figure 2 also shows that these two sets of shares are generated through different polynomials whose coefficients are assigned from separate VOs built upon different searchable dimensions. *Age* and *Salary* values are also split through a secure searchable secret sharing scheme [6, 9].

When a query such as "SELECT *Name* FROM *Employee* WHERE $36 < Age < 45$" is posed by the user, the client translates it into "SELECT $share_{Age}(Name)$ FROM *Employee* WHERE $\alpha < share_i(Age) < \beta$" where α and β are the corresponding ith shares of 36 and 46, respectively. That is, the server returns satisfying shares of *Name* split based on the searchable attribute *Age*. For the case of "SELECT *Name* FROM *Employee* WHERE $180 < Salary < 280$", the client translates the query such that the values of the second *Name* column at the server side, i.e., $Name_{Salary}$, whose values have been split regarding the searchable attribute *Salary*, are returned by the server. □

This extension while imposes a limited storage overhead on the server, keeps the computation and communication cost equal to the efficient case of single searchable dimension.

4. SECURITY ANALYSIS

In this section, we analyze the robustness of our solution against attacks on the integrity and completeness of query results as well as attacks on the confidentiality of the outsourced data.

In general, security of our solution against curious and malicious servers is based on the security of the base MSS scheme [18, 21]. A curious server having access to data shares obtains confidential attribute values condition to the ability of accomplishing the reconstruction function $\mathcal{R} : V^t \times \mathcal{X} \rightarrow V^4$ without accessing the key vector $X \in \mathcal{X}$. This ability is equivalent to finding out the actual solution for a set of simultaneous equations where the number of unknown variables is more than the number of equations. On the other hand, a malicious server can scape detecting its unauthorized manipulations of query results condition

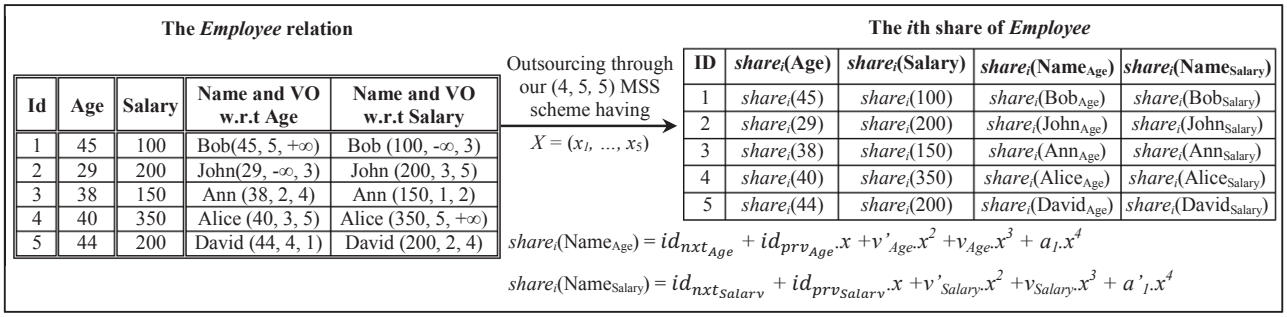

FIGURE 2. Splitting the verifiable attribute value $Name$ in a relation having two searchable dimensions

to the ability of accomplishing the splitting relation $\mathcal{S} : V^4 \times \mathcal{X} \rightarrow V^n$ without accessing the key vector $X \in \mathcal{X}$. Such ability is equal to choosing a correct set of shares among $|P|^n$ possible sets of shares which is a negligible probability.

4.1. Attacks on Query Result Correctness

Inadvertent data loss for any reason, e.g., due to a virus attack, which may occur for apparently honest cloud servers affirms that assuring the correctness of query results is a real requirement when both computation and storage are supposed to be delegated to the cloud. In this regard, unauthorized manipulations of data shares, including changes, insertions, and removals when either storing on the server or transferring over the network may bring about incorrect query results, which must be detected at the client side using the verification process.

The general idea of using redundant shares and extra interpolations in a threshold $(t,\ n)$ secret sharing has already been considered for the integrity verification of shares [6, 24] when the servers are assumed non-colluding. This assumption, which needs to guarantee outsourcing data to non-communicating servers, limits the practicality issues of secret sharing based data outsourcing. Consider a situation in which all participants, the servers in the data outsourcing scenario, add a fixed number to their shares or decide to remove a specific tuple from their results. Both manipulations remain undetected by the clients when the verification process is solely based on redundant shares and extra interpolations in a (t, n) secret sharing scheme. We show that our solution is secure against active attacks while data shares can be stored either on a single server or on separate but communicating servers which may collude to escape from the detection of a correctness violation. Considering Definition 1 in Section 3.1, we prove that our verification process is able to detect:

1. if query conditions have been truly satisfied for the returned share values.
2. if returned shares are subjects of any changes.
3. if returned shares are removed from or inserted into the returned set of values.

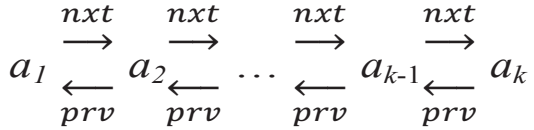

FIGURE 3. The chain of values in the result set

Consider the query "SELECT $attr$ FROM r WHERE $l \leq attr' \leq u$" posed by a user to the system. The query is then translated into "SELECT $ID, share_i(attr)$ FROM r_i WHERE $l' \leq share_i(attr') \leq u'$" and sent to the ith share of r. Assume that the actual result set of the query is $Res_Q = \{(id_1, a_1), (id_2, a_2), \ldots, (id_k, a_k)\}$ where id_is are the corresponding $TupleID$s of the tuples for which query condition is satisfied and a_is are $attr$ values and require verification. a_is have been chained with respect to their corresponding $attr'$ values as shown in Figure 3.

Res_Q is obtained at the client side after the reconstruction of returned shares, as the servers' result $SrvRes_Q$:

$$SrvRes_Q = \{(id_1, \{share_1(a_1), \ldots, share_t(a_1)\}),\\ (id_2, \{share_1(a_2), \ldots, share_t(a_2)\}),\\ \ldots,\\ (id_k, \{share_1(a_k), \ldots, share_t(a_k)\})\}$$

Let $v.nxt$ and $v.prv$, accessible from VO_v, be the id of respectively predecessor and successor values of v in the chain. Let also $v.SrcVal$, which is accessible from VO_v as well, be the v's corresponding searchable value in the tuple. Now, we show that different kinds of active attacks on $SrvRes_Q$ causes the verification process to return $False$, meaning the detection of correctness violation.

