Secure Grouping and Aggregation with MapReduce

Radu Ciucanu[1], Matthieu Giraud[1], Pascal Lafourcade[1] and Lihua Ye[2]
[1]*LIMOS, Université Clermont Auvergne, Aubière, France*
[2]*Harbin Institute of Technology, Harbin, Weihai, Shenzhen, China*

Keywords: Database Queries, MapReduce, Security, Grouping, Aggregation.

Abstract: MapReduce programming paradigm allows to process big data sets in parallel on a large cluster. We focus on a scenario where the data owner outsources her data on an *honest-but-curious* server. Our aim is to evaluate grouping and aggregation with SUM, COUNT, AVG, MIN, and MAX operations for an authorized user. For each of these five operations, we assume that the public cloud provider and the user do not collude i.e., the public cloud does not know the secret key of the user. We prove the security of our approach for each operation.

1 INTRODUCTION

We address the fundamental problem of how to group and aggregate data from a relation in a privacy-preserving manner using MapReduce. We assume that the data is externalized in the cloud by the data owner and there is a user that queries it. We consider the following five aggregation operations, which are precisely those included in the SQL standard: SUM, COUNT, AVG, MIN, and MAX.

We start by a running example to present the concepts of grouping and aggregation, and of MapReduce computations. Then, we present our problem statement and illustrate with the same example the privacy issues related to grouping and aggregation with MapReduce.

Example 1. *Assume there is a university storing the relation R corresponding to the list of professors with their associated department and salary. The grouping and aggregation operation on the relation R, in the case where we assume one group attribute and one aggregate function, is denoted by $\gamma_{A,\theta(B)}(R)$, where A is the grouping attribute and θ is one of the five aggregation operations applied on the attribute B different from the grouping attribute. In this example (Figure 1), we consider the attribute "Department" as the grouping attribute and* SUM *is the aggregation operation applied on attribute "Salary". Hence, for each department we sum all the associated salaries. Since Alice and Bob are in the Computer Science department, the sum of salaries associated to the Computer Science department is* $1900 + 1800 = 3700$. *In the same way, we sum the salaries of Mallory and Oscar from the Mathematics department. Since Eve is the only one in the Physics department, the sum corresponds to the salary of Eve which is equal to* 2000. *For the query* $\gamma_{Department,\mathsf{SUM}(Salary)}(R)$, *we obtain the relation presented in Figure 2. Aggregation operations* COUNT, AVG, MIN, *or* MAX *work similarly.*

Name	Department	Salary
Alice	Computer Science	1900
Mallory	Mathematics	1750
Bob	Computer Science	1800
Eve	Physics	2000
Oscar	Mathematics	1600

Figure 1: Relation R.

Department	SUM (salary)
Computer Science	3700
Physics	2000
Mathematics	3350

Figure 2: Result of $\gamma_{\text{Department},\mathsf{SUM}(\text{Salary})}(R)$.

Grouping-and-aggregation with MapReduce. An algorithm to perform grouping and aggregation with MapReduce is presented in Chapter 2 of (Leskovec et al., 2014). First, a set of nodes has chunks of the relation. The *map* function creates for each tuple a *key-value* pair where *key* is equal to the value of the grouping attributes in the considered tuple, and *value* is equal to the value of the aggregation attribute of the considered tuple. Then, the key-value pairs are grouped by key, i.e., key-value pairs output by the map phase which have the same key are sent to the same reducer. For each key, the *reduce* function ap-

Ciucanu, R., Giraud, M., Lafourcade, P. and Ye, L.
Secure Grouping and Aggregation with MapReduce.
DOI: 10.5220/0006843803480355
In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE 2018) - Volume 2: SECRYPT*, pages 348-355
ISBN: 978-989-758-319-3

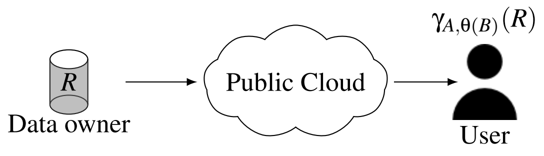

Figure 3: The system architecture.

plies the aggregate function on the associated values of the considered key.

Example 2. *Following Example 1, we perform grouping and aggregation with MapReduce on the relation R where the grouping attribute is the attribute "Department", the aggregation attribute is the attribute "Salary", and the operation is the* SUM. *We start grouping and aggregation with MapReduce by applying the map function. Since the grouping attribute is the attribute "Department" and that the aggregation attribute is the attribute "Salary", the map function emits the pairs* $(Computer\,Science, 1900)$, $(Mathematics, 1750)$, *(Computer Science, 1800),* $(Physics, 2000)$, *and* $(Mathematics, 1600)$. *Pairs sharing the same key (i.e., same value of the grouping attribute) are sent on the same reducer. Then, the reduce function performs on each reducer the aggregation, consisting here of the sum, and we obtain the pairs* $(Computer\,Science, 3700)$ *since* $1900 + 1800 = 3700$, *etc. We present the final result in Figure 2.*

Problem Statement. We assume three participants: the data owner, the public cloud and the user (presented in Figure 3). The data owner stores a relation R in the distributed file system of some public cloud provider. A user (who does not know the relation R) is authorized to perform a grouping and aggregation operation on R.

