Private and Secure Secret Shared MapReduce (Extended Abstract)

Shlomi Dolev, Yin Li, Shantanu Sharma

Private and Secure Secret Shared MapReduce*

(Extended Abstract)

Shlomi Dolev[1], Yin Li[2], and Shantanu Sharma[1]

[1] Ben-Gurion University, Israel.
[2] Xinyang Normal University, China.

Abstract. Data outsourcing allows data owners to keep their data in public clouds, which do not ensure the privacy of data and computations. One fundamental and useful framework for processing data in a distributed fashion is MapReduce. In this paper, we investigate and present techniques for executing MapReduce computations in the public cloud while preserving privacy. Specifically, we propose a technique to outsource a database using Shamir secret-sharing scheme to public clouds, and then, provide privacy-preserving algorithms for performing search and fetch, equijoin, and range queries using MapReduce. Consequently, in our proposed algorithms, the public cloud cannot learn the database or computations. All the proposed algorithms eliminate the role of the database owner, which only creates and distributes secret-shares once, and minimize the role of the user, which only needs to perform a simple operation for result reconstructing. We evaluate the efficiency by (*i*) the number of communication rounds (between a user and a cloud), (*ii*) the total amount of bit flow (between a user and a cloud), and (*iii*) the computational load at the user-side and the cloud-side.

1 Introduction

Data and computation outsourcing move databases and computations from a private cloud to a public cloud, which is not under the control of a single user. Thus, the outsourcing results in less burden on a private cloud in terms of the maintenance of databases, infrastructures, and executions of queries. Unfortunately, the ease in storing data and executing computations in the public clouds implies a risk of violating security and privacy of the databases and the computations.

MapReduce [4] provides efficient and fault tolerant parallel processing of large-scale data without dealing with security and privacy of data and computations. The main obstacle for providing privacy-preserving framework for MapReduce in the adversarial (public) clouds is computational and storage efficiency. An adversarial cloud may breach the privacy of data and computations. In this paper, we present techniques for executing MapReduce computations in public cloud while preserving privacy.

* Details appear as a technical report in [7]. We thank Jeffrey Ullman for valuable comments. This work is supported by the Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant 428/11), the Israeli Internet Association, and the Ministry of Science and Technology, Infrastructure Research in the Field of Advanced Computing and Cyber Security.

Motivating examples. We present an example of equijoin to show the need for security and privacy of data and query execution using MapReduce in the public cloud.

Secure and privacy-preserving equijoin of two relations $X(A, B)$ and $Y(B, C)$. *Problem statement*: The join of relations $X(A, B)$ and $Y(B, C)$, where the joining attribute is B, provides output tuples $\langle a, b, c\rangle$, where (a, b) is in A and (b, c) is in C. In the equijoin of $X(A, B)$ and $Y(B, C)$, all tuples of both the relations with an identical value of the attribute B should appear together for providing the final output tuples.

Consider that the relations X and Y belong to two organizations, *e.g.*, a company and a hospital, while a third user wants to perform the equijoin. However, both the two organizations want to provide results while maintaining the privacy of their databases, *i.e.*, without revealing the whole database to the other organization or the user. Hence, it is required to perform the equijoin in a secure and privacy-preserving manner.

Our contributions. We are interested in making a secure and privacy-preserving computation execution and storage-efficient technique for MapReduce computations in the public clouds. Hence, our focus is on *information-theoretically secure data and computation outsourcing technique* and query execution using MapReduce. Specifically, we use Shamir secret-sharing (SSS) [14] for making secret-shares of each tuple of a relation and send them to the clouds. A user can execute her queries using accumulating-automata (AA) [5] on these secret-shares without revealing queries/data to the cloud. We can perform `count` (Section 4.1), `search` and `fetch` operations (Section 4.2) in a privacy-preserving manner. *Due to the space limitation, we omit details of privacy-preserving range selection and equijoin, which may be found in [7].*

Related work. PRISM [2], PIRMAP [12], EPiC [1], MrCrypt [16], and Crypsis [15] provide privacy-preserving MapReduce execution in the cloud on encrypted data. However, all these protocols increase computation time due to dependency on encryption and decryption of data.

