Secure Grouping and Aggregation with MapReduce
Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, Lihua Ye

To cite this version:
Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, Lihua Ye. Secure Grouping and Aggregation with MapReduce. 15th International Conference on Information Security and Cryptography, SECRIPT’18, Jul 2018, Porto, Portugal. pp.514–521. hal-01874859

HAL Id: hal-01874859
https://hal.science/hal-01874859
Submitted on 14 Sep 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Secure Grouping and Aggregation with MapReduce

Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, Lihua Ye
LIMOS, Université Clermont Aubervie, Aubière, France
firstname.lastname@uca.fr

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 \textit{SUM}, \textit{COUNT}, \textit{AVG}, \textit{MIN}, and \textit{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: \textit{SUM}, \textit{COUNT}, \textit{AVG}, \textit{MIN}, and \textit{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 $\theta$ 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 \textit{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 corresponds to the salary of Alice which is equal to 1900. 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 $\gamma_{\text{Department}, \text{SUM} (\text{Salary})}(R)$, we obtain the relation presented in Figure 2. Aggregation operations \textit{COUNT}, \textit{AVG}, \textit{MIN}, or \textit{MAX} work similarly.

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 \textit{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 \textit{reduce} function applies the aggregate function on the associated values of the considered key.

<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Computer Science</td>
<td>1900</td>
</tr>
<tr>
<td>Mallory</td>
<td>Mathematics</td>
<td>1750</td>
</tr>
<tr>
<td>Bob</td>
<td>Computer Science</td>
<td>1800</td>
</tr>
<tr>
<td>Eve</td>
<td>Physics</td>
<td>2000</td>
</tr>
<tr>
<td>Oscar</td>
<td>Mathematics</td>
<td>1600</td>
</tr>
</tbody>
</table>

Figure 1: Relation $R$.

<table>
<thead>
<tr>
<th>Department</th>
<th>SUM (salary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science</td>
<td>3700</td>
</tr>
<tr>
<td>Physics</td>
<td>2000</td>
</tr>
<tr>
<td>Mathematics</td>
<td>3350</td>
</tr>
</tbody>
</table>

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

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 paper (Dean and Ghemawat, 2004), different protocols have been proposed to perform operations in a privacy-preserving manner (Bonawitz et al., 2017) provides a technique to compute secure aggregation, while relying on Shamir’s secret sharing (Shamir, 1979) to compute 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 $\gamma_{\mathcal{A},B}(R)$ 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 $\theta$ is an aggregation function ($\text{SUM}$, $\text{COUNT}$, $\text{AVG}$, $\text{MIN}$, $\text{MAX}$).

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 $\text{COUNT}$ in Figure 5(a), for $\text{SUM}$ in Figure 5(b), for $\text{AVG}$ in Figure 5(c), and for $\text{MIN}$ in Figure 5(d). The algorithm for $\text{MAX}$ is very similar to the one for $\text{MIN}$ 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.

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

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

\[
\Pr[\mathcal{B}(\cdot) = 1 : k \leftarrow \{0,1\}^n] - \Pr[\mathcal{B}(\cdot) = 1 : g \leftarrow \text{Func}[\eta_0,\eta_1]] \leq \varepsilon(\eta),
\]

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

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

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

\[ \text{E}^\text{ope}_K(m) : \text{returns a new key } K' \text{ and a ciphertext } c. \]

\[ \text{D}^\text{ope}_K(c) : \text{returns the plaintext } m. \]

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 \( \eta \) be a security parameter. A public key encryption (PKE) scheme is defined by three algorithms \( (\mathcal{G}, \mathcal{E}, \mathcal{D}) \): \n
\[ \mathcal{G}(\eta) : \text{returns a public/private key pair } (pk, sk). \]

\[ \mathcal{E}_{sk}(m) : \text{returns the ciphertext } c. \]

\[ \mathcal{D}_{pk}(c) : \text{returns the plaintext } m. \]

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 \( \mathbb{Z}_n^\ast \) the set of invertible elements of \( \mathbb{Z}_n \). The public key \( pk \) of Paillier’s encryption scheme is \((n, g), g \in \mathbb{Z}_n^\ast \) and \( n = pq \) is the product of two prime numbers such that \( \gcd(p, q) = 1 \). The corresponding private key \( sk \) is \((\lambda, \mu)\), where \( \lambda \) is the least common multiple of \( p-1 \) and \( q-1 \) and \( \mu = (L(g^2 \mod n^2))^{-1} \mod n \), where \( L(x) = x^{\frac{\lambda-1}{n}} \).