Spoofing attacks- The server may forge the possessed data and change some data shares. In this case, at least a share value $share_j(a_i)$ in $SrvRes_Q$ $(1 \leq i \leq k$ and $1 \leq j \leq t)$ has been replaced with $share'_j(a_i)$ by the server:

$$SrvRes_Q = \{(id_1, \{share_1(a_1), \ldots, share_t(a_1)\}),$$

$$(id_2, \{share_1(a_2), \ldots, share_t(a_2)\}),$$
$$\ldots,$$
$$(id_i, \{share_1(a_i), \ldots, share'_j(a_i), \ldots, share_t(a_i)\}),$$
$$\ldots,$$
$$(id_k, \{share_1(a_k), \ldots, share_t(a_k)\})\}$$

After reconstruction, we have
$Res'_Q = \{(id_1, a_1), (id_2, a_2), \ldots, (id_i, a'_i), \ldots, (id_k, a_k)\}$

- If $a'_i = a_i$ then $Res'_Q = Res_Q$ and there is no problem if the result is verified.
- If $a'_i \neq a_i$ then the result is verified if $VO_{a'_i} = VO_{a_i}$, which implies holding $a'_i.nxt = id_{i+1}$, $a'_i.prv = id_{i-1}$, and $l \leq a'_i.SrcVal \leq u$. If so, the adversary, without accessing the reconstruction vector, can accomplish function $\mathcal{R}$ and obtain his arbitrary VO, which is in contradiction to the security of the used MSS scheme.

The same reason is valid when colluding servers try to change shares of an individual value. The colluding servers cannot find out the reconstruction vector despite their collaboration. Therefore, it is not possible to generate share values, which reflect a fake secret whose VO conforms the existing chain of values. Concluding the above discussion, we can derive the following lemma.

Lemma 4.1. *Spoofing attacks do not remain undetected after query result verification.*

Splicing attacks- The server may replace shares of tuples which are not in the result set with share values of the actual result. A special case of this attack occurs when one share of a value is replaced with a share from another cell of the relation. This case is similar to the spoofing attack, detected after verification.

If all shares of a value a_i, are replaced with other shares of a specific tuple with $TupleID = id_s$, then
$SrvRes_Q = \{(id_1, \{share_1(a_1), \ldots, share_t(a_1)\}),$
$(id_2, \{share_1(a_2), \ldots, share_t(a_2)\}),$
$\ldots,$
$(id_{i-1}, \{share_1(a_{i-1}), \ldots, share_t(a_{i-1})\}),$
$(id_s, \{share_1(c), \ldots, share_t(c)\}),$
$(id_{i+1}, \{share_1(a_{i+1}), \ldots, share_t(a_{i+1})\}),$
$\ldots,$
$(id_k, \{share_1(a_k), \ldots, share_t(a_k)\})\}$

After reconstruction of shares in $SrvRes_Q$:

- If $(id_s, c) \in Res_Q$, then $id_s = id_i$ and $c = a_i$ so the client can access to the actual result Res_Q and there is no problem to verify the result.
- If $(id_s, c) \notin ResQ$, two equalities $a_i.nxt = c.nxt$ and $c.prv = a_i.prv$ must hold to have the chain of values verified. Moreover, $l \leq c.SrcVal \leq u$ must hold to have the query truly executed. According to the security of the MSS scheme [18, 21], no adversary, without accessing the reconstruction vector, can choose some shares in order to obtain a desirable VO.

The above analysis motivates derivation of Lemma 4.2 about our solution.

Lemma 4.2. *Splicing attacks do not remain undetected after query result verification.*

Unauthorized value insertion- The server may try to insert a value into Res_Q. We show that for each fake value v_f with its corresponding id_f, inserted into Res_Q, $Res'_Q = Res_Q \cup \{(id_f, v_f)\} = \{(id_1, a_1), (id_2, a_2), \ldots, (id_k, a_k), (id_f, v_f)\}$ is not verified at the client side.

Res'_Q implies that the server returns $SrvRes_Q$ as the following:
$SrvRes_Q = \{(id_1, \{share_1(a_1), \ldots, share_t(a_1)\}),$
$(id_2, \{share_1(a_2), \ldots, share_t(a_2)\}),$
$\ldots,$
$(id_k, \{share_1(a_k), \ldots, share_t(a_k)\}),$
$(id_f, \{share_1(v_f), \ldots, share_t(v_f)\})\}$

- If the server chooses $share_i(v)$ $(1 \leq i \leq t)$ from the relation but out of Res_Q (similar to the splicing attack for a fake insertion), after the reconstruction of $share_i(v_f)$ $(1 \leq i \leq t)$, we have $l \nleq v_f.SrcVal \nleq u$ since the server has chosen $share_i(v_f)$ from the relation but out of Res_Q. Therefore, Res'_Q is not verified as there is a value, v_f, for which the query condition has not been satisfied.
- If the server spoofs some shares and inserts them into the result, the result is not verified at the client side, similar to the spoofing attack. This is because the fake value requires to be inserted into the existing chain of values. No adversary, without access to the key vector, is able to spoof some shares and insert them into the result such that their reconstructed VO desirably modifies the existing chain of values.
- The server can choose share values from Res_Q, e.g., $share_j(a_i)$ $(1 \leq j \leq t)$ and replicate them in the result with either a new or the same $TupleID$. In the latter case, it is obvious for the client that there are repetitive values belonging to the same tuple. However, if the server changes $TupleID$, we have the following $SrvRes_Q$:
$SrvRes_Q = \{(id_1, \{share_1(a_1), \ldots, share_t(a_1)\}),$
$\ldots,$
$(id_i, \{share_1(a_i), \ldots, share_t(a_i)\}),$
$(id', \{share_1(a_i), \ldots, share_t(a_i)\}),$
$\ldots,$
$(id_k, \{share_1(a_k), \ldots, share_t(a_k)\})\}$
After reconstruction, there are two a_i values with the same VO because they have been reconstructed from the same set of shares. The same VOs means that two values have the same position in the chain; $a_{i-1}.nxt = id_i$ and $a_{i-1}.nxt = id'$, which is a contradiction. Therefore, the fake repetitive value a_i with $TupleID = id'$ cannot be inserted into the chain of values unless the fake insertion is detected after the verification.

