Secure Matrix Multiplication with MapReduce

Xavier Bultel, Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade

Secure Matrix Multiplication with MapReduce

Xavier Bultel, Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade
Université Clermont Auvergne & CNRS LIMOS, France
firstname.lastname@uca.fr

Abstract

The MapReduce programming paradigm allows to process big data sets in parallel on a large cluster of commodity machines. The MapReduce users often outsource their data and computations to a public cloud provider. We focus on the fundamental problem of matrix multiplication, and address the inherent security and privacy concerns that occur when outsourcing to a public cloud. Our goal is to enhance the two state-of-the-art algorithms for MapReduce matrix multiplication with privacy guarantees such as: none of the nodes storing an input matrix can learn the other input matrix or the output matrix, and moreover, none of the nodes computing an intermediate result can learn the input or the output matrices. To achieve our goal, we rely on the well-known Paillier's cryptosystem and we use its partially homomorphic property to develop efficient algorithms that satisfy our problem statement. We develop two different approaches called *Secure-Private* (SP) and *Collision-Resistant-Secure-Private* (CRSP), and compare their trade-offs with respect to three fundamental criteria: computation cost, communication cost, and privacy guarantees. Finally, we give security proofs of our protocols.

1 Introduction

MapReduce [6] is a programming paradigm for processing big data sets. The MapReduce programs are automatically parallelized and executed on a large cluster of commodity machines. The users need to specify two functions (*map* and *reduce*), whereas the system takes care of several aspects such as partitioning the data, scheduling the program's execution across the machines, handling machine failures, and managing the communication between different machines.

In practice, the MapReduce users often outsource their data and computations i.e., they rent storage and computing resources from a public cloud provider (e.g., Google Cloud Platform, Amazon Web Services, Microsoft Azure). On the positive side, using a public cloud makes the big data processing accessible to users who cannot afford to build their own clusters. However, outsourcing the data and computations to a public cloud involves inherent security and privacy concerns. Indeed, the MapReduce users are no longer in control of their data, which may be communicated over an untrusted network and processed on some untrusted machine, where malicious public cloud users may breach their privacy.

We address the fundamental problem of *MapReduce matrix multiplication* from a privacy-preserving perspective i.e., we develop algorithms for which the public cloud cannot learn neither the input nor the output data. The matrix multiplication is also the original purpose for which the Google implementation of MapReduce was created. Such multiplications are needed by Google in the computation of the PageRank algorithm. The standard algorithms for MapReduce matrix multiplication use either two or one communications rounds, and moreover, their communication and computation cost analysis have been throughly analysed in Chapter 2 of [12].

Problem statement. Two compatible matrices M and N are stored in the distributed file system of some public cloud provider. A user (who does not know the matrices M and N) wants their product $P = M \times N$. We assume that the matrix M is initially spread over a set $\mathcal{M}$ of nodes, each of them storing a chunk of M i.e., a set of elements of M. Similarly, the matrix N is initially spread over a set $\mathcal{N}$ of nodes. In the case of one round (Figure 1), the final result P is spread over a set $\mathcal{R}$ of nodes before it is sent to the user's nodes $\mathcal{P}$; in the case of two rounds (Figure 2), intermediate results are spread over a set $\mathcal{R}_1$ of nodes and the final result P is spread over a set $\mathcal{R}_2$ of nodes before it is sent to the user's nodes $\mathcal{P}$. We expect the following properties:

1. the user cannot learn any information about input matrices M and N,
2. none of the nodes in $\mathcal{M}$ can learn any information about matrices N and P,
3. none of the nodes in $\mathcal{N}$ can learn any information about matrices M and P,

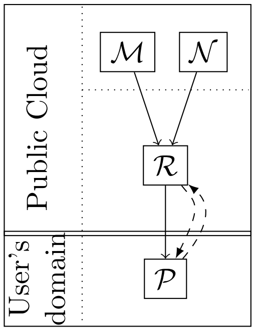

Figure 1: One Round.

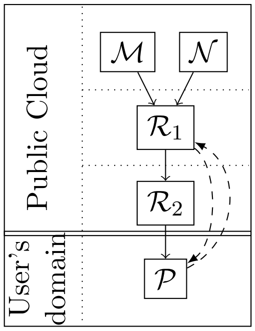

Figure 2: Two Rounds.

Algorithm	*# rounds*	*Computation cost (big-O)*	*Comm. cost (big-O)*	*Privacy*
Standard in [12]	Two	$2n^2 + (C_\times + C_+)n^3$	$n^3 + 3n^2$	None
	One		$2n^3 + n^2$	
Secure-Private (SP)	Two	$(C_+ + 2C_\mathcal{E})n^2 + (2C_\times + 2C_{\exp} + C_\mathcal{D})n^3$	$3n^3 + 4n^2$	All†
	One	$(1 + C_\mathcal{E})n^2 + (C_\times + C_{\exp})n^3$	$2n^3 + n^2$	Only (1)–(3)
CRSP	Two	$2C_\mathcal{E}n^2 + (4C_\mathcal{E} + 7C_\times + 2C_\mathcal{D} + 2C_{\exp})n^3$	$4n^3 + 3n^2$	All
	One		$5n^3 + n^2$	

† Assuming nodes do not collude.

Figure 3: Summary of results. Let n be $\max(a, b, c)$, and $C_\times$ (resp. C_+, $C_{\exp}$, $C_\mathcal{E}$, $C_\mathcal{D}$) is the cost of multiplication (resp. addition, exponentiation, encryption, decryption).

4. none of the nodes in $\mathcal{R}$ (for one round), or in $\mathcal{R}_1$ and $\mathcal{R}_2$ (for two rounds) can learn any information about matrices M, N, and P.

The second and third condition state that none of the nodes storing an input matrix can learn the other input matrix or the output matrix, whereas the fourth condition states that none of the public cloud's nodes storing intermediate or final result can learn the input or the output matrices.

Notice that a straightforward solution would require the use of a fully homomorphic encryption scheme e.g., Gentry [11]. Indeed, a fully homomorphic encryption scheme would allow to execute directly in the encrypted domain all multiplications and additions needed for computing a matrix multiplication. 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 [11]).

Summary of contributions. We propose algorithms that extend the two standard algorithms for MapReduce matrix multiplication (as found in Chapter 2 from [12]) while ensuring data privacy, and remaining efficient from both computational and communication points of view.

Our technique is based on the well-known Paillier's cryptosystem [17], which is partially homomorphic i.e., it is additively homomorphic. The integration of this cryptosystem into the standard MapReduce algorithms for matrix multiplication is not trivial. Indeed, Paillier's cryptosystem does not allow to execute directly in the encrypted domain the multiplications that are needed for matrix multiplication.

Assuming that the public cloud's nodes do not collude, we design the *Secure-Private (SP)* approach which satisfies all afeorementionned conditions. Indeed, we show that if nodes $\mathcal{N}$ collude with nodes $\mathcal{R}_2$, then $\mathcal{R}_2$ can retrieve all elements of matrix N. The second approach designs a sophisticated algorithm that relies on additional communications to overcome these risks of collusions; this idea led to our *Collision-Resistant-Secure-Private (CRSP)* approach, which satisfies all conditions enumerated in the problem statement.

We summarize in Figure 3 the trade-offs between computation cost, communication cost, and privacy guarantees for our two approaches and the two standard MapReduce algorithms for matrix multiplication. In our communication cost analysis, we measure the total size of the data that is emitted from a map or reduce node, and the additional data that needs to be communicated to realize the interactive multiplication in the encrypted domain. The CRSP approach satisfies all privacy constraints and resists to collusions, but requires a communication overhead.

Related work. Chapter 2 of [12] presents an introduction to the MapReduce paradigm. In particular, it includes the two MapReduce algorithms for matrix multiplication that we enhance with privacy preservation.

The security and privacy concerns of MapReduce have been summarized in a recent survey [7]. To the best of our knowledge, no existing work has addressed the problem of matrix multiplication, on which we focus in this paper. More precisely, the state-of-the-art techniques for execution of MapReduce computations while preserving privacy focus on problems such as word search [3, 8], information retrieval [14, 8], count queries [19, 8], equijoins [8], and range queries [8]. The general goal of these works is to execute MapReduce computations such that the public cloud cannot learn the data. This is precisely our goal too, but for a fundamentally different task i.e., matrix multiplication.