We assume that the public cloud is *honest-but-curious*, i.e., it executes dutifully the computation task but tries to learn the maximum of information on tuples of R. In order to preserve the privacy of the data owner, the cloud should not learn any plain input data, contrary to what happens for standard algorithms as found in Chapter 2 from (Leskovec et al., 2014) and exemplified above.

We assume that the relation R is initially spread over a set $\mathcal{R}$ of nodes, each of them storing a chunk of R i.e., a set of elements of R. The final result $\gamma_{A,\theta(B)}(R)$ is spread over a set of nodes $\mathcal{Q}$ before it is sent to the user's nodes $\mathcal{U}$. We expect that none of the nodes in $\mathcal{Q}$ can learn any information about relation R, or about the final result.

Notice that a straightforward solution would require the use of a fully homomorphic encryption scheme e.g., (Gentry, 2009). Indeed, a fully homomorphic encryption scheme would allow to execute directly in the encrypted domain all operations needed for computing a grouping and aggregation operation. Unfortunately, such an approach would solve our problem only from a theoretical point of view because making a fully homomorphic encryption scheme work in practice remains an open question (as noted e.g., in (Gentry, 2009)).

Contributions. We revisit the standard algorithms for MapReduce grouping and aggregation (as found in Chapter 2 from (Leskovec et al., 2014)) to guarantee the privacy of the data owner. More precisely, neither the public cloud nor the user learn information about the input data that belongs to the data owner. Our approach, denoted SP for Secure-Private, works for each of the considered five aggregation operations. In each case, the SP approach is efficient from both computational and communication points of view, in the sense that the overhead is linear for each of the two complexity measures.

Our technique is essentially based on two encryption schemes: (i) the well-known Paillier's cryptosystem (Paillier, 1999), which is partially homomorphic i.e., it is additive homomorphic for COUNT, SUM, and AVG operations, and (ii) the order-preserving symmetric encryption scheme (Agrawal et al., 2004) for MIN and MAX operations.

We summarize in Figure 4 the trade-offs between computation cost and communication cost for our SP approach vs the standard MapReduce approach for grouping and aggregation for the five studied operations. In our communication cost analysis, we measure the total size of the data that is emitted from a map or reduce node.

Related Work. Since the seminal MapReduce paper (Dean and Ghemawat, 2004), different protocols have been proposed to perform operations in a privacy-preserving manner (Derbeko et al., 2016) such as search (Blass et al., 2012) (Mayberry et al., 2013), count (Vo-Huu et al., 2015), matrix multiplication (Bultel et al., 2017) or joins (Dolev et al., 2016).

Chapter 2 of (Leskovec et al., 2014) presents an introduction to the MapReduce paradigm. In particular, it includes the MapReduce algorithm for grouping and aggregation that we enhance with privacy guarantees. Very few approaches address the privacy preserving execution for grouping and aggregation operations in MapReduce, and moreover they have different assumptions than we do.

(Bonawitz et al., 2017) provides a technique to compute secure aggregation, while relying on Shamir's secret sharing (Shamir, 1979) to compute

Alg.	*Approach*	*Comp. cost (big-O)*	*Comm. cost (big-O)*
COUNT	Standard	$(1+C_+)n$	$2n$
	SP	$(C_f+2C_{\mathcal{E}}+C_\times)n$	$3n$
SUM	Standard	$(1+C_+)n$	$2n$
	SP	$(C_f+2C_{\mathcal{E}}+C_\times)n$	$3n$
AVG	Standard	$(1+2C_+ +C_\div)n$	$2n$
	SP	$(C_f+3C_{\mathcal{E}}+2C_\times)n$	$3n$
MIN/MAX	Standard	$(1+C_{\mathsf{comp}})n$	$2n$
	SP	$(C_f+C_{\mathsf{E^{ope}}}+3C_{\mathcal{E}}+C_{\mathcal{D}}+C_{\mathsf{comp}})n$	$3n$

Figure 4: Summary of results. Let n be the number of tuples in the relation R. Let C_+ (resp. $C_\times$, $C_\div$, C_f, $C_{\mathsf{E^{ope}}}$ $C_{\mathcal{E}}$, $C_{\mathcal{D}}$) is the cost of addition (resp. multiplication, division, pseudo-random function evaluation, order-preserving symmetric encryption, asymmetric encryption, and asymmetric decryption) and 1 represents the cost to access to one tuple in the relation.

the sum of values coming from different sources. Similarly, (Alghamdi et al., 2017) provides a technique to compute secure aggregation for wireless sensor networks. Contrary to us, these two approaches do not consider the MapReduce paradigm and they cannot be easily adapted for MapReduce because values of shared attributes are encrypted in a non-deterministic way. This is not a suitable choice for MapReduce keys that need to be equal in order to aggregate the key-value pairs on the same reducer.