Above discussion suggest Lemma 4.3, which shows that

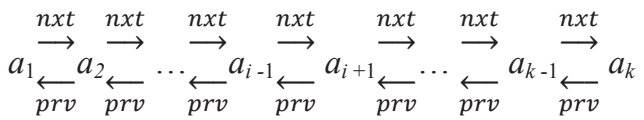

FIGURE 4. The chain of values when a_i is removed from the result

our solution tolerates invalid insertions to query results.

LEMMA 4.3. *Any unauthorized insertion of shares into the servers' results does not remain undetected at the client side.*

Incomplete result- The server may remove one or more values from Res_Q. Assume that the server wants to remove (id_i, a_i) from Res_Q. Thus, the returned result from the server is

$$\begin{aligned} SrvRes_Q = \{&(id_1, \{share_1(a_1), \ldots, share_t(a_1)\}),\\ &(id_2, \{share_1(a_2), \ldots, share_t(a_2)\}),\\ &\ldots,\\ &(id_{i-1}, \{share_1(a_{i-1}), \ldots, share_t(a_{i-1})\}),\\ &(id_{i+1}, \{share_1(a_{i+1}), \ldots, share_t(a_{i+1})\}),\\ &\ldots,\\ &(id_k, \{share_1(a_k), \ldots, share_t(a_k)\})\} \end{aligned}$$

and the reconstructed result at client side is $Res'_Q = \{(id_1, a_1), \ldots, (id_{i-1}, a_{i-1}), (id_{i+1}, a_{i+1}), \ldots, (id_k, a_k)\}$. Res'_Q is verified if the client can build the chain shown in Figure 4 from the values in Res'_Q.

The chain in Figure 4 is built if $a_{i-1}.nxt = id_{i+1}$ whereas $a_{i-1}.nxt = id_i$, according to the actual chain of values. Therefore, the chain cannot be built and Res'_Q is not verified.

There is also a special case of incomplete result when the server removes a value from the result borders. For example the server may want to remove a_1 or a_k. It is worth to mention that the server in our solution, unlike majority of existing solutions, has no information about the outsourced data such as the ordering relation between shares. Therefore, it is not straightforward for the server to identify share values from the borders of a query result. However, for small sets of results it is possible to remove a value, which is either the lower or the upper bound. Regarding the verification process described in Section 3.1.4 (and Algorithm 1), the client performs a separate process to check the boundaries of the result, which enables it to detect removing values from the result borders. Concluding from the above discussion, we derive Lemma 4.4 as below.

LEMMA 4.4. *Incomplete result does not remain undetected in our solution.*

Lemmas 4.1 to 4.4 state that the correctness of a query result is assured when it is verified at client side. Regarding Definition 1 in which a correct query result is described, and concluding from Lemmas 4.1 to 4.4, Theorem 4.1 is derived about our solution.

THEOREM 4.1. *Any unauthorized manipulation of shares, which produces an incorrect result, is detected at the client side.*

We remark that our solution is also secure against replay attacks in which the server without actually executing the query returns the result of a previous query instead of the actual result. In this case, although there is a valid chain of values in the result, reconstructed VOs reveal that query condition is not matched with the query result. Therefore, the result is not verified by the client.

4.2. Data Confidentiality Issues

The risk of data exposure has always been an obstacle for outsourcing data to the servers, which are not fully trusted. One of the major advantages of our solution is that it preserves the confidentiality of outsourced data against the untrustworthy servers hosting data shares.

Data shares generated by the proposed $(4, t, n)$ MSS scheme do not violate the confidentiality of outsourced data. This is because our MSS scheme is based on Yang *et al.*'s method, which offers a perfect secret sharing scheme [18]. In a perfect scheme, t or more participants can pool their shares and reconstruct the secrets while less than t shares provide no more information about the secret compared to an adversary who knows no shares. It is worth to mention that the perfect security property in our solution falls in the multiple-secrets security model [25, 17]. In such a security model, less than t shares having the value of a number of secrets might determine some information on other secrets.

When data shares are stored on separate non-communicating servers, no single server can find out secret attribute values. Even if the data shares are stored either on a single server or on separate but communicating servers, the servers are not yet able to find out the secret values as they have no parts of the key vector used for the secret reconstruction. Such ability is equal to finding out the actual solution for a set of simultaneous equations where the number of unknown variables is more than the number of the equations.

Now, we go through the case in which the untrusted server has *a priori* knowledge about the outsourced data. The server still cannot infer anything about the original data distribution, e.g., the order or the frequencies, by observing the data shares. This is because in Yang *et al.*'s [18] scheme data shares are uniformly distributed across their domain, independently of the original secret distribution. Masucci [17] have shown that the security property of an MSS scheme is independent of the particular probability distribution on the sets of secrets. That is, an MSS scheme with secrets of a particular probability distribution is an MSS scheme for any probability distribution on the set of the secrets.

Figure 5(a) illustrate the frequency distribution of

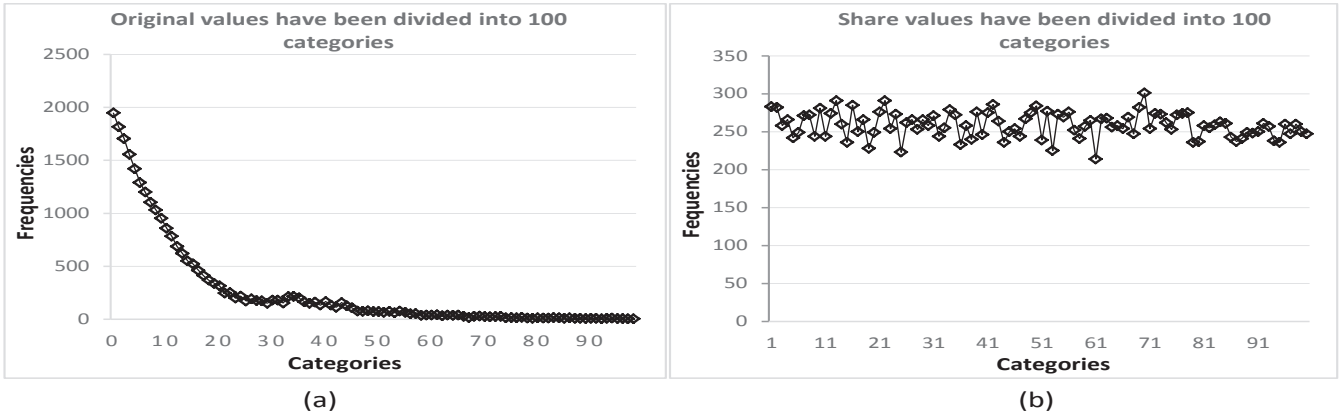

FIGURE 5. Frequency distribution of values (a) and their corresponding shares (b)

