**ABSTRACT**

We develop a secret-sharing-based prototype, entitled Obscure that provides communication-efficient and information-theoretically secure algorithms for aggregation queries using secret-sharing. The query execution algorithms deals with an honest-but-curious, as well as, a malicious server, by supporting result verification. In addition, Obscure prevents an adversary to know the data, the query, and the tuple-identity satisfying the query.

**CSC CONCEPTS**

- Security and privacy → Database and storage security.

**KEYWORDS**

Oblivious computation; secret-sharing; scalability; verification.

**ACM Reference Format:**


**1 INTRODUCTION**

The techniques for secure data outsourcing can be classified based on the cryptographic security into two categories: (i) **Computationally secure techniques** that assume the adversary lacks adequate computational capabilities to break the underlying cryptographic mechanism in polynomial time. Homomorphic encryption, order-preserving encryption (OPE), and searchable-encryption are examples of such techniques. (ii) **Information-theoretically secure techniques** that are unconditionally secure and independent of adversary’s computational capabilities. Shamir’s secret-sharing (SSS) [9] is a well-known information-theoretically secure protocol.

In SSS, multiple (secure) shares of a dataset are kept at mutually suspicous servers, such that a single server cannot learn anything about the data. Secret-sharing-based techniques are secure under the assumption that a majority of the servers (equal to the threshold of the secret-sharing mechanism) do not collude.

While both computationally and information-theoretically secure techniques have been studied extensively in the cryptographic domain, secure data management has focused disproportionately on computationally secure techniques (e.g., OPE, homomorphic encryption, searchable-encryption, and bucketization [8]). Recently, both academia and industries have begun to explore information-theoretically secure techniques using MPC that efficiently supports OLAP tasks involving aggregation queries, while achieving higher security than computationally secure techniques. For instance, commercial systems, such as Jana [2] by Galois, Pulsar [1] by Stealth Software, Sharemind [3] by Cybernetica, and products by companies such as Unbound Tech., Partisia, Secret Double Octopus, and SecretSkyDB Ltd. have explored MPC-based databases systems that offer strong security guarantees. Much of the above work on MPC-based secure data management requires several servers to collaborate to answer queries. These collaborations require several rounds of communication among non-colluding servers. Instead, we explore secure data management based on SSS that does not require servers to collaborate to generate answers and can, hence, be implemented more efficiently.

**Contributions.** Our contributions are as follows: (i) We develop a SSS-based prototype, entitled Obscure that supports a large class of access-pattern-hiding aggregation queries with selection. Obscure uses existing string-matching techniques [4] and order-preserving secret-sharing (OP-SS) [5, 6]. Particularly, Obscure supports count, sum, average, maximum, minimum, top-k, and reverse top-k queries – on a dataset outsourced by a single database (DB) owner or multiple DB owners, without revealing anything about data/query/results to an adversary. (ii) In addition, Obscure supports oblivous result verification for aggregation queries such that an adversary does not learn anything from the verification. Obscure’s verification step is not mandatory. A querier may run verification occasionally to confirm the correctness of results.

**Advantages of Obscure.** Obscure provides several advantages: (i) Deals with honest but curious, as well as, malicious adversaries (which could deviate from the algorithm and delete tuples from the relation). (ii) Does not overburden the DB owner by storing enough data related to polynomials and fully participating in query execution. (iii) Does not reveal access-patterns, while supporting selection predicate search over secret-shared data. (iv) Uses minimal communication rounds between the user and each server, (when having enough shares). Specifically, count, sum, average, and their verification algorithms require at most two rounds between each
3 SUM QUERY

Consider a query: select sum(A2) from R where A1 = v1 ∧ A2 = v2 ∧ ... ∧ A_m = v_m. In the secret-sharing setting, the user transforms the above query into the following query at the jth server: select sum(A2) from S(Rj) where A1 = S(v1j) ∧ A2 = S(v2j) ∧ ... ∧ A_m = S(vmj). The jth server performs the following operation on each attribute on which the user wants to compute the sum, i.e., Af and Ayj

\[ \sum_{k=1}^{n} A_{f}(s_{k}) \times \left( \left( \sum_{i=1}^{m} \langle A_{f}(s_{i}) \rangle \otimes S(v_{ij}) \right) \right) \]

\( \otimes \) shows a string-matching operation that depends on the underlying text representation, whose results will be 0 or 1 of secret-share form. Each server j compares the query predicate value S(vij) against kth value (1 ≤ k ≤ n) of the attribute Aj and multiplies the result with the ith values of the attribute Aj. Finally, the server adds all the values of the Aj attribute and sends to the user. On receiving the values from the servers, the user performs Lagrange interpolation to get the final answer in cleartext.

Result verification. We explain the result verification method using the following query on Employee relation (refer to Table 1): select sum(Salary) from Employee where Dept = 'Testing'. We show the verification operation in cleartext (see Table 3); however, the server will perform all operations over secret-shares. Here, our objective is to verify that (i) all tuples of the databases are checked against the query predicates and (ii) only all qualified values of the Salary attribute are included as an answer to the sum query. The method works as follows:

The DB owner. The DB owner adds two attributes, say Ax and Ay, to the relation R. The ith values of the attributes Ax and Ay are any two random numbers whose difference equals to −a_i, where a_i is the ith value of the attribute A_i. The values of the attributes Ax and Ay are also secret-shared using SSS. For example, in Table 3, boldface numbers show these random numbers of the attribute Ax and Ay in cleartext.