(Dolev et al., 2016) proposed a technique for executing MapReduce computations in the public cloud while preserving data owner privacy. They use the Shamir's secret sharing and accumulating automata (Dolev et al., 2015). Among the five aggregations studied in this paper, they support only the count, whose computation is done on secret-shares in the public cloud, and at the end, the user performs the interpolation on the outputs. On the other hand, in our setting, the user has only to decrypt the final query result, contrary to the need of doing interpolations in (Dolev et al., 2015).

On the other hand, substantial works has been done on privacy-preserving functional queries on traditional rational database. Popa et al. (Popa et al., 2011) designed *CryptDB* a system allowing a user to execute queries over encrypted data. The authors consider two threats. The first threat is a curious database administrator who tries to learn private data while the second threat is an adversary that gains complete control of application. In (Macedo et al., 2017), authors proposed a generic framework called *SafeNoSQL* to compute in a privacy-preserving manner on NoSQL databases. This framework has a modular and extensible design that enables data processing over multiple cryptographic techniques applied on the same database schema. Contrary to us, these two approaches do not consider the MapReduce paradigm.

To the best of our knowledge, we are the first to propose secure algorithms for grouping and aggregation computation with the MapReduce paradigm where the public cloud performs all the computations and where the user has only to decrypt the result sent by the cloud.

Outline. We introduce some preliminary notions in Section 2. Then, we present our SP approach for these five operations in Section 3. Finally, we outline conclusions and future work in Section 4.

2 PRELIMINARIES

2.1 Relational Algebra

A relation R is a set of n tuples. For a tuple $t \in R$, by $\pi_X(t)$ we denote the projection of the tuple t on the attributes X i.e., the tuple obtained from t after removing all attributes values that are not in X.

By $\gamma_{\mathcal{A},\theta(B)}(R)$ we denote the grouping and aggregation operation on R, where $\mathcal{A}$ is the set of attributes on which we group, B is the attribute for which we apply the aggregation function, and θ is an aggregation function (SUM, COUNT, AVG, MIN, MAX).

2.2 Grouping and Aggregation with MapReduce

We recall the MapReduce algorithms for grouping and aggregation algorithms, as found in Chapter 2 of (Leskovec et al., 2014): for COUNT in Figure 5(a), for SUM in Figure 5(b), for AVG in Figure 5(c), and for MIN in Figure 5(d). The algorithm for MAX is very similar to the one for MIN and we omit it.

2.3 Cryptographic Tools

We present definitions of the cryptographic tools used in our protocols: negligible function, pseudo-random function, order-preserving encryption scheme, and public key encryption scheme.

```
Map function:
Input: $(key, value)$
// key: id of a chunk of $R$
// value: collection of $t \in R$
foreach $t \in R$ do
    emit $(\pi_{\mathcal{A}}(t), 1)$.
Reduce function:
Input: $(key, values)$
// key: $\pi_{\mathcal{A}}(t)$ for $t \in R$
// values: collection of 1
count $\leftarrow 0$;
foreach $1 \in values$ do
    count $\leftarrow$ count $+ 1$;
emit $(\pi_{\mathcal{A}}(t), \mathsf{count})$.
```

(a) COUNT operation.

```
Map function:
Input: $(key, value)$
// key: id of a chunk of $R$
// value: collection of $t \in R$
foreach $t \in R$ do
    emit $(\pi_{\mathcal{A}}(t), \pi_B(t))$.
Reduce function:
Input: $(key, values)$
// key: $\pi_{\mathcal{A}}(t)$ with $t \in R$
// values: collection of $\pi_B(t)$ with $t \in R$
sum $\leftarrow 0$
foreach $\pi_B(t) \in values$ do
    sum $\leftarrow$ sum $+ \pi_B(t)$;
emit $(\pi_{\mathcal{A}}(t), \mathsf{sum})$.
```

(b) SUM operation.

```
Map function:
Input: $(key, value)$
// key: id of a chunk of $R$
// value: collection of $t \in R$
foreach $t \in R$ do
    emit $(\pi_{\mathcal{A}}(t), \pi_B(t))$.
Reduce function:
Input: $(key, values)$
// key: $\pi_{\mathcal{A}}(t)$ for $t \in R$
// values: collection of $\pi_B(t)$
cpt $\leftarrow 0$;
sum $\leftarrow 0$;
foreach $\pi_B(t) \in values$ do
    cpt $\leftarrow$ cpt $+ 1$;
    sum $\leftarrow$ sum $+ \pi_B(t)$;
emit $(\pi_{\mathcal{A}}(t), \mathsf{sum}/\mathsf{cpt})$.
```

(c) AVG operation.

```
Map function:
Input: $(key, value)$
// key: id of a chunk of $R$
// value: collection of $t \in R$
foreach $t \in R$ do
    emit $(\pi_{\mathcal{A}}(t), \pi_B(t))$.
Reduce function:
Input: $(key, values)$
// key: $\pi_{\mathcal{A}}(t)$ for $t \in R$
// values: collection of $\pi_B(t)$
min $\xleftarrow{\$} values$;
foreach $\pi_B(t) \in values$ do
    if $\pi_B(t) <$ min then
        min $\leftarrow \pi_B(t)$;
emit $(\pi_{\mathcal{A}}(t), \mathsf{min})$.
```

(d) MIN operation.