about 25000 values extracted from [26], which provides real data sets in different working domains. We used our solution to generate their corresponding shares. Figure 5(b) depicts the frequency distribution of the shares observed on one server and associated with attribute values of Figure 5(a). Our examination by the Chi-Square test method for uniformity, confirmed with a high confidence that the observed distribution of shares in 5(b) is consistent with a uniform distribution. Uniform distribution of shares prevents the servers relying on their *a priori* knowledge to find any valid association between shares and their corresponding secrets; so the attack in [27] is mitigated. This attack describes how untrusted servers align shares based on their ordering and then map the aligned shares onto actual secret values relying upon prior knowledge of the original data distribution. It is shown that in schemes such as [15] and [13], in which the order of shares is revealed to the servers, data confidentiality can be violated.

5. PERFORMANCE ANALYSIS

We implemented a prototype of our solution and assessed its computation and communication overheads. According to the comparative study of Bajaj and Sion [28], Narasimha and Tsudik's work [3], which is based on signature chaining and aggregation, is a superior work on the metrics of query execution and verification times for selection and range queries. Since their work employs signature aggregation, the VO is small and does not incur the high communication overhead of tree based approaches. Moreover, client side verification processes as well as data updates are performed more efficiently. Taking this into account, we have also implemented Narasimha and Tsudik's work [3], referred to NT for short in this section, to compare the efficiency of our solution with that of existing works. To this end, we wrote a program in Java, which implements both our solution and NT.

Regarding our solution, we split verifiable attribute values using our proposed (4, 5, 5) MSS scheme. The searchable secret sharing scheme proposed in [9] was used to split the values of searchable attributes. Supporting multiple search attributes has been implemented using the approach illustrated in 2. Regarding NT, RSA signatures were implemented with the key length 2048 bits which is recommended by NIST [29]. Hash functions are MD5 with 256-bit outputs. Modular calculations in our solution as well as in NT were implemented using the BigInteger class in Java.

The performance tests were executed on a relational dataset of about eight million tuples extracted from IPUMS 2010 ACS data [26]. The relation schema containing ten attributes such as an ID, total income, birth place, age, family size, marital status, etc. We stored data shares on MySQL 5.1 databases running on Windows 7 operating system with 8GB of main memory, a quad core 3GHz processor, and a 256GB SDD hard disk. The reported results of query execution and verifications times are the average of 100 independent executions.

5.1. Query Execution and Verification

In our first series of experiments we examine the query execution and verification times for range queries with different result sizes. The times are extracted at both client and server sides for our work as well as for NT. Figure 6 compares the server-side query execution time, which includes the server required time to execute query (performing a search operation over stored data) plus the verification relevant actions such as constructing VO in NT. We have divided the figure into two parts (a) and (b) for small and large query results. Figure 6(a) and Figure 6(b) indicate that our solution significantly outperforms NT particularly when the number of tuples in the result or the number of searchable attributes in the relation goes up. This is because our solution does not need to build VOs. Instead, they have been embedded into the query results. Consequently, the time is only limited to searching over the stored shares.

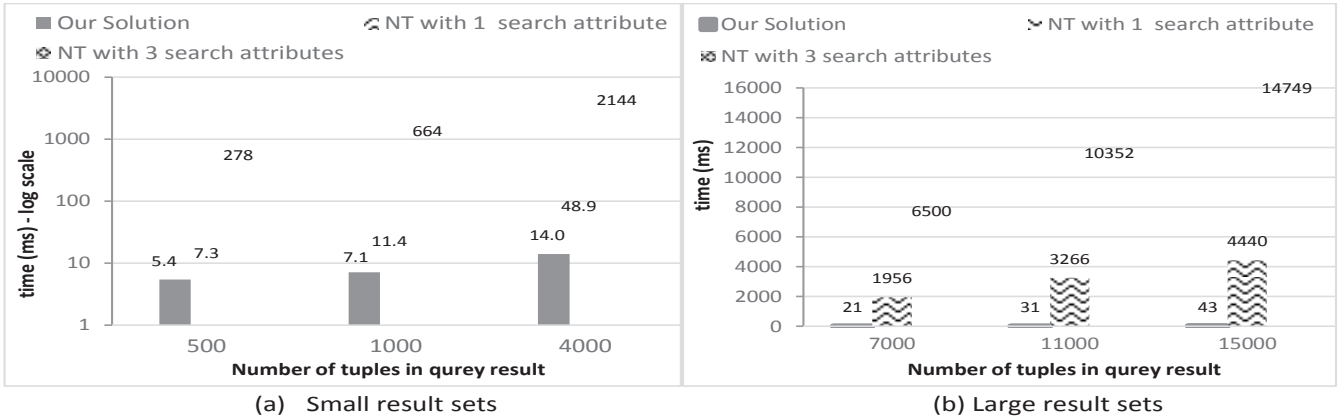

FIGURE 6. Comparison of server-side query execution times

In NT, the server time sharply increases when the size of answer goes up. This is because aggregating signatures is a computationally demanding process. Figure 6 also suggests that increasing the number of searchable attributes has a considerable impact on the server time for NT, whereas it has no impact in our solution, according to the extensions described in Section 3.2 for multiple search attributes.

Figure 7 shows client-side query result verification times in two parts (a) and (b) for small and large result sizes, respectively. Client verification time in our solution consists of building the sorted list of results, checking if the query condition is truly satisfied for each element in the result, and verifying the chain of values (Algorithm 1). Thus, the time complexity of client-side verification in our solution is $O(n)$ when n is the result size. Client-side verification time in NT includes verifying the query result using an aggregated signature plus some VOs returned by the server. Figure 7(a) signifies that our solution surpasses NT for small result sets specially when there are multiple search attributes. However, the verification time increases in our solution proportional to the result size. Consequently, as depicted in Figure 7(b), for query results with about more than 10000 tuples, NT performs better in terms of client-side query result verification.

Efficiency of data updates is the focus of our second series of experiments in which we measure the time of an INSERT operation. The experiments show that our solution can perform well with database updates over time. Figure 8 compares the execution time of an INSERT query per one and three searchable attributes in a relation. The total execution time in the figure has been broken to the server and client required times. The figure demonstrates that our solution performs an INSERT in a lesser time compared to NT due to more efficient server-side processes.