Server. Let select sum(Ai) from R where Aq = v be a query. The server computes two functions, f1 and f2, to verify the conditions of sum-query verification in an obvious manner, as follows:

\[ \phi_1 = \sum_{i=1}^{n} a_i(x_i + a_i + 0) \]
\[ \phi_2 = \sum_{i=1}^{n} a_i(y_i + a_i + 0) \]

i.e., the server executes the functions f_1 and f_2 on n values, added in attributes A_q and A_y. In the above equations, a_i is the output of the string-matching operation carried on the jth value of the attribute A_q, and a_i is the jth (1 ≤ i ≤ n) value of the attribute A_i. Finally, the server sends the following three things to the user: (i) the sum of the resultant values of the attributes A_i, say (sum_{1..1}^k), (ii) the sum of the output of the string-matching operations carried on the attribute A_q, say (sum_{1..1}^k), against the query predicate, and (iii) the sum of outputs of the functions f_1 and f_2, say (sum_{1..1}^k).

User-side. The user interpolates the received three values from each server, which results in Isum_{1..1} = 2 × Isum_{1..1} and Isum_{1..1}. If it finds equal, then it implies that the server has correctly executed the sum query.
4 MAXIMUM QUERY

We provide methods for finding the maximum value and retrieving the corresponding tuple for the two types of queries, where the first type of query (QMax1) does not have any query condition, while another (QMax2) is a conditional query, as follows:

QMax1. select * from Employee where Salary in (select max(Salary) from Employee)
QMax2. select * from Employee as E1 where E1.Dept = 'Testing' and Salary in (select max(salary) from Employee as E2 where E2.Dept = 'Testing')

Note that the string-matching secret-sharing algorithms [4] cannot find the maximum value, as these algorithms provide only equality-checking mechanisms, not comparing mechanisms to compare between values. Here, we provide a method that can solve an unconditional query (like QMax1).

Approach. We assume that \( A_c \) be an attribute of the relation \( S(R^3) \) on which the user wishes to execute maximum queries. Our idea is based on a combination of OP-SS [5] and SSS [4, 9] techniques. Specifically, for answering maximum queries, OBSOURE uses the two relations \( S(R^3) \) and \( S(R^4) \), which are secured using secret-shared and OP-SS, respectively. In particular, the attribute \( A_c \) will exist in the relations \( S(R^3)_i \) and \( S(R^4)_i \) at the server \( i \). The strategy is to jointly execute a query on the relations \( S(R^3)_i \) and \( S(R^4)_i \) and obliviously retrieve the entire tuple from \( S(R^3)_i \). The server can find the maximum value of the attribute \( A_c \) using the relation \( S(R^4)_i \), which is secret-shared using OP-SS, and then, can find the tuple having the maximum value from \( S(R^3)_i \) using string-matching mechanism. Particularly, the \( i^{th} \) server executes the following steps:

1. On the relation \( S(R^3)_i \). Since secret-shared values of the attribute \( A_c \) of relation \( S(R^3)_i \) are comparable, the server \( i \) finds a tuple \((S(t_k), S(value))_i\) having the maximum value in attribute \( A_c \), where \( S(t_k)_i \) is the \( k^{th} \) secret-shared tuple-id (in the attribute SSTID) and \( S(value)_i \) is the secret-shared value of \( A_c \) attribute in the \( k^{th} \) tuple.

2. On the relation \( S(R^4)_i \). Now, the server \( i \) performs the join of the tuple \((S(t_k), S(value)_i)\) with all the tuples of the relation \( S(R^4)_i \) by comparing the tuple-ids (TID attribute’s values) of the relation \( S(R^4)_i \) with \( S(t_k)_i \), as follows:

\[
\sum_{k=1}^{m} p(A_p[S(a_k)_i] \times (\text{TID}[S(a_k)_i]) \otimes S(t_k)_i)
\]

Where \( p \) (1 < \( p \) < \( m \)) is the number of attributes in the relation \( R \) and TID is the tuple-attribute of \( S(R^3)_i \). The server \( i \) compares the tuple-id \((S(t_k)_i)\) with each \( k^{th} \) value of the attribute TID of \( S(R^3)_i \) and multiplies the resultant by the first \( m \) attribute values of the tuple \( k \). The server \( i \) adds all the values of each \( m \) attribute and sends the resultant values to the user.

5 EXPERIMENTS

AWS servers of 144GB RAM, 3.0GHz Intel Xeon CPU with 72 cores were used to store the secret-shared data. We used a 16GB RAM machine as a DB owner and as a user that communicates with AWS servers. We used four columns (OrderKey, PartKey, Linenumber, and Supplykey) of LineItem table of TPC-H benchmark to generate 1M and 6M rows. To the best of our knowledge, this is the first such experiment of SSS-based approaches to such large datasets.

Exp 1. Obscure performance. We executed count, sum, unconditional and conditional maximum, and group-by queries on the Lineitem table 6M rows using fifteen shares; see Figure 1. In Obscure, the processing time at each server can be greatly reduced by parallelizing the computation. Since identical computations are executed on each row of the table, we can use multiple cores of CPU by writing a parallel program, which reduces the processing time. We wrote one-dimensional (1D) count/sum, two/three-dimensional conjunctive-equality (2CE/3CE) count/sum, and two/three-dimensional disjunctive-equality (2DE/3DE) count/sum, unconditional maximum queries, and group-by (that divide rows into blocks with one thread processing one block, and then, the intermediate results (generated by each thread) are reduced by the master thread to produce the final result.

Exp 2. Overheads of result verification. OBSOURE, also, verifies the query result. This experiment finds the overheads of the result verification approaches. Figure 2 shows the overhead of result verification approaches and compares it against non-verification algorithms. It shows that OBSOURE result verification steps do not incur a significant cost at the servers.

REFERENCES