Figure 5: Grouping and aggregation with MapReduce for COUNT, SUM, AVG, MIN operations.

Definition 1 (Negligible function)**.** *A function $\varepsilon : \mathbb{N} \to \mathbb{N}$ is negligible in η if for every positive polynomial $p(\cdot)$ and sufficiently large η, $\varepsilon(\eta) < 1/p(\eta)$.*

Definition 2 (Pseudo-random function)**.** *A function $f\colon \{0,1\}^\eta \times \{0,1\}^{n_0} \to \{0,1\}^{n_1}$ is a pseudo-random function if it is calculable in polynomial time in η and if for all polynomial-size algorithm $\mathcal{B}$,*

$$\left|\Pr\left[\mathcal{B}^{f_k(\cdot)} = 1\colon k \xleftarrow{\$} \{0,1\}^\eta\right] - \Pr\left[\mathcal{B}^{g(\cdot)} = 1\colon g \xleftarrow{\$} \mathsf{Func}[n_0, n_1]\right]\right| \leqslant \varepsilon(\eta)\ ,$$

where $\varepsilon(\cdot)$ is a negligible function in η, Func *is the space functions from domain $\{0,1\}^{n_0}$ to domain $\{0,1\}^{n_1}$, and the probabilities are taken over the choice of k and g.*

Definition 3 (Order-Preserving Symmetric Encryption (Agrawal et al., 2004))**.** *Let η be a security parameter. An order-preserving encryption (OPE) scheme is defined by three algorithms $(\mathsf{G}^{\mathsf{ope}}, \mathsf{E}^{\mathsf{ope}}, \mathsf{D}^{\mathsf{ope}})$:*

$\mathsf{G}^{\mathsf{ope}}(\eta)$**:** *returns a secret key K.*

$\mathsf{E}^{\mathsf{ope}}_K(m)$**:** *returns a new key K' and a ciphertext c.*

$\mathsf{D}^{\mathsf{ope}}_K(c)$**:** *returns the plaintext m.*

such that for any two ciphertexts c_1 and c_2 with corresponding messages m_1 and m_2 we have $c_1 < c_2$ if and only if $m_1 < m_2$.

Definition 4 (Public Key Encryption (PKE))**.** *Let η be a security parameter. A public key encryption (PKE) scheme is defined by three algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$:*

$\mathcal{G}(\eta)$**:** *returns a public/private key pair (pk, sk).*

$\mathcal{E}_{\mathsf{pk}}(m)$**:** *returns the ciphertext c.*

$\mathcal{D}_{\mathsf{sk}}(c)$**:** *returns the plaintext m.*

In the following, we require an additive homomorphic encryption scheme to secure the grouping and aggregation with MapReduce. There exists several schemes that have this property (Okamoto and Uchiyama, 1998; Paillier, 1999; Naccache and Stern, 1998). We choose Paillier's cryptosystem (Paillier, 1999) to illustrate specific required homomorphic properties. Our results and proofs are generic, since any other encryption schemes having such properties can be used instead of Paillier's scheme.

We recall the key generation, the encryption and decryption algorithms.

Key Generation. We denote by $\mathbb{Z}_n$, the ring of integers modulo n and by $\mathbb{Z}_n^*$ the set of invertible elements of

$\mathbb{Z}_n$. The public key pk of Paillier's encryption scheme is (n,g), where $g \in \mathbb{Z}^*_{n^2}$ and $n = p \times q$ is the product of two prime numbers such that $gcd(p,q) = 1$. The corresponding private key sk is (λ,μ), where λ is the least common multiple of $p-1$ and $q-1$ and $\mu = (L(g^{\lambda} \mod n^2))^{-1} \mod n$, where $L(x) = \frac{x-1}{n}$.
Encryption Algorithm. Let m be a message such that $m \in \mathbb{Z}_n$. Let g be an element of $\mathbb{Z}^*_{n^2}$ and r be a random element of $\mathbb{Z}^*_n$. We denote by $\mathcal{E}_{\mathsf{pk}}$ the encryption function that produces the ciphertext c from a given plaintext m with the public key $\mathsf{pk} = (n,g)$ as follows: $c = g^m \times r^n \mod n^2$.
Decryption Algorithm. Let c be the ciphertext such that $c \in \mathbb{Z}_{n^2}$. We denote by $\mathcal{D}_{\mathsf{sk}}$ the decryption function of the plaintext c with the secret key $\mathsf{sk} = (\lambda,\mu)$ defined as follows: $m = L\left(c^{\lambda} \mod n^2\right) \times \mu \mod n$.
Homomorphic Addition of Plaintexts. Paillier's cryptosystem is a partial homomorphic encryption scheme. Let m_1 and m_2 be two plaintexts in $\mathbb{Z}_n$. The product of the two associated ciphertexts with the public key $\mathsf{pk} = (n,g)$, denoted $c_1 = \mathcal{E}_{\mathsf{pk}}(m_1) = g^{m_1} \times r_1^n \mod n^2$ and $c_2 = \mathcal{E}_{\mathsf{pk}}(m_2) = g^{m_2} \times r_2^n \mod n^2$, is the encryption of the sum of m_1 and m_2.