The authors [8] provide a privacy-preserving join operation using secret-sharing. However, the approach [8] requires that two different data owners share some information for constructing *an identical share for identical values* in their relations. The authors [9] provide a technique for data outsourcing using a variation of SSS. However, the approach [9] suffers from two major disadvantages, as follows: (*i*) in order to produce an answer to a query, the data owner has to work on all the shares, and hence, the data owner performs a lot of work instead of the cloud; and (*ii*) a third party cannot directly issue any query on secret-shares, and it has to contact with the data owner. In [9], the authors provide a way for constructing polynomials that can maintain the orders of the secrets. However, this kind of polynomial is based on an integer ring (no modular reduction) rather than a finite field; thus, it has potential security risk.

There are some other works [11,10,3] that provide searching operations on secret-shares. In [11], a data owner builds a Merkle hash tree [13] according to a query. In [10], a user knows the addresses of the desired tuples, so they can fetch all those tuples obliviously from the clouds without performing a search operation in the cloud.

To the best of our knowledge, there is no algorithm that (*i*) eliminates the need of a database owner except one time creation and distribution of secret-shares, (*ii*) minimizes the overhead at the user-side, and (*ii*) provides information-theoretically secure MapReduce computations in the cloud. In this paper, we build a technique for data

Algorithms	Communication cost	Computational cost		# rounds	Matching	Based on
		User	Cloud			
`Count` operation						
EPiC [1]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	1	Online	E
Our solution 4.1	$\mathcal{O}(1)$	$\mathcal{O}(1)$	nw	1	Online	SS
`Search` and single tuple `fetch` operation						
Chor et al. [3]	$\mathcal{O}(nmw)$	$\mathcal{O}(1)$	$\mathcal{O}(nmw)$	$log_2 n$	Online	SS
PRISM [2]	$\mathcal{O}((nm)^{\frac{1}{2}}w)$	$\mathcal{O}((nm)^{\frac{1}{2}}w)$	$\mathcal{O}(nmw)$	q		E
Our solution 4.2	$\mathcal{O}(mw)$	$\mathcal{O}(mw)$	$\mathcal{O}(mw)$	1	Online	SS
`Search` and multi-tuples `fetch` operation						
rPIR [10]	$\mathcal{O}(nm)$	$\mathcal{O}(1)$	$\mathcal{O}(nmw)$	1	No	SS
PIRMAP [12]	$\mathcal{O}(nmw)$	$\mathcal{O}(mw)$	$\mathcal{O}(nmw)$	1	No	E
Goldberg [11]	$\mathcal{O}(n+m)$	$\mathcal{O}(m)$	$\mathcal{O}(nm)$	2	Offline	SS
Emekci et al. [9]	$\mathcal{O}(\ell m)$	$\mathcal{O}(\ell m)$	$\mathcal{O}(n)$	2	Offline	vSS
Our solution: knowing addresses 4.2	$\mathcal{O}\big((log_\ell n + log_2 \ell)\ell\big)$	$\mathcal{O}\big((log_\ell n + log_2 \ell)\ell\big)$	$\mathcal{O}\big((log_\ell n + log_2 \ell)\ell nw\big)$	$\lfloor log_\ell n \rfloor + \lfloor log_2 \ell \rfloor + 1$	Online	SS
Our solution: fetching tuples 4.2	$\mathcal{O}((n+m)\ell w)$	$\mathcal{O}((n+\ell m)w)$	$\mathcal{O}(\ell nmw)$	1	Online	SS
Equijoin						
Our solution (see in [7])	$2nwk + 2k\ell^2 mw$	$2nw + 2k\ell^2 mw$	$2\ell^2 kmw$	$2k$	Online	SS

Notations: Online: perform string matching in the cloud. Offline: perform string matching at the user-side. E: encryption-decryption based. SS: Secret-sharing based. vSS: a variant of SS. n: # tuples, m: # attributes, ℓ: # occurrences of a pattern ($\ell \leq n$), w: bit-length of a pattern.

Table 1. Comparison of different algorithms with our algorithms.

and computation outsourcing based on SSS and AA [5]. In addition, our algorithms can *perform a string matching operation on secret-shares in the cloud, without downloading the whole database of the form of secret-shares.* However, most of the existing secret-sharing based privacy-preserving algorithms are unable to do string matching operations in the cloud; see Table 1.