Distributed matrix multiplication has been thoroughly investigated in the secure multi-party computation model (MPC) [9, 1, 20, 10], whose goal is to allow different nodes to jointly compute a function over their private inputs without revealing them. The aforementioned works on secure distributed matrix multiplication have different assumptions compared to our MapReduce framework: (i) they assume that nodes contain entire vectors, whereas the division of the initial matrices in chunks as done in MapReduce does not have such assumptions, and (ii) in MapReduce, the functions specified by the user [6] are limited to *map* (process a key/value pair to generate a set of intermediate key/value pairs) and *reduce* (merge all intermediate values associated with the same intermediate key), and the matrix multiplication is done in one or two communication rounds [12]; on the other hand, the works in the MPC model assume arbitrary numbers of communication rounds, relying on more complex functions than map and reduce.

Moreover, generic MPC protocols [13, 4] allow several nodes to securely evaluate any function. Such protocols could be used to secure MapReduce. However, due to their generic nature, they are inefficient and require a lot of interactions between parties. Our goal is to design an optimized protocol to secure MapReduce.

Paper organization. We introduce the needed cryptographic tools in Section 2 and the standard algorithms for matrix multiplication with MapReduce in Section 3. We present our SP algorithm for secure matrix multiplication with MapReduce and its analysis in Section 4, and the CRSP algorithm in Section 5. We prove the security of our algorithms in Section 6. We outline conclusion and future work in Section 7.

2 Cryptographic Tools

We start by recalling the definition of negligible function used in security proofs and definition, and security requirements of public key cryptosystems.

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

Definition 2 (Public Key Encryption (PKE))**.** *Let η be a security parameter. A PKE scheme is defined by three algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$:*

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

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

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

A PKE scheme $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ is *indistinguishable under chosen-plaintext attack* (IND-CPA) [2] if for any probalistic polynomial time adversary $\mathcal{A}$, the difference between $\frac{1}{2}$ and the probability that $\mathcal{A}$ wins the IND-CPA experiment in Figure 4 is negligible, where the oracle $\mathcal{E}_{\mathsf{pk}}(\mathsf{LR}_b(\cdot, \cdot))$ takes (m_0, m_1) as input and returns $\mathcal{E}_{\mathsf{pk}}(m_b)$. The standard definition of CPA experiment allows the adversary to call this oracle only one time. However, in [2] authors prove that the two definitions of CPA security are equivalent using an hybrid argument.

In the following, we require an additive homomorphic encryption scheme to secure the computation of matrix multiplication with MapReduce. There exist several schemes that have this property [16, 17, 5, 15]. We choose Paillier's public key encryption scheme [17] 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.

```
$\mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(\eta)$:
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{G}(\eta)$;
$b_* \leftarrow \mathcal{A}^{\mathcal{E}_{\mathsf{pk}}(\mathsf{LR}_b(\cdot,\cdot))}(\mathsf{pk})$
return $(b = b_*)$
```

Figure 4: IND-CPA experiment [2].

A		B
		Knows $c_1 = \mathcal{E}_{\mathsf{pk}_A}(m_1)$ and $c_2 = \mathcal{E}_{\mathsf{pk}_A}(m_2)$
		Picks two randoms $\delta_1, \delta_2 \in \mathbb{Z}_n$
$\mathcal{D}_{\mathsf{sk}_A}(\alpha_1) = m_1 + \delta_1$ and $\mathcal{D}_{\mathsf{sk}_A}(\alpha_2) = m_2 + \delta_2$	$\xleftarrow{\alpha_1,\alpha_2}$	$\alpha_1 = c_1 \times \mathcal{E}_{\mathsf{pk}_A}(\delta_1)$ and $\alpha_2 = c_2 \times \mathcal{E}_{\mathsf{pk}_A}(\delta_2)$
$\beta = \mathcal{E}_{\mathsf{pk}_A}((m_1 + \delta_1) \times (m_2 + \delta_2))$	$\xrightarrow{\beta}$	$\mathcal{E}_{\mathsf{pk}_A}(m_1 \times m_2) = \frac{\beta}{\mathcal{E}_{\mathsf{pk}_A}(\delta_1 \times \delta_2) \times c_1^{\delta_2} \times c_2^{\delta_1}}$

Figure 5: Paillier interactive mutiplicative homomorphic protocol, of computation cost $O(4C_{\mathcal{E}}+6C_{\times}+2C_{\mathcal{D}}+2C_{\exp})$ and communication cost $O(3)$.

2.1 Paillier's Scheme

Paillier's scheme is an IND-CPA scheme [18], 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 inversible 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: $m = L\left(c^\lambda \mod n^2\right) \times \mu \mod n$.

2.2 Homomorphic Properties

Paillier's cryptosystem is a partial homomorphic encryption scheme. We present these properties.

Homomorphic Addition of Plaintexts. 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.

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

We also remark that: $\frac{\mathcal{E}_{\mathsf{pk}}(m_1)}{\mathcal{E}_{\mathsf{pk}}(m_2)} = \mathcal{E}_{\mathsf{pk}}(m_1 - m_2)$.

Specific Homomorphic Multiplication of Plaintexts. Let m_1 and m_2 be two plaintexts in $\mathbb{Z}_n$ and $c_1 \in \mathbb{Z}_{n^2}^*$ be the ciphertext of m_1 with the public key pk $(c_1 = \mathcal{E}_{\mathsf{pk}}(m_1))$. With Paillier's scheme, c_1 raised to the power of

m_2 is the encryption of the product of the two plaintexts m_1 and m_2.

$$\begin{aligned}\mathcal{E}_{\mathsf{pk}}(m_1)^{m_2} &= c_1^{m_2} \mod n^2 \\ &= (g^{m_1} \times r_1^n)^{m_2} \mod n^2 \\ &= (g^{m_1 \cdot m_2} \times (r_1^{m_2})^n) \mod n^2 \\ &= \mathcal{E}_{\mathsf{pk}}(m_1 \times m_2 \mod n) \ .\end{aligned}$$

Interactive Homomorphic Multiplication of Ciphertexts. Cramer et al. [4] show that an interactive protocol makes possible to perform multiplication over ciphertexts using additive homomorphic encryption schemes as Paillier's encryption scheme. More precisely, Bob knows two ciphertexts $c_1, c_2 \in \mathbb{Z}_{n^2}^*$ of the plaintexts $m_1, m_2 \in \mathbb{Z}_n$ with the public key of Alice, he wants to obtain the cipher of the product of m_1 and m_2 without revealing to Alice m_1 and m_2. For this Bob has to interact with Alice as described in Figure 5. Bob first picks two randoms δ_1 and δ_2 and sends to Alice $\alpha_1 = c_1 \times \mathcal{E}_{\mathsf{pk}_A}(\delta_1)$ and $\alpha_2 = c_2 \times \mathcal{E}_{\mathsf{pk}_A}(\delta_2)$. By decrypting respectively α_1 and α_2, Alice recovers respectively $m_1 + \delta_1$ and $m_2 + \delta_2$. She sends to Bob $\beta = \mathcal{E}_{\mathsf{pk}_A}((m_1 + \delta_1) \times (m_2 + \delta_2))$. Then, Bob can deduce the value of $\mathcal{E}(m_1 \cdot m_2)$ by computing: $\frac{\beta}{\mathcal{E}_{\mathsf{pk}_A}(\delta_1 \times \delta_2) \times c_1^{\delta_1} \times c_2^{\delta_2}}$, since $\mathcal{E}_{\mathsf{pk}_A}((m_1 + \delta_1) \times (m_2 + \delta_2)) = \mathcal{E}_{\mathsf{pk}_A}(m_1 \times m_2) \times \mathcal{E}_{\mathsf{pk}_A}(m_1 \times \delta_2) \times \mathcal{E}_{\mathsf{pk}_A}(m_2 \times \delta_1) \times \mathcal{E}_{\mathsf{pk}_A}(\delta_1 \times \delta_2)$.