$$\begin{aligned}\mathcal{E}_{\mathsf{pk}}(m_1) \quad \times \quad & \mathcal{E}_{\mathsf{pk}}(m_2) \\ &= c_1 \times c_2 \mod n^2 \\ &= (g^{m_1} \times r_1^n) \times (g^{m_2} \times r_2^n) \mod n^2 \\ &= \left(g^{m_1+m_2} \times (r_1 \times r_2)^n\right) \mod n^2 \\ &= \mathcal{E}_{\mathsf{pk}}(m_1 + m_2 \mod n) \,.\end{aligned}$$

3 SECURE PRIVATE APPROACH

We present our SP approach for the COUNT, SUM, AVG, MIN, and MAX aggregation functions with MapReduce. We denote respectively these five protocols: SP-COUNT, SP-SUM, SP-AVG, SP-MIN, and SP-MAX. The algorithm for SP-MAX is very similar to SP-MIN and we omit it to avoid redundancy.

3.1 SP Protocols

To avoid the cloud to learn the content of the relation R, the data owner protects it before the outsourcing. We denote the protected relation by $\hat{R}$.

The data owner protects the relation using a pseudo-random function with her secret key k and by applying it on values of grouping attributes of each tuples of the relation R. These deterministic pseudo-random function evaluations allow the cloud to perform equality tests between values of grouping attributes. Moreover, the data owner encrypts each value of the aggregation attribute either with Paillier's scheme (using the user public key pk_u) or the OPE scheme (using the shared secret key K between the data owner and the user), depending on the aggregation function. We present the preprocessing phase in Algorithm 1, where E represents either the Paillier encryption (in the case of COUNT, SUM, AVG operations) or the OPE encryption (in the case of MIN and MAX operations). We stress that A^f and $A^{\mathcal{E}}$ are just notations making explicit the correspondences between initial and outsourced data and that $\hat{\mathbb{R}}$ is the schema of $\hat{R}$. For instance, if a relation R has two attributes such that "Name" is the grouping attribute and "Age" is the aggregation attribute, then $\hat{R}$ has attributes "Name^f", "Name$^{\mathcal{E}}$" and "Age$^{\mathcal{E}}$".

```
Algorithm: PreProc(R)
$\hat{R} \leftarrow \varnothing$;
$\mathbb{A}^f \leftarrow \{A^f | A \in \mathcal{A}\}$;
$\mathbb{A}^{\mathcal{E}} \leftarrow \{A^{\mathcal{E}} | A \in \mathcal{A}\}$;
$\hat{\mathbb{R}} \leftarrow \mathbb{A}^f \cup \mathbb{A}^{\mathcal{E}} \cup B$;
for $t \in R$ do
    $t_f \leftarrow \times_{A^f \in \mathbb{A}^f} f_k(\pi_A(t))$;
    $t_{\mathcal{E}} \leftarrow \times_{A^{\mathcal{E}} \in \mathbb{A}^{\mathcal{E}}} (\mathcal{E}_{\mathsf{pk}_u}(\pi_A(t)))$;
    $\hat{R} \leftarrow \hat{R} \cup \{t_f \times t_{\mathcal{E}} \times \mathsf{E}(\pi_B(t))\}$;
```

Algorithm 1: Preprocessing of relations.

SP-COUNT (Figure 6(a)). Value of pairs sent by the map function contains the Paillier encryption of the grouping attribute value and the Paillier encryption of 1. Using the homomorphic property of the Paillier's scheme, each reducer multiplies encryption of 1 to obtain the count of tuples sharing the same value of the grouping attribute.

SP-SUM (Figure 6(b)). Value of pairs sent by the map function contains the Paillier encryption of the grouping attribute value and the Paillier encryption of the aggregation attribute value. Similarly to the SP-COUNT protocol, we use the homomorphic property of the Paillier's scheme allowing each reducer to multiply encrypted aggregates to obtain the encryption of the sum of tuples values sharing the same grouping attribute value.

SP-AVG (Figure 6(c)). The protocol combines the SP-COUNT protocol and the SP-SUM protocol. This allows the MapReduce user to compute the average.