The proposed technique overcomes all the disadvantages of the existing secret-sharing based data outsourcing techniques [8,9,11,10,3]. Thus, there is no need for (*i*) sharing information among different data owners, (*ii*) working at the database owners, except creation and distribution of secret-shares, (*iii*) having an identical share for multiple occurrences of a value, and (*iv*) a third party can directly execute her queries in the clouds without revealing her queries to the clouds.

2 System and Adversarial Settings

We consider, for the first time, data and computation outsourcing of the form of secret-shares to c *non-communicating* clouds that they do not exchange data with each other, only exchange data with the user or the database owner.

The system architecture. The architecture is simple but powerful and assumes the following:

STEP 1. A data owner outsources her databases of the form of secret-shares to c (non-communicating) clouds only once; see STEP 1 in Fig. 1. We use c clouds to provide privacy-preserving computations. Note that a single cloud cannot provide privacy-preserving computations using secret-sharing.

STEP 2. A preliminary step is carried out at the user-side who wants to perform a MapReduce computation. The user sends a query of the form of secret-shares to all c clouds to find the desired result of the form of secret-shares; see STEP 2 in Fig. 1. The query must be sent to at least $c' < c$ number of clouds, where c' is the threshold of SSS.

STEP 3. The clouds deploy a *master process* that executes the computation by assigning the *map tasks* and the *reduce tasks*; see STEP 3 in Fig. 1. The user interacts only with the master process in the cloud, and the master process provides the addresses of the outputs to the user. It must be noted that the communication between the user and the clouds is presumed to be the same as the communication between the user and the master process.

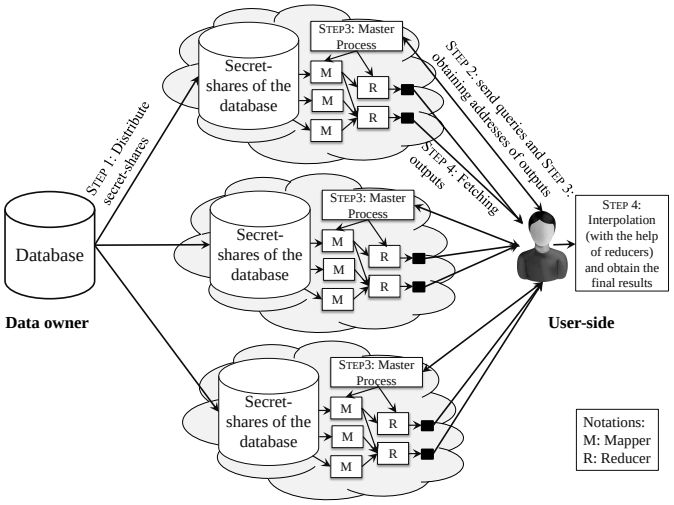

Fig. 1. The system architecture.

STEP 4. The user fetches the outputs from the clouds and performs interpolation (with the help of reducers) for obtaining the secret-values; see STEP 4 in Fig. 1.

Adversarial Settings. We assume, on one hand, that an adversary cannot launch any attack against the data owner. Also, the adversary cannot access the secret-sharing algorithm and machines at the database owner side. On the other hand, an adversary can access public clouds and data stored therein. A user who wants to perform a computation on the data stored in public clouds may also behave as an adversary. Moreover, the cloud itself can behave as an adversary, since it has complete privileges to all the machines and storage. Both the user and the cloud can launch any attack for compromising the privacy of data or computations. We consider an honest-but-curious adversary, which performs assigned computations correctly, but tries to breach the privacy of data or MapReduce computations. However, such an adversary does not modify or delete information from the data. We assume that an adversary can know less than $c' < c$ clouds locations that store databases and execute queries. In addition, the adversary cannot eavesdrop all the c' or c channels (between the database owner and the clouds, and between the user and the clouds). Hence, we do not impose private communication channels. Under such an adversarial setting, we provide a guaranteed solution so that an adversary cannot learn the data or computations. It is important to mention that an

adversary can break our protocols by colluding c' clouds, which is the threshold for which the secret sharing scheme is designed for.