3 Matrix Multiplication

Let M and N be two compatible matrices, respectively of size $a \times b$ and $b \times c$. We denote by m_{ij} the element of the matrix M which is in the i-th row and the j-th column with $1 \leq i \leq a$ and $1 \leq j \leq b$. In the same way, we denote by n_{jk} the element of the matrix N which is in the j-th row and k-th column with $1 \leq j \leq b$ and $1 \leq k \leq c$. Moreover, we denote by P the product $M \times N$.

3.1 Two MapReduce Rounds

The matrix multiplication with two MapReduce rounds is composed of four functions: first Map function (2R-M1), first Reduce function (2R-R1), second Map function (2R-M2) and second Reduce function (2R-R2).

The First Map Function (Figure 6(a)). 2R-M1 consists of rewriting each element of matrices M and N in the form of key-value pairs such that elements needed to compute each element of the product $M \times N$ share the same key. Hence, nodes $\mathcal{M}$ create pairs of the form $(j, (M, i, m_{ij}))$, where $1 \leq i \leq a$ and $1 \leq j \leq b$ and send them to the set of nodes $\mathcal{R}_1$. In the same way, nodes $\mathcal{N}$ create pairs of the form $(j, (N, k, n_{jk}))$ where $1 \leq j \leq b$ and $1 \leq k \leq c$ and send them to $\mathcal{R}_1$. We stress that M and N in the values are the names of matrices, that can be encoded with a single bit, and not the matrices themselves.

The First Reduce Function (Figure 6(b)). To compute elements of P, the first step is to compute each product $m_{ij} \times n_{jk}$ for $1 \leq j \leq b$. Hence 2R-M1 executed on $\mathcal{R}_1$ creates key-values pairs and sends them to $\mathcal{R}_2$ where for a key (i, k) with $1 \leq i \leq a$ and $1 \leq k \leq c$, values are $m_{ij} \times n_{jk}$ for $1 \leq j \leq b$.

The Second Map Function. In this second round, 2R-M2 is the identity function, hence we further omit it from the analysis of the computation and communication costs.

The Second Reduce Function (Figure 6(c)). It is executed by nodes of type $\mathcal{R}_2$ and aggregates all values with the same key to obtain key-value pairs $((i, k), \sum_{j=1}^{b} m_{ij} \times n_{jk})$. Then $\mathcal{R}_2$ sends results to $\mathcal{P}$.

3.2 One MapReduce Round

The matrix multiplication with one MapReduce round is composed of two functions: the Map function (1R-M) and the Reduce function (1R-R).

The Map Function (Figure 7(a)). It is executed on $\mathcal{M}$ and $\mathcal{N}$, and creates the sets of matrix elements that are needed to compute each element of the product $M \times N$. Since an element of M or N is used for many elements of the final result, the output of the Map function sent to $\mathcal{R}$ gives key-value pairs where keys are (i, k) where i is the row of M and k is the column of N, and values are of the form (N, j, m_{ij}) and (N, j, n_{jk}).

The Reduce Function (Figure 7(b)). With the previous Map function, we obtain for a key (i, k) a set of values of the form (M, j, m_{ij}) and (N, j, n_{jk}) with $1 \leq j \leq b$. Then, 1R-R executed by $\mathcal{R}$ computes the sum of $m_{ij} \times n_{jk}$

```
Input: $(key, value)$
// $key$: id of a chunk of $M$ or $N$
// $value$: collection of $(i, j, m_{ij})$
//                    or $(j, k, n_{jk})$
foreach $(i, j, m_{ij}) \in value$ do
    emit$_{\mathcal{M}\to\mathcal{R}_1}(j, (M, i, m_{ij}))$
foreach $(j, k, n_{jk}) \in value$ do
    emit$_{\mathcal{N}\to\mathcal{R}_1}(j, (N, k, n_{jk}))$
```

(a) 1st Map function (2R-M1).

Computation cost: $2n^2$.
Communication cost: $2n^2$.

```
Input: $(key, values)$
//$key$: $1 \leq j \leq b$
//$values$: collection of $(M, i, m_{ij})$
//                    or $(N, k, n_{jk})$
foreach $(M, i, m_{ij}) \in values$ do
    foreach $(N, k, n_{jk}) \in values$ do
        emit$_{\mathcal{R}_1\to\mathcal{R}_2}((i, k), (m_{ij} \times n_{jk}))$
```

(b) 1st Reduce function (2R-R1).

Computation cost: $C_\times n^3$.
Communication cost: n^3.

```
Input: $(key, values)$
//$key$: $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$
//$values$: collection of $v = m_{ij} \times n_{jk}$
emit$_{\mathcal{R}_2\to\mathcal{P}}((i, k), \sum_{((i,k),v)} v)$
```

(c) 2nd Reduce function (2R-R2).

Computation cost: $C_+ n^3$.
Communication cost: n^2.

```
Input: $(key, value)$
// $key$: id of a chunk of $M$ or $N$
// $value$: collection of $(i, j, m_{ij})$
//                    or $(j, k, n_{jk})$
foreach $(i, j, m_{ij}) \in value$ do
    emit$_{\mathcal{M}\to\mathcal{R}_1}(j, (M, i, \mathcal{E}_{\mathsf{pk}_p}(m_{ij})))$
foreach $(j, k, n_{jk}) \in value$ do
    emit$_{\mathcal{N}\to\mathcal{R}_1}(j, (N, k, (n_{jk} + \tau_{jk}, \mathcal{E}_{\mathsf{pk}_{r_2}}(\tau_{jk}))))$
```

(d) 1st Map function (2R-SP-M1).

Computation cost: $(C_+ + 2C_\mathcal{E})n^2$.
Communication cost: $3n^2$.

```
Input: $(key, values)$
//$key$: $1 \leq j \leq b$
//$values$: collection of $(M, i, \mathcal{E}_{\mathsf{pk}_p}(m_{ij}))$
// or $(N, k, (n_{jk} + \tau_{jk}, \mathcal{E}_{\mathsf{pk}_{r_2}}(\tau_{jk})))$
foreach $(M, i, \mathcal{E}_{\mathsf{pk}_p}(m_{ij})) \in values$ do
    foreach $(N, k, (n_{jk} + \tau_{jk}, \mathcal{E}_{\mathsf{pk}_{r_2}}(\tau_{jk}))) \in values$ do
        emit$_{\mathcal{R}_1\to\mathcal{R}_2}((i, k),$
          $(\mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{n_{jk}+\tau_{jk}}, \mathcal{E}_{\mathsf{pk}_p}(m_{ij}), \mathcal{E}_{\mathsf{pk}_{r_2}}(\tau_{jk})))$
```

(e) 1st Reduce function (2R-SP-R1).

Computation cost: $C_{\exp} n^3$.
Communication cost: $3n^3$.

```
Input: $(key, values)$
//$key$: $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$
//$values$: collection of $(v_1, v_2, v_3) =$
// $(\mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{n_{jk}+\tau_{jk}}, \mathcal{E}_{\mathsf{pk}_p}(m_{ij}), \mathcal{E}_{\mathsf{pk}_{r_2}}(\tau_{jk}))$
emit$_{\mathcal{R}_2\to\mathcal{P}}((i, k), \prod_{((i,k),(v_1,v_2,v_3))} v_1/(v_2^{\mathcal{D}_{\mathsf{sk}_{r_2}}(v_3)}))$
```

(f) 2nd Reduce function (2R-SP-R2).

Computation cost: $(2C_\times + C_{\exp} + C_\mathcal{D})n^3$.
Communication cost: n^2.

```
Input: $(key, value)$
// $key$: id of a chunk of $M$ or $N$
// $value$: collection of $(i, j, m_{ij})$
//                    or $(j, k, n_{jk})$
foreach $(i, j, m_{ij}) \in value$ do
    emit$_{\mathcal{M}\to\mathcal{R}_1}(j, (M, i, \mathcal{E}_{\mathsf{pk}_p}(m_{ij})))$
foreach $(j, k, n_{jk}) \in value$ do
    emit$_{\mathcal{N}\to\mathcal{R}_1}(j, (N, k, \mathcal{E}_{\mathsf{pk}_p}(n_{jk})))$
```

(g) 1st Map function (2R-CRSP-M1).