Overall, taking figures 6, 7, and 8 into consideration clearly shows that the total time of query execution and verification as well as the time for data updates in our solution is considerably less than those of NT.

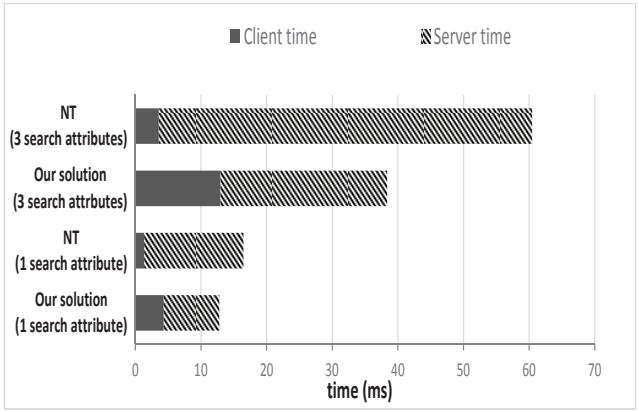

FIGURE 8. Comparison of the required time to INSERT a new tuple into the relation

5.2. Communication Cost

For each query execution, communication cost includes the query answer plus its corresponding VO, which are transmitted from the server to the querier client. The imposed overhead related to the query answer depends on the granularity level at which correctness verification is provided. Correctness verification at the tuple level implies that the client verifies the correctness when the entire data pertaining to the tuple is returned. That is, the smallest unit of returned data is the whole tuple, even for projection queries for which a client may request only a single attribute. Our approach, however, verifies query results at the granularity of attribute value. This results in no overhead to the query answer despite verification.

The size of VO is another parameter, which directly affects the communication cost. For selection and range queries, signature-based schemes with aggregated signatures such as NT provide the smallest VO and

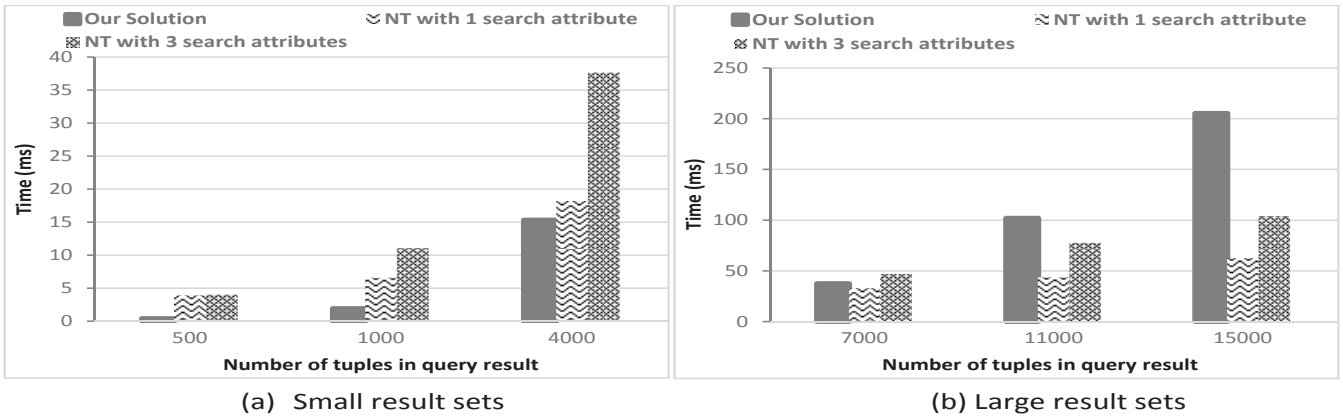

FIGURE 7. Comparison of client-side query result verification times

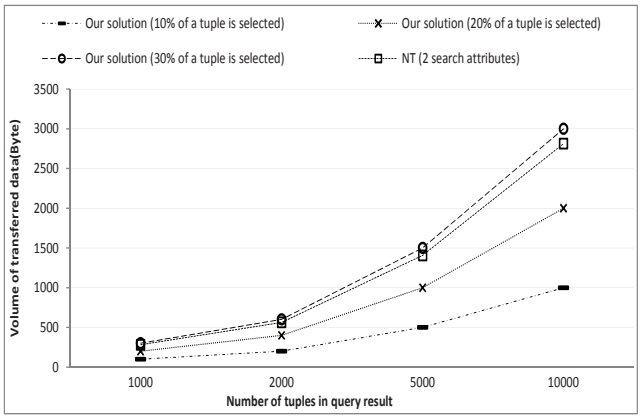

FIGURE 9. Communication cost per different result sizes

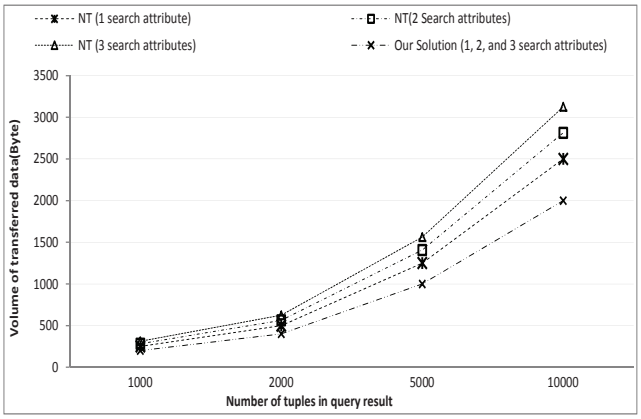

FIGURE 10. Communication cost for different numbers of searchable attributes

thus impose the least network overhead. However, since the verification is performed at the tuple level, whole tuples must be returned in response to projection queries. Figure 9 compares the communication cost of our solution with NT per different result sizes. In this experiment, there are two searchable attributes in the relation and the size of a tuple is 256 Bytes. Figure 9 indicates that our solution has less communication cost than NT when the querier requests small percentages of a tuple, e.g., below 30%, through projection queries. Although our solution does not have any explicit VO in query results, it has more communication cost when a high percentage of a tuple should be retrieved. This is due to the communication overhead of using (t, n) secret sharing schemes for data outsourcing, which requires transferring t shares per secret value. We remark that the imposed cost of sending t shares per attribute value is for achieving data confidentiality against the curious servers.