Parameters for analysis. We analyze our privacy-preserving algorithms on the following parameters: (*i*) *communication cost*: is the sum of all the bits that are required to transfer between a user and a cloud; (*ii*) *computational cost*: is the sum of all the bits over which a cloud or a user works; and (*iii*) *number of rounds*: shows how many times a user communicates with a cloud for obtaining the results.

3 Creation and Distribution of Secret-Shares of a Relation

Assume that a database only contains English words. Since the English alphabets consist of 26 letters, each letter can be represented as a unary vector with 26 bits. Hence, the letter 'A' is represented as $(1_1, 0_2, 0_3, \ldots, 0_{26})$, where the subscript represents the position of the letter; since 'A' is the first letter, the first value in the vector is one and others are zero. Similarly, 'B' is $(0_1, 1_2, 0_3, \ldots, 0_{26})$, and so on.

The reason of using unary representation here is that it is very easy for verifying two identical letters. The expression $S = \sum_{i=0}^{r} u_i \times v_i$, compares two letters, where $(u_0, u_1, \cdots u_r)$ and $(v_0, v_1, \cdots, v_r)$ are two unary representations. It is clear that whenever any two letters are identical, S is equal to one; otherwise, S is equal to zero. Binary representation can also be accepted, but the comparison function is different from that used in the unary representation [6].

A secure way for creating *secret-shares*. When outsourcing a vector to the clouds, we use SSS and make secret-shares of every bit by selecting different polynomials of an identical degree. For example, we create secret-shares of the vector of 'A' $((1_1, 0_2, 0_3, \ldots, 0_{26}))$ by using 26 polynomials of an identical degree, since the length of the vector is 26. Following that, we can create secret-shares for all the other letters and distribute them to different clouds.

Since we use SSS, a cloud cannot infer a secret. Moreover, it is important to emphasize that we use *different* polynomials for creating secret-shares of each letter, thereby multiple occurrences of a word in a database have different secret-shares. Therefore, a cloud is also unable to know the total occurrences of a word in the whole database.

Secret-shares of numeral values. We follow the similar approach for creating secret-shares of numeral values as used for alphabets. In particular, we create a unary vector of length 10 and put all the values 0 except only 1 according to the position of a number. For example, '1' becomes $(1_1, 0_2, \ldots, 0_{10})$. After that, use SSS to make secret-shares of every bit in each vector by selecting different polynomials of an identical degree for each number, and send them to multiple clouds.

4 Privacy-Preserving Query Processing on Secret-Shares using MapReduce in the Clouds

4.1 Count Query

We present a privacy-preserving algorithm for counting the number of occurrences of a pattern, p, in the cloud; throughout this section, we denote a pattern by p. This algorithm is divided into two phases, as: PHASE 1: Privacy-preserving counting in the clouds and PHASE 2: Result reconstruction at the user-side.

In short, we apply a string matching algorithm, which is done using AA that compares each value of a relation with p. If a value and p match, it will results in 1; otherwise, we have 0. We apply the same algorithm on each value and accumulate all one that provide the number of occurrences of p. Note that all the values of a relation, a pattern, and the result, *i.e.*, 0 or 1, are of the form of secret-share.

Working at the user-side. A user creates unary vectors for each letter of p. In order to hide the vectors of p, the user creates secret-shares of each vector of p, as suggested in Section 3, sends them to c clouds. In addition, the user sends length (x) of p and the attribute of the relation (m') where to count p, to c clouds.

Working in the cloud. Now, a cloud has two things, as: (*i*) a relation of the form secret-shares, and (*ii*) a searching pattern of the form of secret-shares with its length, x. In order to count the number of occurrences of p, the mapper in the cloud performs $x + 1$ steps, see Table 2, for comparing the pattern with each value of the specified attribute of the relation.

At this time, the mapper is unable to know the value of the node N_{x+1} in each iteration and sends the final value of N_{x+1} to the user of form of a $\langle key, value \rangle$ pair, where a key is an identity of an input split over which the operation has performed, and the corresponding $value$ is the final value of the node N_{x+1} of the form of secret-shares. The user collects $\langle key, value \rangle$ pairs from all the clouds or a sufficient number of clouds such that the secret can be generated using those shares.