Computation cost: $2C_\mathcal{E} n^2$.
Communication cost: $2n^2$.

```
Input: $(key, values)$
//$key$: $1 \leq j \leq b$
//$values$: collection of $(M, i, \mathcal{E}_{\mathsf{pk}_p}(m_{ij}))$
//                    or $(N, k, \mathcal{E}_{\mathsf{pk}_p}(n_{jk}))$
foreach $(M, i, \mathcal{E}_{pk_p}(m_{ij})) \in values$ do
    foreach $(N, k, \mathcal{E}_{pk_p}(n_{jk})) \in values$ do
        emit$_{\mathcal{R}_1\to\mathcal{R}_2}((i, k),$
          $inter(\mathcal{E}_{\mathsf{pk}_p}(m_{ij}), \mathcal{E}_{\mathsf{pk}_p}(n_{jk})))$
```

(h) 1st Reduce function (2R-CRSP-R1).

Computation cost: $(4C_\mathcal{E} + 6C_\times + 2C_\mathcal{D} + 2C_{\exp})n^3$.
Communication cost: $4n^3$.

```
Input: $(key, values)$
//$key$: $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$
//$values$: collection of $v = \mathcal{E}_{\mathsf{pk}_p}(m_{ij} \times n_{jk})$
emit$_{\mathcal{R}_2\to\mathcal{P}}((i, k), \prod_{((i,k),v)} v)$
```

(i) 2nd Reduce function (2R-CRSP-R2).

Computation cost: $C_\times n^3$.
Communication cost: n^2.

Figure 6: Algorithms for two MapReduce rounds, the highlighting emphasizes differences between algorithms of the first column and the second one, and between algorithms of the second column and the third one. By n we denote $\max(a, b, c)$. Computation and communication costs are given in *big-O* notation.

```
Input: $(key, value)$
//$key$: id of a chunk of $M$ or $N$
//$value$: collection of $(i, j, m_{ij})$ or $(j, k, n_{jk})$
foreach $(i, j, m_{ij}) \in value$ do
    foreach $1 \leq k \leq b$ do
        $\texttt{emit}_{\mathcal{M}\to\mathcal{R}}((i, k), (M, j, m_{ij}))$
foreach $(j, k, n_{jk}) \in value$ do
    foreach $1 \leq i \leq a$ do
        $\texttt{emit}_{\mathcal{N}\to\mathcal{R}}((i, k), (N, j, n_{jk}))$
```

(a) Map function (1R-M).

Computation cost: $2n^2$.
Communication cost: $2n^3$.

```
Input: $(key, values)$
//$key$: $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$
//$values$: collection of $(M, j, m_{ij})$
//                     or $(N, j, n_{jk})$
foreach $j \in [\![1, b]\!]$ do
    $\texttt{emit}_{\mathcal{R}\to\mathcal{P}}((i, k), \sum_{j=1}^{b}(m_{ij} \times n_{jk}))$
```

(b) Reduce function (1R-R).

Computation cost: $(C_+ + C_\times)n^3$.
Communication cost: n^2.

```
Input: $(key, value)$
//$key$: id of a chunk of $M$ or $N$
//$value$: collection of $(i, j, m_{ij})$ or $(j, k, n_{jk})$
foreach $(i, j, m_{ij}) \in value$ do
    foreach $1 \leq k \leq c$ do
        $\texttt{emit}_{\mathcal{M}\to\mathcal{R}}((i, k), (M, j, \mathcal{E}_{\mathsf{pk}_p}(m_{ij})))$
foreach $(j, k, n_{jk}) \in value$ do
    foreach $1 \leq i \leq a$ do
        $\texttt{emit}_{\mathcal{N}\to\mathcal{R}}((i, k), (N, j, n_{jk}))$
```

(c) Map function (1R-SP-M).

Computation cost: $(1 + C_\mathcal{E})n^2$.
Communication cost: $2n^3$.

```
Input: $(key, values)$
//$key$: $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$
//$values$: collection of $(M, j, \mathcal{E}_{\mathsf{pk}_p}(m_{ij}))$
//                      or $(N, j, n_{jk})$
foreach $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$ do
    $\texttt{emit}_{\mathcal{R}\to\mathcal{P}}((i, k), \prod_{j=1}^{b} \mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{n_{jk}})$
```

(d) Reduce function (1R-SP-R).

Computation cost: $(C_\times + C_{\exp})n^3$.
Communication cost: n^2.

```
Input: $(key, value)$
//$key$: id of a chunk of $M$ or $N$
//$value$: collection of $(i, j, m_{ij})$ or $(j, k, n_{jk})$
foreach $(i, j, m_{ij}) \in value$ do
    foreach $1 \leq k \leq c$ do
        $\texttt{emit}_{\mathcal{M}\to\mathcal{R}}((i, k), (M, j, \mathcal{E}_{\mathsf{pk}_p}(m_{ij})))$
foreach $(j, k, n_{jk}) \in value$ do
    foreach $1 \leq i \leq a$ do
        $\texttt{emit}_{\mathcal{N}\to\mathcal{R}}((i, k), (N, j, \mathcal{E}_{\mathsf{pk}_p}(n_{jk})))$
```

(e) Map function (1R-CRSP-M).

Computation cost: $2C_\mathcal{E}n^2$.
Communication cost: $2n^3$.

```
Input: $(key, values)$
//$key$: $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$
//$values$: collection of $(M, j, \mathcal{E}_{\mathsf{pk}_p}(m_{ij}))$
//                      or $(N, j, \mathcal{E}_{\mathsf{pk}_p}(n_{jk}))$
foreach $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$ do
    $\texttt{emit}_{\mathcal{R}\to\mathcal{P}}((i, k), \prod_{j=1}^{b} inter(\mathcal{E}_{\mathsf{pk}_p}(m_{ij}), \mathcal{E}_{\mathsf{pk}_p}(n_{jk})))$
```

(f) Reduce function (1R-CRSP-R).

Computation cost $(4C_\mathcal{E} + 7C_\times + 2C_\mathcal{D} + 2C_{\exp})n^3$.
Communication cost: $3n^3 + n^2$

Figure 7: Algorithms for one MapReduce round, the highlighting emphasizes differences between algorithms of the first column and the second one, and between algorithms of the second column and the third one. By n we denote $\max(a, b, c)$. Computation and communication costs are given in *big-O* notation.

with $1 \leq j \leq b$ for each key (i, k). These results are paired to the keys (i, k) for $1 \leq i \leq a$ and $1 \leq k \leq c$ in the output of 1R-R and sent to the MapReduce user $\mathcal{P}$.

4 SP Matrix Multiplication

The SP matrix multiplication with MapReduce uses the Paillier's scheme and the idea of adding a random mask to ensure privacy of elements of matrices.

4.1 SP Two MapReduce Rounds

The matrix multiplication for SP two MapReduce rounds is composed of four functions: the first Map function (2R-SP-M1), the first Reduce function (2R-SP-R1), the second Map function (2R-SP-M2) and the second Reduce function (2R-SP-R2).

The First SP Map Function (Figure 6(d)). 2R-SP-M1 encrypts all elements m_{ij} of the matrix M using Paillier's cryptosystem with the public key pk_p of user $\mathcal{P}$ and masks all elements n_{jk} of the matrix N by adding a random element $\tau_{jk} \in \mathbb{Z}_n$ to n_{jk}. Moreover, 2R-SP-M1 encrypts and sends each τ_{jk} with the public key pk_{r_2} of nodes $\mathcal{R}_2$.

The First SP Reduce Function (Figure 6(e)). 2R-SP-R1 is executed on the set of nodes $\mathcal{R}_1$. As we have seen in the standard MapReduce, 2R-R1 produces key-value pairs where the key is equal to (i, k) and values is equal to $m_{ij} \cdot n_{jk}$ for $1 \leq j \leq b$. In the SP approach, keys are also equal to (i, k) but values are tuples equal to $(\mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{n_{jk}+\tau_{jk}}, \mathcal{E}_{\mathsf{pk}_p}(m_{ij}), \mathcal{E}_{\mathsf{pk}_{\mathcal{R}_2}}(\tau_{jk}))$. Thus, $\mathcal{R}_1$ can compute $\mathcal{E}_{\mathsf{pk}_p}(m_{ij} \times n_{jk}) \times \mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{\tau_{jk}}$ using homomorphic properties of Paillier's cryptosystem which is equal to $\mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{n_{jk}+\tau_{jk}}$ in the tuple. The mask removal is later on done in nodes $\mathcal{R}_2$.