The number of searchable attribute is another parameter, which affects the communication cost. Figure 10 compares communication costs per different result sizes when 20 percent of tuples are retrieved through projection queries. It suggests that the number of searchable attributes slightly affects the communication cost in NT. In our solution, however, the communication is fixed, independent of the number of searchable attributes, as discussed in 3.3.

6. RELATED WORK

Existing solutions for query result verification try to guarantee the correctness of query results either by the use of costly cryptographic functions such as digital signatures or by constructing Authentication Data Structures (ADS) such as Merkel Hash Trees (MHT). However, the correctness is usually assured at the considerable expenses of computation, storage, and communication overheads for the construction, storage, and transfer of VOs especially for dynamic data sets [28]. Some solutions suggest probabilistic guarantees to reduce the overheads [30, 31]. Taking both approaches into account, in this section, we briefly review important research on query result verification.

6.1. Correctness Assurance Structures

Existing structures used for correctness assurance of query results are either tree based or signature based. Authenticated skip lists are also another data structure proposed for completeness verification of query results.
a) Merkel Hash Tree and its Variants: Merkel Hash Tree (MHT) is an authentication data structure exploited at first by Devanbu *et al.* [2] to verify the integrity and completeness of query results. MTH based verification usually suffers from the inefficiency of data update. Moreover, it imposes a considerable communication cost between the server and the clients due to the large VO constructed at the server side. The result of a study shows that the cost of data transfer between a cloud server and a querier client is 2-3 orders of magnitude higher than processing costs [32]. Different variants of MHT were proposed to improve its performance in data updates [33] and verification efficiency [1]. Using the idea of MHT, some researchers enriched B-tree index structures and proposed Merkel B-tree (MB-tree) for the verification of query results [34, 35]. Jain and Prabhakar [36] also customized Merkel B-tree such that the execution and verification of one dimensional range queries comply access control policies. The solutions in which B-tree based index structures are modified require some changes to the existing DBMSs' internals which hinder their usage in practice. Considering this problem, Wei and Yu [37] serialized MB-tree structures into a relational database that can verify the integrity of query results without requiring modifications to the server-side DBMS.
b) Signatures and Message Authentication Codes: Using digital signature and Message Authentication Code (MAC) have been widely adopted to verify the authenticity of a message. In the context of data outsourcing, the data owner can outsource the data either with its signature or its MAC as VO. MAC is appropriate for the unified client model where the client and the querier are the same. Instead, in multi-querier and multi-owner models of data outsourcing in which we do not want to share the same MAC key between all owners and legitimate queriers, public key digital signatures are proposed [38]. While processing a query, in addition to the query result, the server sends back the corresponding MACs or signatures. They are used by the client for verifying the integrity of the returned result. However, neither the MAC nor the signature, cannot verify the completeness of query results by themselves.

Narasimha and Tsudik [3] proposed signature chaining in which a signature of a tuple depends not only on the content of the tuple but also on the content of the immediate predecessor tuple of the to-be-signed tuple with respect to a searchable dimension. They used such a signature structure for the purposes of completeness and integrity verifications. Considering the efficiency of MAC compared to signature schemes, Silvrio *et al.* [39] proposed a MAC based scheme to check the correctness of query results. They use the idea of chained MACs as well to provide the completeness verifiability. In their solution, the chained MAC of a tuple is inserted into the outsourced relation schema, whose value comes from a MAC operation over the MAC value of the tuple, XORed with the MAC of its previous tuple.

The granularity of data on which MAC or signature is performed is an important concern in above solutions. At the finest level of granularity, each attribute value is signed by the owner that imposes a high computation overhead to the data owner. Moreover, it uses up server-side storage to keep a signature per each attribute value. Switching to courser granularity, e.g., at the tuple level as proposed in [3, 39], while substantially reduces the owner computation cost and the server-side storage overhead, increases the communication cost because the server must return the whole tuple in response to projection queries. It is a significant overhead when the outsourced relation contains lots of attributes in its schema.

Another problem with these solutions is that the client-side verification cost linearly goes up proportional to the number of tuples in the query result. Aggregated signature schemes such as condensed RSA were proposed to tackle this problem [3, 38]. Aggregated signature combines signatures of a set of tuples into a unique signature, whose verification at client-side implies the verification of all individual signatures. Compared to the MHT based data structures, with aggregated signature, the VO size is fixed and equal to a single signed message. Although it seems acceptable for communication overhead, the server-side computation cost to construct an aggregated signature is considerably expensive [32].
c) Skip Lists: Authenticated skip lists are hierarchical data structures to store an ordered list of elements over an outsourced relation. Wang *et al.* [5] proposed a skip list based structure, which supports authentication of range queries in static as well as in dynamic scenarios. Other schemes such as [4, 40] used authenticated skip-lists as well for query verification in outsourcing scenarios.

6.2. Probabilistic Correctness Assurance

Previous approaches discussed in Section 6.1 usually suffer from the extra ordinary computation overhead for the client-side verification process, the server-side proof construction, or requested data updates. Some probabilistic approaches for query verification [24, 30] are aimed at proposing efficient verification. However, they cannot guarantee the correctness of query results.
a) Bloom Filters: Bloom filter, as a probabilistic data structure, originally proposed to check the membership of an element in a set. In the context of data outsourcing, bloom filters were also proposed to provide

integrity checking [41, 42]. Zhang *et al.* [35] mapped the fields of a tuple onto a bit string using a set of hash functions. The owner stores small size bloom filters associated to each tuple, instead of possibly large tuples. When a tuple is returned as query result, the owner can probabilistically verify the integrity of the tuple using its corresponding bloom filter. Indeed, this structure may bring about false positives whose probability depends on the length of the bloom filter, which in turn has a direct impact on the client-side storage overhead. Ferretti *et al.* [43] used an encrypted bloom filter for efficient integrity verification of database stored on the cloud. Their solution can be integrated with existing proposals which use encryption for the confidentiality of outsourced data.

b) Fake Tuples and Replication: A probabilistic approach used for query verification in data outsourcing scenarios suggests inserting fake tuples in the relation before its outsourcing [31]. While the owner and querier clients can distinguish between real and fake tuples, the server to which data is outsourced cannot. When a client receives a query result, it checks whether the fake tuples that satisfy the query condition are in the result. Missing a fake tuple in the received result implies that the result is incomplete. Xie *et al.* [31] showed that a limited number of fake tuples can provide a high probabilistic guarantee for the completeness of query results.