SP-MIN (Figure 6(d)). We stress that before to apply the map function, the data owner must encrypt all values of the aggregate attribute using an OPE scheme with the secret key K shared between the data owner and the MapReduce user.

```
*Map function*
**Input:** $(key, value)$
// *key*: id of a chunk of $\hat{R}$
// *value*: collection of $t \in \hat{R}$
**foreach** $t \in \hat{R}$ **do**
    emit $(\pi_{\mathbb{A}^f}(t), (\pi_{\mathbb{A}^{\mathcal{E}}}(t), \mathcal{E}_{\mathsf{pk}_u}(1)))$
*Reduce function*
**Input:** $(key, values)$
// *key*: $\pi_{\mathbb{A}^f}(t)$ for $t \in \hat{R}$
// *values*: collection of $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \mathcal{E}_{\mathsf{pk}_u}(1))$
count $\leftarrow 1$
**foreach** $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \mathcal{E}_{\mathsf{pk}_u}(1)) \in values$ **do**
    count $\leftarrow$ count $\cdot \mathcal{E}_{\mathsf{pk}_u}(1)$
    emit $(\pi_{\mathbb{A}^{\mathcal{E}}}, \mathsf{count})$
```

(a) SP-COUNT protocol.

```
*Map function*
**Input:** $(key, value)$
// *key*: id of a chunk of $\hat{R}$
// *value*: collection of $t \in \hat{R}$
**foreach** $t \in \hat{R}$ **do**
    emit $(\pi_{\mathbb{A}^f}(t), (\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t)))$
*Reduce function*
**Input:** $(key, value)$
// *key*: $\pi_{\mathbb{A}^f}(t)$ for $t \in \hat{R}$
// *value*: collection of $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t))$
sum $\leftarrow 1$
**foreach** $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t)) \in values$ **do**
    sum $\leftarrow$ sum $\cdot \pi_B(t)$
emit $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \mathsf{sum})$
```

(b) SP-SUM protocol.

```
*Map function*
**Input:** $(key, value)$
// *key*: id of a chunk of $\hat{R}$
// *value*: collection of $t \in \hat{R}$
**foreach** $t \in \hat{R}$ **do**
    emit $(\pi_{\mathbb{A}^f}(t), (\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t), \mathcal{E}_{\mathsf{pk}_u}(1)))$
*Reduce function*
**Input:** $(key, value)$
// *key*: $\pi_{\mathbb{A}^f}(t)$ for $t \in \hat{R}$
// *value*: collection of $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t), \mathcal{E}_{\mathsf{pk}_u}(1))$
cpt $\leftarrow 1$
sum $\leftarrow 1$
**foreach** $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t), \mathcal{E}_{\mathsf{pk}_u}(1)) \in values$ **do**
    cpt $\leftarrow$ cpt $\cdot \mathcal{E}_{\mathsf{pk}_u}(1)$
    sum $\leftarrow$ sum $\cdot \pi_B(t)$
    emit $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \mathsf{cpt}, \mathsf{sum})$
```

(c) SP-AVG protocol.

```
*Map function*
**Input:** $(key, value)$
// *key*: id of a chunk of $\hat{R}$
// *value*: collection of $t \in \hat{R}$
**foreach** $t \in \hat{R}$ **do**
    emit $(\pi_{\mathbb{A}^f}(t), (\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t)))$
*Reduce function*
**Input:** $(key, values)$
// *key*: $\pi_{\mathbb{A}^f}(t)$ for $t \in \hat{R}$
// *values*: collection of $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t))$
$(v_1, v_2) \overset{\$}{\leftarrow} values$
min $\leftarrow \mathcal{D}_{\mathsf{sk}_c}(v_2)$
**foreach** $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \pi_B(t)) \in values$ **do**
    $x \leftarrow \mathcal{D}_{\mathsf{sk}_c}(\pi_B(t))$
    **if** $x <$ min **then**
        min $\leftarrow x$
    emit $(\pi_{\mathbb{A}^{\mathcal{E}}}(t), \mathcal{E}_{\mathsf{pk}_u}(\mathsf{min}))$
```

(d) SP-MIN protocol.

Figure 6: Secure grouping and aggregation with MapReduce for COUNT, SUM, AVG, and MIN operations. The highlighting emphasizes differences w.r.t. the standard non-secured approach cf. Figure 5.

Value of pairs sent by the map function contains the encryption of the pre-computed OPE ciphertexts using an IND-CPA public key encryption scheme with the public key pk_c of the public cloud. Since the OPE encryption is deterministic, the additional public key encryption avoids an eavesdropper between the data owner and the public cloud to have any information on repetitions of values sent by the data owner.

After received the key-value pairs, the public cloud uses its secret key sk_c to obtain OPE ciphers. Using the property of the OPE scheme, each reducer of the public cloud computes the minimum to obtain the minimum value associated to the considered value of the grouping attribute. Finally, the public cloud uses the public key pk_u of the user to encrypt each OPE ciphertext and sends the result to the user.

Remark: As we can see in the SP-COUNT protocol (Figure 6(a)), a public cloud knowing that it performs the count operation can deduce the value of the count even if it can not decrypt the encryption of 1. In fact, the public cloud can count tuples that each reducer receives. Hence, it deduce the count result for the corresponding key. We stress that the plain value of the key stay unknown from the public cloud since it does not have the secret key sk_u of the user to decrypt it. In the following, we present the SP$^{\mathsf{comb}}$-COUNT and the SP$^{\mathsf{comb}}$-AVG protocols in Figure 7 using combiners (Leskovec et al., 2014) to avoid this leakage of information.