The Second SP Map Function. It is also the identity.

The Second SP Reduce Function (Figure 6(f)). 2R-SP-R2, executed on $\mathcal{R}_2$, multiplies all values associated to the same key (i,k). Moreover, $\mathcal{R}_2$ removes all masks for each value using $\mathcal{E}_{\mathsf{pk}_p}(m_{ij})$ and $\mathcal{E}_{\mathsf{pk}_{r_2}}(\tau_{jk})$ emitted by $\mathcal{R}_1$. In fact, to compute $\mathcal{E}_{\mathsf{pk}_p}(m_{ij} \times n_{jk})$, $\mathcal{R}_2$ computes

$$\begin{aligned}
&\mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{n_{jk}+\tau_{jk}} / \left(\mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{\mathcal{D}_{\mathsf{sk}_{r_2}}(\mathcal{E}_{\mathsf{sk}_{r_2}}(\tau_{jk}))}\right) \\
&\qquad = \mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{n_{jk}} \times \mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{\tau_{jk}} / \mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{\tau_{jk}} \\
&\qquad = \mathcal{E}_{\mathsf{pk}_p}(m_{ij} \times n_{jk}) \ .
\end{aligned}$$

4.2 SP One MapReduce Round

The matrix multiplication for SP one MapReduce round is composed of two functions: the Map function (1R-SP-M) and the Reduce function (1R-SP-R). Unlike in two MapReduce rounds, elements of matrix N cannot be masked since there is only one reduce function.

The SP Map Function (Figure 7(c)). In 1R-SP-M, only elements of matrix M are encrypted with the public key pk_p of the user $\mathcal{P}$ using Paillier's cryptosystem.

The SP Reduce Function (Figure 7(d)). 1R-SP-R uses the homomorphic property of Paillier's cryptosystem. In fact, instead to summing all products $m_{ij} \times n_{jk}$; 1R-SP-R mutltiplies, for each key (i,k), all encrypted values $\mathcal{E}_{\mathsf{pk}_p}(m_{ij})^{n_{jk}}$ corresponding to $\mathcal{E}_{\mathsf{pk}_p}(m_{ij} \times n_{jk})$.

5 CRSP Matrix Multiplication

The MapReduce matrix multiplication with one and two rounds reveals all intermediate results to each set of nodes of the cluster. For example, when matrix multiplication is performed with two MapReduce rounds, $\mathcal{R}_1$ knows all elements of matrices M and N. We describe below MapReduce algorithms with the CRSP approach, which precludes the cluster nodes from learning elements of the matrices.

5.1 CRSP Two MapReduce Rounds

The matrix multiplication for CRSP two MapReduce rounds is composed of four functions: the first Map function (2R-CRSP-M1), the first Reduce function (2R-CRSP-R1), the second Map function (2R-CRSP-M2) and the second Reduce function (2R-CRSP-R2).

The First CRSP Map Function (Figure 6(g)). Unlike 2R-SP-M1, 2R-CRSP-M1 encrypts all elements m_{ij} of the matrix M and all elements n_{jk} of the matrix N with the public key pk_p of the user $\mathcal{P}$. Hence, when 2R-CRSP-M1 emits the key-value pairs to $\mathcal{R}_1$, which learns nothing about elements of matrices M and N.

The First CRSP Reduce Function (Figure 6(h)). It has to perform multiplications of encrypted values having the same key on $\mathcal{R}_1$, i.e. for a key (i,k), $\mathcal{R}_1$ multiplies $\mathcal{E}_{\mathsf{pk}_p}(m_{ij})$ by $\mathcal{E}_{\mathsf{pk}_p}(n_{jk})$ for $1 \leq j \leq b$. We use the interactive protocol in Figure 5 to perform mutiplication of two ciphertexts with the Paillier's cryptosystem. Hence, $\mathcal{R}_1$ interacts with the client $\mathcal{P}$ to perform the multiplications.

The Second CRSP Map Function. It is also the identity.

The Second CRSP Reduce Function (Figure 6(i)). Since $\mathcal{R}_2$ receives key-value pairs where values v equal to $\mathcal{E}_{\mathsf{pk}_p}(m_{ij} \times n_{jk})$, $\mathcal{R}_2$ uses the homomorphic property of Paillier's cryptosystem to compute the sum of all encrypted values associated to the same key (i,k). Pricisely, 2R-CRSP-R2 computes $\prod_{((i,k),v)} v$ for a key (i,k) which represents $\mathcal{E}_{\mathsf{pk}_p}(\sum_{((i,k),v)} v)$.

5.2 CRSP One MapReduce Round

Finally, we present the matrix multiplication for CRSP one MapReduce round composed of two functions: the Map function (1R-CRSP-M) and the Reduce function (1R-CRSP-R).

The CRSP Map Function (Figure 7(e)). 1R-CRSP-M creates the sets of matrix elements used to compute the product of matrices M and N. All these matrix elements are encrypted with the public key pk_p of the user $\mathcal{P}$. Hence, $\mathcal{M}$, $\mathcal{N}$ and $\mathcal{R}$ cannot learn any information on elements of these matrices.

The CRSP Reduce Function (Figure 7(f)). For a key (i, k), it takes as input values of the form $(M, j, \mathcal{E}_{\mathsf{pk}_p}(m_{ij}))$ and $(N, j, \mathcal{E}_{\mathsf{pk}_p}(n_{jk}))$, for $1 \leq j \leq b$. To compute $\sum_{j=1}^{b} m_{ij} \times n_{jk}$ from these encrypted values, 1R-CRSP-R uses Paillier's interactive multiplicative homomorphic protocol.

6 Security Proofs

We provide formal security proof of our two rounds CRSP protocol. Proofs for the one round CRSP protocol and for SP protocols are quite similar but having no space for all four, we focus on the most technically interesting.

We use the standard multi-party computations definition of security against semi-honest adversaries [13]. We consider several entities that run a secure protocol in order to evaluate a multivariate function f. For example, consider two parties A and B using respectively inputs a and b that run a secure two-party protocol to evaluate the multivariate function $f = (f_A, f_B)$. At the end of the protocol, A learns $f_A(a, b)$ and B learns $f_B(a, b)$. Such a protocol is *secure* when A (resp. B) learns nothing else than $f_A(a, b)$ about b (resp. $f_B(a, b)$ about a). We consider *semi-honest* adversaries in the sense that A and B run *honestly* the protocols, but they try to exploit all intermediate information that they have received during the protocol.

We model our protocol with five entities $\mathcal{M}, \mathcal{N}, \mathcal{R}_1, \mathcal{R}_2$ and $\mathcal{P}$ using respective inputs $I = (I_{\mathcal{M}}, I_{\mathcal{N}}, I_{\mathcal{R}_1}, I_{\mathcal{R}_2}, I_{\mathcal{P}})$ and a function $f = (f_{\mathcal{M}}, f_{\mathcal{N}}, f_{\mathcal{R}_1}, f_{\mathcal{R}_2}, f_{\mathcal{P}})$ such that:

- $\mathcal{M}$ has the input $I_{\mathcal{M}} = (M, \mathsf{pk}_p)$ where M is a matrix and pk_p is a Paillier's public key, and returns $f_{\mathcal{M}}(I) = \perp$ (where $\perp$ denotes that the function returns nothing), because $\mathcal{M}$ does not learn anything.
- $\mathcal{N}$ has the input $I_{\mathcal{N}} = (N, \mathsf{pk}_p)$ where N is a matrix and pk_p is a Paillier's public key, and returns $f_{\mathcal{N}}(I) = \perp$, because $\mathcal{N}$ does not learn anything.
- $\mathcal{R}_1$ has the input $I_{\mathcal{R}_1} = \mathsf{pk}_p$ where pk_p is a Paillier's public key, and returns $f_{\mathcal{R}_1}(I) = \perp$, because $\mathcal{R}_1$ does not learn anything.
- $\mathcal{R}_2$ has the input $I_{\mathcal{R}_2} = \mathsf{pk}_p$ where pk_p is a Paillier's public key, and returns $f_{\mathcal{R}_2}(I) = \perp$, because $\mathcal{R}_2$ does not learn anything.
- $\mathcal{P}$ has the input $I_{\mathcal{P}} = (\mathsf{pk}_p, sk_p)$ where (pk_p, sk_p) is a Paillier's key pair, and returns $f_{\mathcal{P}}(I) = M \times N$.