Another probabilistic approach relies upon the replication of a subset of the outsourced encrypted relation, i.e., a subset of encrypted tuples [44]. The replicated tuples in the relation must be indistinguishable from their original copies from the server's viewpoint. To this purpose, a subset of tuples is chosen by the owner and encrypted twice by different keys. The other tuples are encrypted just once by one of the keys. To execute a query, the query is translated into two queries regarding the two different keys. Results of these two queries are then compared by the client, after decryption, to check whether they contain exactly the same duplicate tuples. Missing a duplicate tuple in one of the results immediately results in the completeness violation. Probabilistic completeness guarantee of this approach is directly affected by the number of replicated tuples.

6.3. Using Trusted Hardware

There are other works that investigate hardware support for query verification. Bajaj and Sion [28] proposed tamper proof coprocessors as trusted hardware to act on behalf of clients but in close proximity to the outsourced data, i.e., at the server side. They claim that this hardware based solution, despite its higher acquisition cost, is relatively more efficient and even practical than the existing software based cryptographic protocols. The tamper proof coprocessor provides a secure execution environment in which any illicit physical manipulation destroys the device.

7. CONCLUSION

Existing approaches for correctness assurance are mainly based on authentication data structures and digital signatures. While the former usually suffers from the inefficient verification process for large data sets, the latter imposes more storage overhead specially with digital signatures on fine grained data elements.

In this paper, we used the idea of multi-secret sharing schemes to propose a solution, which verifies the correctness of query results at the finest level of granularity, i.e., an individual attribute value. We split each attribute value together with its VO into several data shares at a single splitting process. When the server returns data shares as a query result, the client reconstructs attribute values and their corresponding VOs through a single reconstruction process. Then, reconstructed VOs are used to verify the correctness of the result.

Our solution does not impose extra communication and storage costs due to its fine granularity. From the computation viewpoint, the server searches the result into which its VO has been previously embedded and sends it to the client. At the client side, the VO is reconstructed in parallel with the result. Our solution not only provides the verifiability of query results but also preserves the confidentiality of outsourced data by secure splitting of attribute values.

The solution can be applied in a multi-cloud setting in which multiple and potentially colluding cloud servers provide data management services as well as in a single cloud model of data outsourcing. More importantly, it can directly be applied to the existing DBMSs without changes to their engines.

We plan to extend our solution to guarantee the freshness of results in dynamic environments such that stale results to the same queries are detected by the verification process.

REFERENCES

[1] Goodrich, M. T., Tamassia, R., and Triandopoulos, N. (2008) Super-efficient verification of dynamic outsourced databases. *Proceedings of the 2008 The Cryptopgraphers' Track at the RSA Conference on Topics in Cryptology*, pp. 407–424. Springer-Verlag.

[2] Devanbu, P., Gertz, M., Martel, C., and Stubblebine, S. (2001) Authentic third-party data publication. In Thuraisingham, B., van de Riet, R., Dittrich, K., and Tari, Z. (eds.), *Data and Application Security*, pp. 101–112. Springer US.

[3] Narasimha, M. and Tsudik, G. (2006) Authentication of outsourced databases using signature aggregation and chaining. *Proceedings of the 11th International Conference on Database Systems for Advanced Applications*, Berlin, Heidelberg DASFAA'06, pp. 420–436. Springer-Verlag.

[4] Di Battista, G. and Palazzi, B. (2007) Authenticated relational tables and authenticated skip lists. In Barker, S. and Ahn, G.-J. (eds.), *Data and Applications Security XXI*, Lecture Notes in Computer Science, **4602**, pp. 31–46. Springer Berlin Heidelberg.

[5] Wang, J. and Du, X. (2009) Skip list based authenticated data structure in das paradigm. *Eighth International Conference on Grid and Cooperative Computing (GCC'09)*, Aug, pp. 69–75.

[6] Emekci, F., Methwally, A., Agrawal, D., and Abbadi, A. E. (2014) Dividing secrets to secure data outsourcing. *Information Sciences*, **263**, 198–210.

[7] Shamir, A. (1979) How to share a secret. *Commun. ACM*, **22**, 612–613.

[8] Le, M., Kant, K., and Jajodia, S. (2011) Cooperative data access in multi-cloud environments. In Li, Y. (ed.), *Data and Applications Security and Privacy XXV*, Lecture Notes in Computer Science, **6818**, pp. 14–28. Springer Berlin Heidelberg.

[9] Hadavi, M. A., Damiani, E., Jalili, R., Cimato, S., and Ganjei, Z. (2013) AS5: A Secure Searchable Secret Sharing Scheme for Privacy Preserving Database Outsourcing. *Data Privacy Management and Autonomous Spontaneous Security*, Lecture Notes in Computer Science, **7731**, pp. 201–216. Springer Berlin Heidelberg.

[10] Hadavi, M. A. and Jalili, R. (2010) Secure data outsourcing based on threshold secret sharing: Towards a more practical solution. *Proceedings of VLDB PhD Workshop*, pp. 54–59. VLDB Endowment.

[11] Hadavi, M., Jalili, R., Damiani, E., and Cimato, S. (2015) Security and Searchability in Secret Sharing-based Data Outsourcing. *International Journal of Information Security*, **preprint**, 1–17.

[12] Agrawal, D., El Abbadi, A., Emekci, F., and Metwally, A. (2009) Database Management as a Service: Challenges and Opportunities. *IEEE 25th International Conference on Data Engineering(ICDE'09)*, pp. 1709–1716.

[13] Agrawal, D., Abbadi, A. E., Emekci, F., Metwally, A., and Wang, S. (2011) Secure data management service on cloud computing infrastructures. *New Frontiers in Information and Software as Services*, Lecture Notes in Business Information Processing, **74**, pp. 57–80. Springer.

[14] AlZain, M. A., Soh, B., and Pardede, E. (2012) A new model to ensure security in cloud computing services. *Journal of Service Science Research*, **4**, 49–70.

[15] Tian, X., Sha, C., Wang, X., and Zhou, A. (2011) Privacy preserving query processing on secret share based data storage. *Database Systems for Advanced Applications*, Lecture Notes in Computer Science, **6587**, pp. 108–122. Springer Berlin Heidelberg.

[16] Hadavi, M. A., Noferesti, M., Jalili, R., and Damiani, E. (2012) Database as a service: Towards a unified solution for security requirements. *Proceedings of 36th IEEE COMPSACW*, July, pp. 415–420. IEEE Computer Society.