Encryption Algorithm. Let \( m \) be a message such that \( m \in \mathbb{Z}_n \). Let \( g \) be an element of \( \mathbb{Z}_n^\ast \) and \( t \) be a random element of \( \mathbb{Z}_n^\ast \). We denote by \( E_{pk} \) the encryption function that produces the ciphertext \( c \) from the given plaintext \( m \) with the public key \( pk = (n, g) \) as follows: \[ c = g^m \times t^r \mod n^2 \]

Decryption Algorithm. Let \( c \) be the ciphertext such that \( c \in \mathbb{Z}_{n^2} \). We denote by \( D_{pk} \) the decryption function of the plaintext \( c \) with the secret key \( sk = (\lambda, \mu) \) defined as follows: \[ m = L(c^\lambda \mod n^2) \times \mu \mod n \]

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 \( pk = (n, g) \), denoted \( c_1 = E_{pk}(m_1) = g^{m_1} \times r_1^t \mod n^2 \) and \( c_2 = E_{pk}(m_2) = g^{m_2} \times r_2^t \mod n^2 \), is the encryption of the sum of \( m_1 \) and \( m_2 \).

\[ E_{pk}(m_1) \times E_{pk}(m_2) = c_1 \times c_2 \mod n^2 = (g^{m_1} \times r_1^t) \times (g^{m_2} \times r_2^t) \mod n^2 = (g^{m_1+m_2} \times (r_1 \times r_2)^t) \mod n^2 = E_{pk}(m_1+m_2 \mod n) \]

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 grouping 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 \( A^e \) are just notations making explicit the correspondences between initial and outsourced data and that \( \mathbb{E} \) 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\(^e \)” and “Age\(^e \)”.

Algorithm: PreProc(\( R \))

\[ \hat{R} \leftarrow \emptyset ; \]
\[ A^f \leftarrow \{ A^f | A \in A \} ; \]
\[ A^e \leftarrow \{ A^e | A \in A \} ; \]
\[ \mathbb{E} \leftarrow A^f \cup A^e \cup B ; \]
\[ \text{for } i \in R \text{ do} \]
\[ t_f \leftarrow \times_{A^f \in A^f} f_r(\pi_A(t)) ; \]
\[ t_e \leftarrow \times_{A^e \in A^e} E_{pk_u}(\pi_A(t)) ; \]
\[ \hat{R} \leftarrow \hat{R} \cup \{ t_f \times t_e \times E(\pi_B(t)) \} ; \]

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.

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 \( \text{pk}_u \) 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 \( \text{sk}_u \) 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 \( \text{pk}_u \) 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 deduces 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 \( \text{sk}_u \) of the user to decrypt it. In the following, we present the Spcomb-COUNT and the Spcomb-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.

![Figure 6](image_url)

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.
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 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 SPcomb.COUNT protocol in Figure 7.

Similarly, we can use combiners for the AVG operation. Even if the sum is encrypted, combinators hide the count used for each grouping attribute value i.e., for each computed average. We present this refinement called SPcomb.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.

---

\(^1\)Apache Hadoop: [https://hadoop.apache.org/](https://hadoop.apache.org/)
ACKNOWLEDGEMENTS

This research was conducted with the support of the FEDER program of 2014-2020, the region council 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 Recherche Avancée (CEFIPRA) through the project DST/INRIA-CNRS Targeted Programme.

REFERENCES