Note that for the sake of clarity, we consider that $\mathcal{R}_2$ sends the product of the encrypted matrices to $\mathcal{P}$ instead of storing them in a database. Moreover, we show that our two-rounds protocol is secure even if the two reduce nodes $\mathcal{R}_1$ and $\mathcal{R}_2$ collude, i.e. they share all their information.

We start by formally defining the *Computational Indistinguishability* and the *view* of an entity before formally presenting the security of CRSP MapReduce protocols.

Definition 3 (Computational indistinguishability). *Let η be a security parameter and X_η and Y_η two distributions. We say that X_η and Y_η are* Computationally Indistinguishable*, denoted $X_\eta \overset{\mathrm{c}}{\equiv} Y_\eta$, if for every probabilistic polynomial-time distinguisher $\mathcal{D}$ we have:*

$$|\Pr[x \leftarrow X_\eta : 1 \leftarrow \mathcal{D}(x)] - \Pr[y \leftarrow Y_\eta : 1 \leftarrow \mathcal{D}(y)]| \leq \epsilon(\eta) ,$$

where ϵ is a negligible function in η.

Definition 4 (view). *Let π be a n-parties protocol that computes the function $f = (f_i)_{1 \leq i \leq n}$ for the entites $(E_i)_{1 \leq i \leq n}$ using inputs $I = (I_i)_{1 \leq i \leq n}$. The view of a party E_i (where $1 \leq i \leq n$) during an execution of π, denoted $\textsc{view}^{\pi}_{E_i}(I)$, is the set of all values sent and received by E_i during the protocol. We denote by $\textsc{view}^{\pi}_{E_i,E_j}(I) = (\textsc{view}^{\pi}_{E_i}(I), \textsc{view}^{\pi}_{E_j}(I))$ (where $1 \leq i, j \leq n$) the view of a collusion between E_i and E_j.*

To prove that a party E learns nothing during execution of the protocol, we show that E can run a *simulator* algorithm that simulates the protocol, such that E (or any polynomialy bounded algorithm) is not able to differentiate an execution of the simulator and an execution of the real protocol. The idea is the following: since the entity E is able to generate his view using the simulator without the secret inputs of other entities, E cannot extract any information from his view during the protocol. This notion is formalized in Definition 5.

Definition 5 (Security with respect to semi-honest behavior)**.** *Let π be a n-parties protocol that computes the function $f = (f_i)_{1 \leq i \leq n}$ for entites $(E_i)_{1 \leq i \leq n}$ using inputs $I = (I_i)_{1 \leq i \leq n} \in \mathcal{I}$. We say that π securely computes f* in the presence of semi-honest adversaries *if for each E_i (where $1 \leq i \leq n$) there exists probabilistic polynomial-time simulators S_{E_i} such that:*

$$S_{E_i}(I_i, f_{E_i}(I)) \overset{c}{\equiv} \textsc{view}^{\pi}_{E_i}(I) \ .$$

We say that π is secure against collusions between E_i and E_j (where $1 \leq i, j \leq n$) if there exist probabilistic polynomial-time simulators S_{E_i,E_j} such that:

$$S_{E_i,E_j}((I_i, f_{E_i}(I)), (I_j, f_{E_j}(I))) \overset{c}{\equiv} \textsc{view}^{\pi}_{E_i,E_j}(I) \ .$$

The security of CRSP two MapReduce rounds is given by Theorem 1.

Theorem 1. *Assume Paillier's cryptosystem is IND-CPA, then the CRSP two-rounds protocol securely computes the matrix mutiplication in the presence of semi-honest adversaries even if $\mathcal{R}_1$ and $\mathcal{R}_2$ collude. Moreover, the CRSP one-round protocol securely computes the matrix multiplication in the presence of semi-honest adversaries.*

In a security point of view, the only difference between the two protocols is that the collusion of the nodes $\mathcal{R}_1$ and $\mathcal{R}_2$ in the two-rounds protocol becomes a unique node $\mathcal{R}$ in the one-round protocol, then the security of the two-rounds protocol implies the security of the one-round one.

The security proof for the two-rounds protocol (Theorem 1) is decomposed in Lemma 1 for $\mathcal{M}$ and $\mathcal{N}$, Lemma 2 for $\mathcal{R}_1$ and $\mathcal{R}_2$ and Lemma 3 for $\mathcal{P}$.

Lemma 1. *There exists probabilistic polynomial-time simulators $\mathcal{S}_{\mathcal{M}}$ and $\mathcal{S}_{\mathcal{N}}$ such that for all $I = (I_{\mathcal{M}}, I_{\mathcal{N}}, I_{\mathcal{R}_1}, I_{\mathcal{R}_2}, I_{\mathcal{P}})$ we have:*

$$\mathcal{S}_{\mathcal{M}}(I_{\mathcal{M}}, f_{\mathcal{M}}(I)) \overset{c}{\equiv} \textsc{view}^{\mathsf{CRSP}}_{\mathcal{M}}(I) \ ,$$

$$\mathcal{S}_{\mathcal{N}}(I_{\mathcal{N}}, f_{\mathcal{N}}(I)) \overset{c}{\equiv} \textsc{view}^{\mathsf{CRSP}}_{\mathcal{N}}(I) \ .$$

Proof. We build the simulator $\mathcal{S}_{\mathcal{M}}$ presented in Algorithm 1. The view of $\mathcal{M}$ only contains the encryption of M that is sent to $\mathcal{R}_1$. We remark that $\mathcal{S}_{\mathcal{M}}((M, \mathsf{pk}_p), \perp)$ uses exactly the same algorithm as the real protocol of CRSP, then it describes exactly the same distribution as $\textsc{view}^{\mathsf{CRSP}}_{\mathcal{M}}(I)$, which concludes the proof. Building the simulator $\mathcal{S}_{\mathcal{N}}$ as $\mathcal{S}_{\mathcal{M}}$, we prove that $\mathcal{S}_{\mathcal{N}}((N, \mathsf{pk}_p), \perp)$ describes exactly the same distribution as $\textsc{view}^{\mathsf{CRSP}}_{\mathcal{N}}(I)$. □

```
$S_{\mathcal{M}}((M, \mathsf{pk}_p), \perp)$:
$M' \leftarrow \left(\mathcal{E}_{\mathsf{pk}_p}(m_{ij})\right)_{1 \leq i \leq a, 1 \leq j \leq b}$ ;
$\textsc{view} = (M')$ ;
return $\textsc{view}$.
```

Algorithm 1: Simulator $\mathcal{S}_{\mathcal{M}}$.

Lemma 2. *Assume Paillier's cryptosystem is IND-CPA, then there exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{R}_1,\mathcal{R}_2}$ such that for all $I = (I_{\mathcal{M}}, I_{\mathcal{N}}, I_{\mathcal{R}_1}, I_{\mathcal{R}_2}, I_{\mathcal{P}})$ we have:*

$$\mathcal{S}_{\mathcal{R}_1,\mathcal{R}_2}((I_{\mathcal{R}_1}, f_{\mathcal{R}_1}(I)), (I_{\mathcal{R}_2}, f_{\mathcal{R}_2}(I))) \overset{c}{\equiv} \textsc{view}^{\mathsf{CRSP}}_{\mathcal{R}_1,\mathcal{R}_2}(I) \ .$$

Proof. Let $\mathcal{S}_{\mathcal{R}_1,\mathcal{R}_2}$ be the simulator presented in Algorithm 2. It outputs the view of $\mathcal{R}_1$ that contains the two encrypted matrices M' and N' sent by $\mathcal{M}$ and $\mathcal{N}$, all couples of ciphertexts (x_{ijk}, y_{ijk}) sent to $\mathcal{P}$ and all corresponding ciphertexts z_{ijk} returned by $\mathcal{P}$ to compute multiplication on encrypted coefficients, and all the values z_{ijk} that are forwarded to $\mathcal{R}_2$. It also outputs the view of $\mathcal{R}_2$ that contains all values z_{ijk} sent by $\mathcal{R}_1$ and coefficients u_{ik} of the encrypted matrix P' sent to $\mathcal{P}$.

