OBSCURE: Information-Theoretically Secure, Oblivious, and Verifiable Aggregation Queries*

Peeyush Gupta,[1] Yin Li,[2] Sharad Mehrotra,[1] Nisha Panwar,[1,3] and Shantanu Sharma[1]
[1]University of California, Irvine, USA. [2]Xinyang Normal University, China. [3]Augusta University, USA.

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.

CCS CONCEPTS

• **Security and privacy → Database and storage security**.

KEYWORDS

Oblivious computation; secret-sharing; scalability; verification.

ACM Reference Format:
Peeyush Gupta,[1] Yin Li,[2] Sharad Mehrotra,[1] Nisha Panwar,[1,3] and Shantanu Sharma[1]. 2020. OBSCURE: Information-Theoretically Secure, Oblivious, and Verifiable Aggregation Queries. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20), March 16–18, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3374664.3379533

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 suspicious 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 oblivious 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

*The conference version of this poster has appeared in VLDB 2019; refer to [7] for additional details of OBSCURE. This material is based on research sponsored by DARPA under agreement number FA8750-16-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. This work is partially supported by NSF grants 1527536 and 1545071, and National Natural Science Foundation of China (Grant no. 61402393, 61601396).

ACM ISBN 978-1-4503-7107-0/20/03…$15.00
https://doi.org/10.1145/3374664.3379533

Table 1: A relation: Employee.

EmpID	Name	Salary	Dept
E101	John	1000	Testing
E101	John	100000	Security
E102	Adam	5000	Testing
E103	Eve	2000	Design
E104	Alice	1500	Design
E105	Mike	2000	Design

Table 2: Two relations obtained from Employee relation.

EmpID	Name	Salary	Dept	TID	Index
E101	John	1000	Testing	3	3
E101	John	100000	Security	2	2
E102	Adam	5000	Testing	5	5
E103	Eve	2000	Design	4	4
E104	Alice	1500	Design	1	1
E105	Mike	2000	Design	6	6

(a) R^1 = Employee1 relation.

CTID	SSTID	Salary
1	1	1500
5	5	5000
3	3	1000
6	6	2000
2	2	100000
4	4	2000

(b) R^2 = Employee2 relation.

server and the user. However, maximum/minimum finding algorithms require at most four communication rounds. Also, Obscure achieves the minimum communication cost for aggregate queries, especially, for count, sum, and average queries, by aggregating data locally at each server. (v) Neither involves the DB owner to verify the results nor requires a trusted-third-party verifier.

2 DATA OUTSOURCING

To outsource a relation R having attributes $A_1, A_2, \ldots, A_m$ and n tuples, the DB owner creates the following two relations R^1 and R^2:

• **Relation** R^1 that consists of all the attributes $A_1, A_2, \ldots, A_m$ with two additional attributes, namely `TID` (tuple-id) and `Index`. `TID` attribute helps in finding tuples having the maximum/minimum/top-k values, and `Index` attribute is used in finding tuples satisfying the query predicate. The i^{th} values of `TID` and `Index` attributes have the *same* and *unique* random number between 1 to n.

• **Relation** R^2 that consists of three attributes `CTID` (cleartext tuple-id), `SSTID` (secret-shared tuple-id), and an attribute, say A_c, on which a comparison operator (minimum, maximum, and top-k) needs to be supported. The i^{th} values of the attributes `CTID` and `SSTID` of the relation R^2 keep the i^{th} value of the `TID` attribute of the relation R^1. The i^{th} value of the attributes A_c of the relation R^2 keeps the i^{th} value of an attribute of the relation R^1 on which the user wants to execute a comparison operator. Further, the tuples of the relations R^2 are randomly permuted.

Example. Consider the `Employee` relation (see Table 1). The DB owner creates R^1 = `Employee1` relation (see Table 2a) with `TID` and `Index` attributes and creates R^2 = `Employee2` relation (see Table 2b) having three attributes `CTID`, `SSTID`, and `Salary`.