3.2 Refinement: Combiners

Combiners allow to push some of what the reducers do to the map function. In the case of the COUNT operation, the map function counts tuples of the chunk that share the same value for the grouping attribute. Hence, each reducer receives key-value pairs, where key is the grouping attribute value, and value is the count of tuples sharing this key in the chunk.

We use homomorphic property of the Paillier's scheme to count in the map function the number of

```
Map function:
Input: $(key, value)$
// key: id of a chunk of $\hat{R}$
// value: collection of $(t_1,t_2,t_3) \in \hat{R}$
$L \leftarrow [\,]$;                 // Let $L$ be a dictionary
foreach $(t_1,t_2,t_3) \in \hat{R}$ do
    if $(t_1,t_2) \in L$ then $L[(t_1,t_2)] \leftarrow L[(t_1,t_2)] \cdot \mathcal{E}_{\mathsf{pk}_u}(1)$;
    else $L[(t_1,t_2)] \leftarrow \mathcal{E}_{\mathsf{pk}_u}(1)$;
foreach $(t_1,t_2) \in L$ do
    emit $(t_1,(t_2,L[(t_1,t_2)]))$.
Reduce function:
Input: $(key, values)$
// key: $\pi_A(t)$ for $t \in \hat{R}$
// values: collection of $(\mathcal{E}_{\mathsf{pk}_u}(a), \mathcal{E}_{\mathsf{pk}_u}(b))$
count $\leftarrow 1$;
foreach $(\mathcal{E}_{pk_u}(a), \mathcal{E}_{pk_u}(b)) \in values$ do
    count $\leftarrow$ count $\cdot \mathcal{E}_{\mathsf{pk}_u}(b)$;
    emit $(\mathcal{E}_{\mathsf{pk}_u}(a), \mathsf{count})$.
```

Figure 7: $\mathsf{SP}^{\mathsf{comb}}$-COUNT protocol.

```
Map function:
Input: $(key, value)$
// key: id of a chunk of $\hat{R}$
// value: collection of $(t_1,t_2,t_3) \in \hat{R}$
$L \leftarrow [\,]$;                 // Let $L$ be a dictionary
$M \leftarrow [\,]$;                 // Let $M$ be a dictionary
foreach $(t_1,t_2,t_3) \in \hat{R}$ do
    if $(t_1,t_2) \in L$ then $L[(t_1,t_2)] \leftarrow L[(t_1,t_2)] \cdot \mathcal{E}_{\mathsf{pk}_u}(1)$;
    else $L[(t_1,t_2)] \leftarrow \mathcal{E}_{\mathsf{pk}_u}(1)$;
    if $(t_1,t_2) \in M$ then $M[(t_1,t_2)] \leftarrow M[(t_1,t_2)] \cdot t_3$;
    else $M[(t_1,t_2)] \leftarrow t_3$;
foreach $(t_1,t_2) \in L$ do
    emit $(t_1,(t_2,L[(t_1,t_2)],M[(t_1,t_2)]))$.
Reduce function:
Input: $(key, values)$
// key: $\pi_A(t)$ for $t \in \hat{R}$
// values: collection of $(\mathcal{E}_{\mathsf{pk}_u}(a), \mathcal{E}_{\mathsf{pk}_u}(b), \mathcal{E}_{\mathsf{pk}_u}(c))$
cpt $\leftarrow 1$;
sum $\leftarrow 1$;
foreach $(\mathcal{E}_{pk_u}(a), \mathcal{E}_{pk_u}(b), \mathcal{E}_{pk_u}(c)) \in values$ do
    cpt $\leftarrow$ cpt $\cdot \mathcal{E}_{\mathsf{pk}_u}(b)$;
    sum $\leftarrow$ sum $\cdot \mathcal{E}_{\mathsf{pk}_u}(c)$;
    emit $(\mathcal{E}_{\mathsf{pk}_u}(a), \mathsf{cpt}, \mathsf{sum})$.
```

Figure 8: $\mathsf{SP}^{\mathsf{comb}}$-AVG protocol.

tuples in the chunk that share the same grouping attribute value. Then, each reducer multiplies all encrypted counts for the considered grouping attribute value to obtain the final encrypted count sent to the user. We present this refinement called $\mathsf{SP}^{\mathsf{comb}}$-COUNT protocol in Figure 7.

Similarly, we can use combiners for the AVG operation. Even if the sum is encrypted, combiners hide the count used for each grouping attribute value i.e., for each computed average. We present this refinement called $\mathsf{SP}^{\mathsf{comb}}$-AVG protocol in Figure 8.

We stress that we can also use combiners with SUM, and MIN/MAX operations but they do not add privacy as in previous operations.

3.3 Security Proofs

The security proofs of the SP-SUM protocol (Theorem 1) and of the SP-MIN protocol (Theorem 2) are presented in (Ciucanu et al., 2018). We emphasize that the security for SP-(COUNT-AVG) protocols are similar to the SP-SUM protocol so we do not present them. Moreover, the security proofs for SP-MIN and SP-MAX protocols are identical so we do not present it too.