Let η be the security parameter used for the Paillier's cryptosystem. Assume there exists a polynomial-time distinguisher $\mathcal{D}$ such that for all $I \in \mathcal{I}$:

$$\Big| \Pr[s \leftarrow \mathcal{S}_{\mathcal{R}_1,\mathcal{R}_2}((I_{\mathcal{R}_1}, f_{\mathcal{R}_1}(I)), (I_{\mathcal{R}_2}, f_{\mathcal{R}_2}(I))) : 1 \leftarrow \mathcal{D}(s)] \\ - \Pr[s \leftarrow \text{VIEW}^{\mathsf{CRSP}}_{\mathcal{R}_1,\mathcal{R}_2}(I) : 1 \leftarrow \mathcal{D}(s)] \Big| = \epsilon(\eta) ,$$

where ϵ is a non-negligible function in η. We show how to build a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}$ has a non-negligible advantage to win the $\mathsf{IND\text{-}CPA}$ experiment on the Paillier's cryptosystem. Then we conclude the proof by contraposition. Adversary $\mathcal{A}$ is presented in Algorithm 3. At the end of its execution, $\mathcal{A}$ uses the distinguisher $\mathcal{D}$ to compute the bit b_* before returning it. First, we remark that:

$$\mathsf{Pr}[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\text{Paillier},\mathcal{A}}(\eta) | b = 0] = \\ \Pr[s \leftarrow \text{VIEW}^{\mathsf{CRSP}}_{\mathcal{R}_1,\mathcal{R}_2}(I) : 0 \leftarrow \mathcal{D}(s)] .$$

Indeed, when $b = 0$, the view that $\mathcal{A}$ uses as input for $\mathcal{D}$ is computed as in the real protocol CRSP. Then the probability that the experiment returns 1 (which is the probability that $b_* = b = 0$) is equal to the probability that the distinguisher returns 0 on inputs computed as in the real protocol. On the other hand, we have:

$$\Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\text{Paillier},\mathcal{A}}(\eta) \mid b = 1] = \\ \Pr[s \leftarrow \mathcal{S}_{\mathcal{R}_1,\mathcal{R}_2}((I_{\mathcal{R}_1}, f_{\mathcal{R}_1}(I)), (I_{\mathcal{R}_2}, f_{\mathcal{R}_2}(I))) : 1 \leftarrow \mathcal{D}(s)] .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for $\mathcal{D}$ is computed as in the simulator $\mathcal{S}_{\mathcal{R}_1,\mathcal{R}_2}$. Then the probability that the experiment returns 1 (which is the probability that $b_* = b = 1$) is equal to the probability that the distinguisher returns 1 on inputs computed as in the simulator. Finally, we evaluate the probability that $\mathcal{A}$ wins the experiment, i.e. $b_* = b$:

$\Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\text{Paillier},\mathcal{A}}(\eta)]$

$$\begin{aligned}
&= \Pr[b = 0] \cdot \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\text{Paillier},\mathcal{A}}(\eta) \mid b = 0] \\
&\quad + \Pr[b = 1] \cdot \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\text{Paillier},\mathcal{A}}(\eta) \mid b = 1] \\
&= \frac{1}{2} \cdot \Pr[s \leftarrow \text{VIEW}^{\mathsf{CRSP}}_{\mathcal{R}_1,\mathcal{R}_2}(I) : 0 \leftarrow \mathcal{D}(s)] \\
&\quad + \frac{1}{2} \cdot \Pr[s \leftarrow \mathcal{S}_{\mathcal{R}_1,\mathcal{R}_2}((I_{\mathcal{R}_1}, f_{\mathcal{R}_1}(I)), (I_{\mathcal{R}_2}, f_{\mathcal{R}_2}(I))) : 1 \leftarrow \mathcal{D}(s)] \\
&= \frac{1}{2} \cdot \Pr[s \leftarrow \text{VIEW}^{\mathsf{CRSP}}_{\mathcal{R}_1,\mathcal{R}_2}(I) : 0 \leftarrow \mathcal{D}(s)] \\
&\quad + \frac{1}{2} \cdot \Big(\Pr[s \leftarrow \text{VIEW}^{\mathsf{CRSP}}_{\mathcal{R}_1,\mathcal{R}_2}(I) : 1 \leftarrow \mathcal{D}(s)] \pm \epsilon(\eta)\Big) \\
&= \frac{1}{2} \cdot \Pr[s \leftarrow \text{VIEW}^{\mathsf{CRSP}}_{\mathcal{R}_1,\mathcal{R}_2}(I) : 0 \leftarrow \mathcal{D}(s)] \\
&\quad + \frac{1}{2} - \frac{1}{2} \cdot \Pr[s \leftarrow \text{VIEW}^{\mathsf{CRSP}}_{\mathcal{R}_1,\mathcal{R}_2}(I) : 0 \leftarrow \mathcal{D}(s)] \pm \frac{1}{2} \cdot \epsilon(\eta) \\
&= \frac{1}{2} \pm \frac{\epsilon(\eta)}{2} .
\end{aligned}$$

We deduce the advantage of $\mathcal{A}$:

$$\left| \Pr\left[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\text{Paillier},\mathcal{A}}(\eta)\right] - \frac{1}{2} \right| = \frac{\epsilon(\eta)}{2} .$$

Which concludes the proof by contraposition. □

```
$S_{\mathcal{R}_1,\mathcal{R}_2}((\mathsf{pk}_p,\bot),(\mathsf{pk}_p,\bot))$:
for $1 \leq i \leq a$ do
    for $1 \leq j \leq b$ do $\alpha_{ij} \xleftarrow{\$} \mathbb{Z}_n$;
for $1 \leq j \leq b$ do
    for $1 \leq k \leq c$ do  $\beta_{jk} \xleftarrow{\$} \mathbb{Z}_n$;
$M' \leftarrow (\mathcal{E}_{\mathsf{pk}_p}(\alpha_{ij}))_{1\leq i\leq a,1\leq j\leq b}$ ;
$N' \leftarrow (\mathcal{E}_{\mathsf{pk}_p}(\beta_{jk}))_{1\leq j\leq b,1\leq k\leq c}$ ;
for $1 \leq i \leq a$ do
    for $1 \leq k \leq c$ do
        for $1 \leq j \leq b$ do
            $(r_{ijk}, s_{ijk}, t_{ijk}) \xleftarrow{\$} (\mathbb{Z}_n)^3$ ;
            $x_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(\alpha_{ij}) \times \mathcal{E}_{\mathsf{pk}_p}(r_{ijk})$ ;
            $y_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(\beta_{jk}) \times \mathcal{E}_{\mathsf{pk}_p}(s_{ijk})$ ;
            $z_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(t_{ijk})$ ;
        $u_{ik} = \prod_{j=1}^{b} z_{ijk}$ ;
$\text{VIEW}_{\mathcal{R}_1} = (M', N', \{(x_{ijk}, y_{ijk}), z_{ijk}\}_{1\leq i\leq a,1\leq j\leq b,1\leq k\leq c})$ ;
$\text{VIEW}_{\mathcal{R}_2} = (\{z_{ijk}\}_{1\leq i\leq a,1\leq j\leq b,1\leq k\leq c}, \{u_{ik}\}_{1\leq i\leq a,1\leq k\leq c})$ ;
return $\text{VIEW} = (\text{VIEW}_{\mathcal{R}_1}, \text{VIEW}_{\mathcal{R}_2})$.
```

Algorithm 2: Simulator $\mathcal{S}_{\mathcal{R}_1,\mathcal{R}_2}$.