Creating secret-shares. Let $A_i[a_j]$ $(1 \leq i \leq m+1$ and $1 \leq j \leq n)$ be the j^{th} value of the attribute A_i. The DB owner creates c secret-shares of each attribute value $A_i[a_j]$ of the relation R^1 using a secret-sharing mechanism that allows string-matching operations at the server, *e.g.*, [4]. However, c shares of the j^{th} value of the attribute A_{m+2} (*i.e.*, `Index`) are obtained using SSS. This will result in c relations: $S(R^1)_1, S(R^1)_2, \ldots, S(R^1)_c$, each with $m+2$ attributes. For R^2, the DB owner creates c secret-shares of each value of `SSTID` using a secret-sharing mechanism that allows string-matching operations on the servers and each value of A_c using OP-SS [5, 6]. The attribute `CTID` is outsourced in cleartext with the shared relation $S(R^2)_i$. Note that an adversary cannot learn by just observing any two tuples of the two relations that whether these tuples share a common value in the attribute `TID`/`SSTID` and A_c or not.

Table 3: An execution of the sum query verification.

Dept	Salary	o values	A_x and f_1	A_y and f_2
Testing	1000	1	1(**200**+1000+1)=1201	1(−**1200**+1000+1)= −199
Security	100000	0	0(**1000**+100000+0)=0	0(−**101000**+100000+0)=0
Testing	5000	1	1(−**5900**+5000+1)= −899	1(**900**+5000+1)=5901
Design	2000	0	0(**2000**+2000+0)=0	0(−**4000**+2000+1)=0
Design	1500	0	0(**500**+1500+0)=0	0(−**2000**+1500+0)=0
Design	2000	0	0(−**2100**+2000+0)=0	0(**100**+2000+0)=0
		2	$\sum f_1$ =302	$\sum f_2$ =5702

3 SUM QUERY

Consider a query: `select sum(`A_ℓ`) from` R `where` $A_1 = v_1 \wedge A_2 = v_2 \wedge \ldots \wedge A_m = v_m$. In the secret-sharing setting, the user transforms the above query into the following query at the j^{th} server: `select sum(`A_ℓ`) from` $S(R^1)_j$ `where` $A_1 = S(v_1)_j \wedge A_2 = S(v_2)_j \wedge \ldots \wedge A_m = S(v_m)_j$. The j^{th} server performs the following operation on each attribute on which the user wants to compute the sum, *i.e.*, A_ℓ and A_q:

$$\sum_{k=1}^{k=n} A_\ell[S(a_k)]_j \times (\prod_{i=1}^{i=m} (A_i[S(a_k)]_j \otimes S(v_i)_j))$$

$\otimes$ 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 $S(v_i)$ against k^{th} value $(1 \leq k \leq n)$ of the attribute A_i and multiplies the resistant with the i^{th} values of the attribute A_ℓ. Finally, the server adds all the values of the A_ℓ 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 sum 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 A_x and A_y, to the relation R^1. The i^{th} values of the attributes A_x and A_y are any two random numbers whose difference equals to $-a_i$, where a_i is the i^{th} value of the attribute A_ℓ. The values of the attributes A_x and A_y are also secret-shared using SSS. For example, in Table 3, boldface numbers show these random numbers of the attribute A_x and A_y in cleartext.

Server. Let `select sum(`A_ℓ`) from` R `where` $A_q = v$ be a query. The server computes two functions, f_1 and f_2, to verify the conditions of sum-query verification in an oblivious manner, as follows:

$$op_1 = f_1(x) = \sum_{i=1}^{i=n} o_i(x_i + a_i + o_i)$$

$$op_2 = f_1(x) = \sum_{i=1}^{i=n} o_i(y_i + a_i + o_i)$$

i.e., the server executes the functions f_1 and f_2 on n values, added in attributes A^x and A^y. In the above equations, o_i is the output of the string-matching operation carried on the i^{th} value of the attribute A_q, and a_i be the i^{th} $(1 \leq i \leq n)$ value of the attribute A_ℓ. Finally, the server sends the following three things to the user: (i) the sum of the resultant values of the attributes A_ℓ, say $\langle sum_\ell \rangle_k$, ($ii$) the sum of the output of the string-matching operations carried on the attribute A_q, say $\langle sum_q \rangle_k$, against the query predicate, and (iii) the sum of outputs of the functions f_1 and f_2, say $\langle sum_{f1f2} \rangle_k$.

User-side. The user interpolates the received three values from each server, which results in $Isum_\ell$, $Isum_q$, and $Isum_{f1f2}$. The user checks the value of $Isum_{f1f2} - 2 \times Isum_q$ and $Isum_\ell$, and if it finds equal, then it implies that the server has correctly executed the sum query.