[17] Masucci, B. (2006) Sharing Multiple Secrets: Models, Schemes and Analysis. *Designs, Codes and Cryptography*, **39**, 89–111.

[18] Yang, C.-C., Chang, T.-Y., and Hwang, M.-S. (2004) A (t,n) multi-secret sharing scheme. *Applied Mathematics and Computation*, **151**, 483–490.

[19] Liu, Y.-X., Harn, L., Yang, C.-N., and Zhang, Y.-Q. (2012) Efficient (n, t, n) secret sharing schemes. *Journal of Systems and Software*, **85**, 1325–1332.

[20] Hsu, C.-F., Cheng, Q., Tang, X., and Zeng, B. (2011) An ideal multi-secret sharing scheme based on MSP. *Information Sciences*, **181**, 1403–1409.

[21] Pang, L.-J. and Wang, Y.-M. (2005) A new (t, n) multi-secret sharing scheme based on shamir's secret sharing. *Applied Mathematics and Computation*, **167**, 840–848.

[22] Lin, C. and Harn, L. (2012) Unconditionally secure multi-secret sharing scheme. *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, May, pp. 169–172.

[23] D'Arco, P., Kishimoto, W., and Stinson, D. R. (2006) Properties and constraints of cheating-immune secret sharing schemes. *Discrete Appl. Math.*, **154**, 219–233.

[24] Harn, L. and Lin, C. (2009) Detection and identification of cheaters in (t, n) secret sharing scheme. *Designs, Codes and Cryptography*, **52**, 15–24.

[25] Blundo, C., De Santis, A., Di Crescenzo, G., Gaggia, A., and Vaccaro, U. (1994) Multi-Secret Sharing Schemes. In Desmedt, Y. (ed.), *Advances in Cryptology – CRYPTO'94*, Lecture Notes in Computer Science, **839**, pp. 150–163. Springer Berlin Heidelberg.

[26] Ruggles, S., Alexander, J. T., Genadek, K., Goeken, R., Schroeder, M. B., and Sobek, M. (2010) Integrated Public Use Microdata Series: Version 5.0 [Machine-readable database]. Technical report. University of Minnesota, Minneapolis: University of Minnesota.

[27] Dautrich, J. L. and Ravishanka, C. V. (2012) Security limitations of using secret sharing for data outsourcing. *Proceedings of DBSec 2012*, July Lecture Notes in Computer Science, pp. 145–160. Springer-Verlag.

[28] Bajaj, S. and Sion, R. (2013) Correctdb: Sql engine with practical query authentication. *Proceedings of VLDB Endowment*, **6**, 529–540.

[29] Barker, E. B., Barker, W. C., Burr, W. E., Polk, W. T., and Smid, M. E. (2007) Sp 800-57. recommendation for key management, part 1: General (revised). Technical report.

[30] Ghasemi, S., Noferesti, M., Hadavi, M. A., Dorri Nogoorani, S., and Jalili, R. (2012) Correctness verification in database outsourcing: A trust-based fake tuples approach. *Information Systems Security*, Lecture Notes in Computer Science, **7671**, pp. 343–351. Springer Berlin Heidelberg.

[31] Xie, M., Wang, H., Yin, J., and Meng, X. (2007) Integrity auditing of outsourced data. *Proceedings of the 33rd International Conference on Very Large Data Bases* VLDB'07, pp. 782–793. VLDB Endowment.

[32] Chen, Y. and Sion, R. (2011) To cloud or not to cloud?: Musings on costs and viability. *Proceedings of the 2Nd ACM Symposium on Cloud Computing* SOCC'11, pp. 1–7. ACM.

[33] Mouratidis, K., Sacharidis, D., and Pang, H. (2009) Partially materialized digest scheme: An efficient verification method for outsourced databases. *The VLDB Journal*, **18**, 363–381.

[34] Li, F., Hadjieleftheriou, M., Kollios, G., and Reyzin, L. (2006) Dynamic authenticated index structures for outsourced databases. *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data* SIGMOD'06, pp. 121–132. ACM.

[35] Pang, H. and Tan, K.-L. (2004) Authenticating query results in edge computing. *20th International Conference on Data Engineering*, March, pp. 560–571.

[36] Jain, R. and Prabhakar, S. (2013) Access control and query verification for untrusted databases. *Data and Applications Security and Privacy XXVII*, Lecture Notes in Computer Science, **7964**, pp. 211–225. Springer Berlin Heidelberg.

[37] Wei, W. and Yu, T. (2014) Integrity assurance for outsourced databases without dbms modification. *Data and Applications Security and Privacy XXVIII*, Lecture Notes in Computer Science, **8566**, pp. 1–16. Springer Berlin Heidelberg.

[38] Mykletun, E., Narasimha, M., and Tsudik, G. (2006) Authentication and integrity in outsourced databases. *Trans. Storage*, **2**, 107–138.

[39] Silverio, A. L., Custodio, R. F., Carlos, M. C., and dos Santos Mello, R. (2014) Efficient data integrity checking for untrusted database systems. *Proceedings of the Sixth International Conference on Advances in Databases, Knowledge, and Data Applications*, pp. 118–124.

[40] Heitzmann, A., Palazzi, B., Papamanthou, C., and Tamassia, R. (2008) Efficient integrity checking of untrusted network storage. *Proceedings of the 4th ACM International Workshop on Storage Security and Survivability* StorageSS '08, pp. 43–54. ACM.

[41] Pang, H., Zhang, J., and Mouratidis, K. (2009) Scalable verification for outsourced dynamic databases. *Proceedings VLDB Endowment*, **2**, 802–813.

[42] Zhang, M., Cai, K., and Feng, D. (2010) Fine-grained cloud db damage examination based on bloom filters. In Chen, L., Tang, C., Yang, J., and Gao, Y. (eds.), *Web-Age Information Management*, Lecture Notes in Computer Science, **6184**, pp. 157–168. Springer Berlin Heidelberg.

[43] Ferretti, L., Pierazzi, F., Colajanni, M., Marchetti, M., and Missiroli, M. (2014) Efficient detection of unauthorized data modification in cloud databases. *Computers and Communication (ISCC), 2014 IEEE Symposium on*, June, pp. 1–6.

[44] Wang, H., Yin, J., Perng, C.-s., and Yu, P. S. (2008) Dual encryption for query integrity assurance. *Proceedings of the 17th ACM Conference on Information and Knowledge Management* CIKM'08, pp. 863–872. ACM.