Lemma 3. *There exist a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{P}}$ such that for all $I = (I_{\mathcal{M}}, I_{\mathcal{N}}, I_{\mathcal{R}_1}, I_{\mathcal{R}_2}, I_{\mathcal{P}})$ we have:*

$$\mathcal{S}_{\mathcal{P}}(I_{\mathcal{P}}, f_{\mathcal{P}}(I)) \overset{\mathsf{C}}{\equiv} \text{VIEW}_{\mathcal{P}}^{\mathsf{CRSP}}(I) \ .$$

Proof. We build the simulator $\mathcal{S}_{\mathcal{P}}$ presented in Algorithm 4. The view of $\mathcal{P}$ contains the couple of ciphertexts (x_{ijk}, y_{ijk}) sent by $\mathcal{R}_1$ and the answer z_{ijk} that contains the encryption of the multiplication of x_{ijk} and y_{ijk}. Since x_{ijk} and y_{ijk} are randomized by $\mathcal{R}_1$, there are indistinguishable to random ciphertexts in the $\mathcal{P}$ point of view. The view of $\mathcal{P}$ also contains $P' = \mathcal{E}_{\mathsf{pk}_p}(P)$ encrypted by pk_p that is sent by $\mathcal{R}_2$. Finally, $\mathcal{S}_{\mathcal{P}}((\mathsf{pk}_p, sk_{\mathcal{P}}), P)$ describes exactly the same distribution as $\text{VIEW}_{\mathcal{P}}^{\mathsf{CRSP}}(I)$, which concludes the proof. □

7 Conclusion and Future Works

We have presented efficient algorithms for MapReduce matrix multiplication that enjoy privacy guarantees such as: none of the nodes storing an input matrix can learn the other input matrix or the output matrix, and moreover, none of the nodes computing an intermediate result can learn the input or the output matrices. To achieve our goal, we have relied on Paillier's cryptosystem and we developed two different approaches: one resisting to collusions between nodes (Collision-Resistant-Secure-Private) and an other which does not require communication overhead (Secure-Private). We have thoroughly compared these two approaches 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. Additionally, we aim to investigate the matrix multiplication 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 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 Promo-

```
$\mathcal{A}(\mathsf{pk}_p)$:

for $1 \le i \le a$ do
    for $1 \le j \le b$ do $m_{ij} \xleftarrow{\$} \mathbb{Z}_n$;
for $1 \le j \le b$ do
    for $1 \le k \le c$ do  $n_{jk} \xleftarrow{\$} \mathbb{Z}_n$;
$M = (m_{ij})_{1 \le i \le a, 1 \le j \le b}$ ;
$N = (n_{jk})_{1 \le j \le b, 1 \le k \le c}$ ;
for $1 \le i \le a$ do
    for $1 \le k \le c$ do
        for $1 \le j \le b$ do
            $(v_{ijk}, w_{ijk}) \xleftarrow{\$} (\mathbb{Z}_n)^2$ ;
            $x_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(v_{ijk})$ ;
            $y_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(w_{ijk})$ ;
            $\theta_{ijk} \xleftarrow{\$} \mathbb{Z}_n$ ;
            $z_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(LR_b(m_{ij} \times n_{jk}, \theta_{ijk}))$ ;
        $u_{ik} = \prod_{j=1}^{b} z_{ijk}$ ;
for $1 \le i \le a$ do
    for $1 \le j \le b$ do
        $\mu_{i,j} \xleftarrow{\$} \mathbb{Z}_n$ ;
        $m'_{ij} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(LR_b(m_{ij}, \mu_{i,j}))$ ;
$M' \leftarrow (m'_{ij})_{1 \le i \le a, 1 \le j \le b}$ ;
for $1 \le j \le b$ do
    for $1 \le k \le c$ do
        $\delta_{j,k} \xleftarrow{\$} \mathbb{Z}_n$ ;
        $n'_{jk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(LR_b(n_{jk}, \delta_{j,k}))$ ;
$N' \leftarrow (n'_{jk})_{1 \le j \le b, 1 \le k \le c}$ ;
$\textsc{view}_{\mathcal{R}_1} = (M', N', \{x_{ijk}, y_{ijk}, z_{ijk}\})_{1 \le i \le a, 1 \le j \le b, 1 \le k \le c})$ ;
$\textsc{view}_{\mathcal{R}_2} = (\{z_{ijk}\}_{1 \le i \le a, 1 \le j \le b, 1 \le k \le c}, \{u_{ik}\}_{1 \le i \le a, 1 \le k \le c})$ ;
$b_* \leftarrow \mathcal{D}(\textsc{view}_{\mathcal{R}_1}, \textsc{view}_{\mathcal{R}_2})$ ;
return $b_*$.
```

Algorithm 3: Adversary $\mathcal{A}$.

```
$\mathcal{S}_{\mathcal{P}}\left((\mathsf{pk}_p, sk_{\mathcal{P}}), P\right)$:

for $1 \le i \le a$ do
    for $1 \le k \le c$ do
        for $1 \le j \le b$ do
            $(r_{ijk}, s_{ijk}) \xleftarrow{\$} (\mathbb{Z}_n)^2$ ;
            $x_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(r_{ijk})$ ;
            $y_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(s_{ijk})$ ;
            $z_{ijk} \leftarrow \mathcal{E}_{\mathsf{pk}_p}(r_{ijk} \cdot s_{ijk})$ ;
$(p_{ik})_{1 \le i \le a, 1 \le k \le c} = P$;
$P' \leftarrow \left(\mathcal{E}_{\mathsf{pk}_p}(p_{jk})\right)_{1 \le i \le a, 1 \le k \le c}$ ;
$\textsc{view} = (\{(x_{ijk}, y_{ijk}), z_{ijk}\}_{1 \le i \le a, 1 \le j \le b, 1 \le k \le c}, P')$ ;
return $\textsc{view}$.
```

Algorithm 4: Simulator $\mathcal{S}_{\mathcal{P}}$.

tion De La Recherche Avancée (CEFIPRA) through the project DST/CNRS 2015-03 under DST-INRIA-CNRS Targeted Programme.

References

[1] Artak Amirbekyan and Vladimir Estivill-Castro. A new efficient privacy-preserving scalar product protocol. In *AusDM*, volume 70 of *CRPIT*, pages 209–214, 2007.

[2] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT*, 2000.

[3] Erik-Oliver Blass, Roberto Di Pietro, Refik Molva, and Melek Önen. PRISM - privacy-preserving search in mapreduce. In *PETS*, volume 7384, pages 180–200, 2012.

[4] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT*, pages 280–299, 2001.

[5] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC*, pages 119–136, 2001.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[7] Philip Derbeko, Shlomi Dolev, Ehud Gudes, and Shantanu Sharma. Security and privacy aspects in mapreduce on clouds: A survey. *Computer Science Review*, 20:1–28, 2016.

[8] Shlomi Dolev, Yin Li, and Shantanu Sharma. Private and secure secret shared mapreduce. In *DBSec*, volume 9766, pages 151–160, 2016.

[9] W. Du and M.J. Atallah. Privacy-preserving cooperative statistical analysis. In *ACSAC*, pages 102–110, 2001.

[10] Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. Private multi-party matrix multiplication and trust computations. In *SECRYPT*, pages 61–72, July 2016.

[11] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[12] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[13] Qingkai Ma and Ping Deng. Secure multi-party protocols for privacy preserving data mining. In *WASA*, pages 526–537, 2008.

[14] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. PIRMAP: efficient private information retrieval for mapreduce. In *Financial Cryptography and Data Security*, volume 7859, pages 371–385, 2013.

[15] D. Naccache and J. Stern. A new public key cryptosystem based on higher residues. In *CCS*, pages 59–66, 1998.

[16] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *EUROCRYPT*, pages 308–318, 1998.

[17] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[18] Kazue Sako. Goldwasser-micali encryption scheme. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, page 516. 2011.

[19] Triet D. Vo-Huu, Erik-Oliver Blass, and Guevara Noubir. Epic: Efficient privacy-preserving counting for mapreduce. In *NETYS*, volume 9466, pages 426–443, 2015.

[20] I. Wang, C. Shen, T. Hsu, C. Liao, D. Wang, and J. Zhan. Towards empirical aspects of secure scalar product. In *ISA*, pages 573–578, 2008.