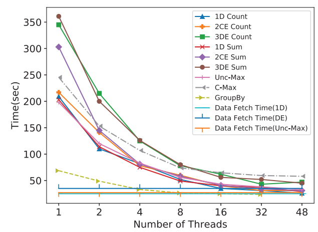
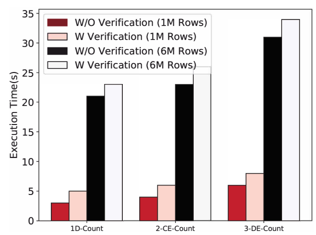
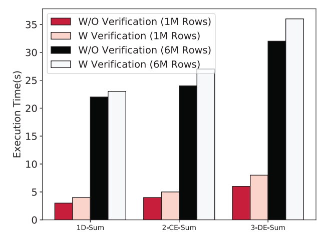


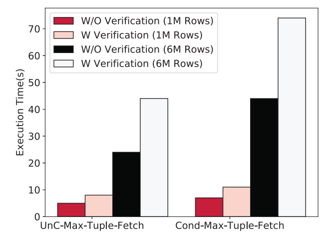
Figure 1: Exp 1. Obscure performance on 6M rows.



(a) Count query.



(b) Sum query.



(c) Maximum tuple fetch query.

Figure 2: Exp 2. Result verification.

4 MAXIMUM QUERY

We provide methods for finding the maximum value and retrieving the corresponding tuples 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 $S(R^1)$ 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, Obscure uses the two relations $S(R^1)$ and $S(R^2)$, which are secured using secret-shared and OP-SS, respectively. In particular, the attribute A_c will exist in the relations $S(R^1)_i$ and $S(R^2)_i$ at the server i. The strategy is to jointly execute a query on the relations $S(R^1)_i$ and $S(R^2)_i$ and obliviously retrieve the entire tuple from $S(R^1)_i$. The server can find the maximum value of the attribute A_c using the relation $S(R^2)$, which is secret-shared using OP-SS, and then, can find the tuple having the maximum value from $S(R^1)$ using string-matching mechanism. Particularly, the i^{th} server executes the following steps:

(1) *On the relation* $S(R^2)_i$. Since secret-shared values of the attribute A_c of relation $S(R^2)_i$ are comparable, the server i finds a tuple $\langle S(t_k), S(value)\rangle_i$ 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* $S(R^1)_i$. Now, the server i performs the join of the tuple $\langle S(t_k), S(value)\rangle_i$ with all the tuples of the relation $S(R^1)_i$ by comparing the tuple-ids (`TID` attribute's values) of the relation $S(R^1)_i$ with $S(t_k)_i$, as follows:

$$\sum_{k=1}^{k=n} A_p[S(a_k)]_i \times (\texttt{TID}[S(a_k)]_i \otimes S(t_k)_i)$$

Where p ($1 \leq p \leq m$) is the number of attributes in the relation R and `TID` is the tuple-id attribute of $S(R^1)_i$. The server i compares the tuple-id $\langle S(t_k)\rangle_i$ with each k^{th} value of the attribute `TID` of $S(R^1)_i$ 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 Suppkey) 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. Obscure, 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 Obscure result verification steps do not incur a significant cost at the servers.

REFERENCES

[1] Stealth Pulsar, available at:http://www.stealthsoftwareinc.com/.
[2] D. W. Archer et al. From keys to databases - real-world applications of secure multi-party computation. *IACR Cryptology ePrint*, 2018.
[3] D. Bogdanov et al. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, volume 5283, pages 192–206, 2008.
[4] S. Dolev et al. Accumulating automata and cascaded equations automata for communicationless information theoretically secure multi-party computation. *Theor. Comput. Sci.*, 795:81–99, 2019.
[5] F. Emekçi et al. Privacy preserving query processing using third parties. In *ICDE*, page 27, 2006.
[6] F. Emekçi et al. Dividing secrets to secure data outsourcing. *Inf. Sci.*, 263:198–210, 2014.
[7] P. Gupta et al. Obscure: Information-theoretic oblivious and verifiable aggregation queries. *PVLDB*, 12(9):1030–1043, 2019.
[8] H. Hacigümüs et al. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*, pages 216–227, 2002.
[9] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.