STEP 1: $N_1 = 1$, $N_x^0 = 0$
STEP 2: $N_2^{(i)} = N_1 \times v_1$
STEP 3: $N_3^{(i)} = N_2^{(i)} \times v_2$
$\vdots$
STEP $x+1$: $N_x^{(i)} = N_x^{(i-1)} + N_{x-1}^{(i)} \times v_{x-1}$
The notation $N_j^{(i)}$ shows that the node j is executing a step in iteration i. The final value of the node N_{x+1}, which is sent to the user, is the number of occurrences of the pattern.

Table 2. The steps executed by a mapper for a pattern of length x.

Result reconstruction at the user-side. We need to reconstruct the final value of the node N_{x+1}. The user has $\langle key, value \rangle$ pairs from all the clouds. All the values corresponding to a key are assigned to a reducer that performs Lagrange interpolation and provides the final value of the node N_{x+1}. If there are more than one reducer, then after the interpolation the sum of the final values shows the total occurrences of p.

Aside. If a user searches `John` in a database containing names like 'John' and 'Johnson,' then our algorithm will show two occurrences of `John`. However, it is a problem associated with string matching. In order to search a pattern precisely, we may use the terminating symbol for indicating the end of the pattern.

4.2 Search and Fetch Queries

In this section, we provide a privacy-preserving algorithm for fetching all the tuples containing p. The proposed algorithms first count the number of tuples containing p, and then, fetch all the tuples after obtaining their addresses. Specifically, we provide 2-phased algorithms, where: PHASE 1: Finding addresses of tuples containing p, and PHASE 2: Fetching all the tuples containing p.

Unary occurrence of a pattern. When only one tuple contains p, there is no need to obtain the address of the tuple, and hence, we fetch the whole tuple in a privacy-preserving manner. Here, we explain how to fetch a single tuple containing p.

Fetching the tuple. The user sends secret-shares of p. The cloud executes a map function on a specific attribute, and the map function matches p with i^{th} value of the attribute. Consequently, the map function results in either 0 or 1 of the form of secret-shares, if p matches the i^{th} value of the attribute, then the result is 1. After that the map function multiplies the result (0 or 1) to all the m values of the i^{th} tuple. In this manner, the map function creates a relation of n tuples and m attributes. When the map function finishes over all the n tuples, it adds and sends all the secret-shares of each attribute, as: $S_1||S_2||\ldots||S_m$ to the user, where S_i is the sum of the secret-shares of i^{th} attribute. The user on receiving shares from all the clouds executes a reduce function that performs interpolation and provides the desired tuple containing p.

Aside. When we multiply the output of the string matching operation, which is of the form of secret-shares, to all the values in a tuple, it results in all the value of the tuple either 0 or 1 of the form of secret-shares. Thus, the sum of all the secret-shares of an attribute results in only the value of the attribute corresponding to the tuple containing p. By performing identical operations on each tuple and finally adding all the secret-shares of each attribute, the cloud is unable to know which tuple is fetched.

Multiple occurrences of a pattern. When multiple tuples contain p, we cannot fetch all those tuples obliviously without obtaining their addresses. Therefore, we first need to perform a pattern search algorithm to obtain the addresses of all the tuples containing p, and then, fetch the tuples in a privacy-preserving manner. Throughout this section, we consider that ℓ tuples contain p. This algorithms has 2-phases, as follow: PHASE 1: Finding the addresses of the desired ℓ tuples, and PHASE 2: Fetching all the ℓ tuples.

Tree-based algorithm. We propose a search-tree-based keyword search algorithm that consists of two phases, as: finding the address of the desired ℓ tuples in multiple rounds, and then, fetching all the ℓ tuples in one more round. We can also obtain the addresses (or line numbers) in a privacy-preserving manner, if there is only a tuple contains p. Thus, for the case of finding addresses of ℓ tuples containing p, we divide the whole relation into certain blocks such that each block belongs to one of the following cases:

1. A block contains no occurrence of p, and hence, no fetch operation is needed.
2. A block contains one/multiple tuples but only a single tuple contains p.
3. A block contains h tuples, and all the h tuples contain p.
4. A block contains multiple tuples but fewer tuples contain p.