Theorem 1. *The SP-SUM protocol securely computes the grouping and aggregation for the SUM operation in the ROM in the presence of semi-honest adversary even if cloud nodes collude.*

Theorem 2. *The SP-MIN protocol securely computes the grouping and aggregation for the MIN operation in the ROM in the presence of semi-honest adversaries even if cloud nodes collude.*

4 CONCLUSION

We have presented efficient algorithms for grouping and aggregation operations with MapReduce that enjoy privacy guarantees such as none of the nodes of the public cloud computing can learn the input or the output relation. To achieve our goal, we relied on Paillier's cryptosystem and on Order-Preserving encryption. We developed an efficient approach (SP) on the computation cost side as the communication cost side. We have compared this approach to the standard algorithm with respect to three fundamental criteria: computation cost, communication cost, and privacy guarantees.

Looking forward to future work, we plan to study the practical performance of our algorithms in an open-source system that implements the MapReduce paradigm as Hadoop[1]. Additionally, we aim to investigate the grouping and aggregation computation with privacy guarantees in different big data systems (such as Spark or Flink) whose users also tend to outsource data and computations similarly to MapReduce.

ACKNOWLEDGEMENTS

This research was conducted with the support of the FEDER program of 2014-2020, the region council

[1]Apache Hadoop: https://hadoop.apache.org/

of Auvergne-Rhône-Alpes, the support of the "Digital Trust" Chair from the University of Auvergne Foundation, the Indo-French Centre for the Promotion of Advanced Research (IFCPAR) and the Center Franco-Indien Pour La Promotion De La Recherche Avancée (CEFIPRA) through the project DST/CNRS 2015-03 under DST-INRIA-CNRS Targeted Programme.

REFERENCES

Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. (2004). Order-Preserving Encryption for Numeric Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–574.

Alghamdi, W. Y., Wu, H., and Kanhere, S. S. (2017). Reliable and Secure End-to-End Data Aggregation Using Secret Sharing in WSNs. In *2017 IEEE Wireless Communications and Networking Conference, WCNC*, pages 1–6.

Blass, E., Pietro, R. D., Molva, R., and Önen, M. (2012). PRISM - Privacy-Preserving Search in MapReduce. In *Privacy Enhancing Technologies - 12th International Symposium, PETS*, pages 180–200.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. (2017). Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS*, pages 1175–1191.

Bultel, X., Ciucanu, R., Giraud, M., and Lafourcade, P. (2017). Secure Matrix Multiplication with MapReduce. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pages 11:1–11:10.

Ciucanu, R., Giraud, M., Lafourcade, P., and Ye, L. (2018). Secure grouping and aggregation with mapreduce. Cryptology ePrint Archive, Report 2018/501. https://eprint.iacr.org/2018/501.

Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *6th Symposium on Operating System Design and Implementation OSDI*, pages 137–150.

Derbeko, P., Dolev, S., Gudes, E., and Sharma, S. (2016). Security and privacy aspects in MapReduce on clouds: A survey. *Computer Science Review*, 20:1–28.

Dolev, S., Gilboa, N., and Li, X. (2015). Accumulating Automata and Cascaded Equations Automata for Communicationless Information Theoretically Secure Multi-Party Computation: Extended Abstract. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing, SCC@ASIACCS '15*, pages 21–29.

Dolev, S., Li, Y., and Sharma, S. (2016). Private and Secure Secret Shared MapReduce. In *Data and Applications Security and Privacy XXX - 30th Annual IFIP WG 11.3 Conference, DBSec*, pages 151–160.

Gentry, C. (2009). Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178. ACM.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of Massive Datasets*. Cambridge University Press.

Macedo, R., Paulo, J., Pontes, R., Portela, B., Oliveira, T., Matos, M., and Oliveira, R. (2017). A practical framework for privacy-preserving nosql databases. In *36th IEEE Symposium on Reliable Distributed Systems, SRDS 2017, Hong Kong, Hong Kong, September 26-29, 2017*, pages 11–20.

Mayberry, T., Blass, E., and Chan, A. H. (2013). PIRMAP: Efficient Private Information Retrieval for MapReduce. In *Financial Cryptography and Data Security - 17th International Conference, FC*, pages 371–385.

Naccache, D. and Stern, J. (1998). A New Public Key Cryptosystem Based on Higher Residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, pages 59–66, New York, NY, USA. ACM.

Okamoto, T. and Uchiyama, S. (1998). *A New Public-key Cryptosystem as Secure as Factoring*, pages 308–318. Springer Berlin Heidelberg.

Paillier, P. (1999). Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, pages 223–238.

Popa, R. A., Redfield, C. M. S., Zeldovich, N., and Balakrishnan, H. (2011). Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 85–100.

Shamir, A. (1979). How to Share a Secret. *Commun. ACM*, 22(11):612–613.

Vo-Huu, T. D., Blass, E., and Noubir, G. (2015). EPiC: Efficient Privacy-Preserving Counting for MapReduce. In *Networked Systems - Third International Conference, NETYS*, pages 426–443.