Finding addresses. We follow an idea of partitioning the database and counting the occurrences of p in the partitions, until each partition satisfies one of the above mentioned cases. Specifically, we initiate a sequence of Query & Answer (Q&A) rounds. In the first round of Q&A, we count occurrences of p in the whole database (or in an assigned input split to a mapper) and then partition the database into ℓ blocks, since we assumed that ℓ tuples contain p. In the second round, we again count occurrences of p in each block and focus on the blocks satisfying Case 4. There is no need to consider the blocks satisfying Case 2 or 3, since we can apply the algorithm given for unary occurrence of a pattern, in both the cases. However, if the multiple tuples of a block in the second round contain p, *i.e.*, Case 4, we again partition such a block until it satisfies either Case 1, 2 or 3. After that, we can obtain the addresses of the related tuples using the method similar to the algorithm given for unary occurrence of a pattern.

Fetching tuples. We use the approach described in the naive algorithm for fetching multiple tuples after obtaining the addresses of the tuples.

5 Conclusion

MapReduce provides efficient large-scale data processing without dealing with the privacy and security of data and computations. In order to avoid overheads for maintaining and executing queries at the database owner side, a database is outsourced to untrusted public clouds that can reveal the database and computations. We proposed a new information-theoretically secure data and computation outsourcing technique. By the proposed techniques, users can execute their computations in the public cloud without any need of the database owner, and the cloud cannot learn the database or the computations. We provided MapReduce based privacy-preserving algorithms to execute `count`, `search`, and `fetch` quires in the public clouds. Due to the space limitation, privacy-preserving range queries and equijoin are presented in [7]. As compared to the existing algorithms, our algorithms provide perfect privacy protection without introducing computation and communication overhead.

References

1. E. Blass and et al. EPiC: efficient privacy-preserving counting for MapReduce, 2012.
2. E.-O. Blass, R. D. Pietro, R. Molva, and M. Önen. PRISM - Privacy-Preserving Search in MapReduce. In *Privacy Enhancing Technologies*, pages 180–200, 2012.
3. B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. *IACR Cryptology ePrint Archive*, 1998:3, 1998.
4. J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150. USENIX Association, 2004.
5. S. Dolev, N. Gilboa, and X. Li. Accumulating automata and cascaded equations automata for communicationless information theoretically secure multi-party computation: Extended abstract. In *SCC*, pages 21–29, 2015.
6. S. Dolev and Y. Li. Secret shared random access machine. *IACR Cryptology ePrint Archive*, 2015:292, 2015.
7. S. Dolev, Y. Li, and S. Sharma. Private and secure secret shared MapReduce. Technical Report 16-01, Department of Computer Science, Ben-Gurion University of the Negev, 2016. Available at: `https://www.cs.bgu.ac.il/~frankel/reports.html`.
8. F. Emekçi, D. Agrawal, A. El Abbadi, and A. Gulbeden. Privacy preserving query processing using third parties. In *ICDE*, page 27, 2006.
9. F. Emekçi, A. Metwally, D. Agrawal, and A. El Abbadi. Dividing secrets to secure data outsourcing. *Inf. Sci.*, 263:198–210, 2014.
10. L. Li, M. Militzer, and A. Datta. rPIR: Ramp secret sharing based communication efficient private information retrieval. *IACR Cryptology ePrint Archive*, 2014:44, 2014.
11. W. Lueks and I. Goldberg. Sublinear scaling for multi-client private information retrieval. In *FC*, pages 168–186, 2015.
12. T. Mayberry, E. Blass, and A. H. Chan. PIRMAP: efficient private information retrieval for MapReduce. In *FC*, pages 371–385, 2013.
13. R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378, 1987.
14. A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
15. J. J. Stephen, S. Savvides, R. Seidel, and P. Eugster. Practical confidentiality preserving big data analysis. In *HotCloud*, 2014.
16. S. D. Tetali, M. Lesani, R. Majumdar, and T. D. Millstein. MrCrypt: static analysis for secure cloud computations. In *OOPSLA*, pages 271–286, 2013.