Secure Strassen-Winograd Matrix Multiplication with MapReduce[a]

Radu Ciucanu[1], Matthieu Giraud[2], Pascal Lafourcade[2] and Lihua Ye[3]
[1]*INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022, Bourges, France*
[2]*LIMOS, UMR 6158, Université Clermont Auvergne, Aubière, France*
[2]*Harbin Institute of Technology, China*
radu.ciucanu@insa-cvl.fr, {matthieu.giraud, pascal.lafourcade}@uca.fr, 352263187@qq.com

Keywords: Matrix multiplication, Strassen-Winograd algorithm, MapReduce, Security

Abstract: Matrix multiplication is a mathematical brick for solving many real life problems. We consider the Strassen-Winograd algorithm (SW), one of the most efficient matrix multiplication algorithm. Our first contribution is to redesign SW algorithm MapReduce programming model that allows to process big data sets in parallel on a cluster. Moreover, our main contribution is to address the inherent security and privacy concerns that occur when outsourcing data to a public cloud. We propose a secure approach of SW with MapReduce called S2M3, for *Secure Strassen-Winograd Matrix Multiplication with Mapreduce*. We prove the security of our protocol in a standard security model and provide a proof-of-concept empirical evaluation suggesting its efficiency.

1 INTRODUCTION

Matrix multiplication is a mathematical tool of many problems spanning over a plethora of domains e.g., statistics, medicine, or web ranking. Indeed, Markov chains applications on genetics and sociology (Chartrand, 1985), or applications such that computation of shortest paths (Shoshan and Zwick, 1999; Zwick, 1998), convolutional neural network (Krizhevsky et al., 2012) deal with sensitive data processed as matrix multiplication. In such applications, the size of the matrices to be multiplied is often very large. Whereas a naive matrix multiplication algorithm has cubic complexity, many research efforts have been made to propose more efficient algorithms. One of the most efficient algorithms is Strassen-Winograd (Strassen, 1969) (denoted as SW in the sequel), the first sub-cubic time algorithm, with an exponent $\log_2 7 \approx 2.81$. The best algorithm known to date (Le Gall, 2014) has an exponent ≈ 2.38. Although many of the sub-cubic algorithms are not necessarily suited for practical use as their hidden constant in the big-O notation is huge, the SW algorithm and its variants emerged as a class of matrix multiplication algorithms in widespread use.

In this paper, we propose a distributed version of SW that relies on the popular MapReduce paradigm (Dean and Ghemawat, 2004) for outsourcing data and computations to the cloud. Indeed, with the development of the cloud, outsourcing data and computations is nowadays a common fact. A plethora of cloud service providers (e.g., Google Cloud Platform, Amazon Web Services, Microsoft Azure) are available. They allow companies to use large data storage and computation resources on demand for a reasonable price. Despite these benefits, cloud providers do not usually address the fundamental problem of protecting the privacy of users' data. In our case, we consider the problem of matrix multiplication, hence we aim at preserving the privacy of input and output matrices. With a naive algorithm, the cloud would learn all matrices, which may contain sensitive information that the owner of the matrices does not want to disclose.

Problem statement. The data owner has two compatible matrices A and B. The final user is allowed to query the product $C = A \times B$, but is not allowed to know the input matrices A and B.

First, the data owner is expected to encrypt the input matrices A and B before outsourcing them to the public cloud. The matrices are then spread over a set of nodes of the public cloud to run the first phase called the *deconstruction* phase. Then, these results are used by the second phase called the *combination* phase. Finally, the encryption of $C = A \times B$ is sent to the user for decryption. We expect the following

[a]This research was conducted with the support of the FEDER program of 2014-2020, the region council of Auvergne-Rhône-Alpes.

properties:

1. the user cannot learn any information about input matrices A and B,
2. the public cloud cannot learn any information about matrices A, B, and C.

Contributions. We summarize our contributions as follows:

- Our first contribution is a MapReduce version of the SW matrix multiplication algorithm. We call our algorithm SM3 for *Strassen-Winograd Matrix Multiplication with MapReduce*. It improves the efficiency of the computation compared to the standard matrix multiplication with MapReduce, as found in Chapter 2 of (Leskovec et al., 2014).
- Our second contribution is a privacy-preserving version of the aforementioned algorithm. Our new algorithm S2M3 (for *Secure Strassen-Winograd Matrix Multiplication with MapReduce*) relies on the MapReduce paradigm and on Paillier-like public-key cryptosystem. The cloud performs the multiplication on the encrypted data. At the end of the computation, the public cloud sends the result to the user that queried the matrix multiplication result. The user has just to decrypt the result to discover the matrix multiplication result. The cloud cannot learn none of the input or output matrices. We formally prove in the extended paper[1], using a standard security model, that our S2M3 protocol satisfies the aforementioned security property.
- To show the practical efficiency of SM3 and S2M3 protocols, we present a proof-of-concept experimental study using the Apache Hadoop[2] open-source implementation of MapReduce.

Related work. Chapter 2 of (Leskovec et al., 2014) presents an introduction to the MapReduce paradigm. The security and privacy concerns of MapReduce have been summarized in a survey by Derbeko et al. (Derbeko et al., 2016). More precisely, the state-of-the-art techniques for execution of MapReduce computations while preserving privacy focus on problems such as word search (Blass et al., 2012), information retrieval (Mayberry et al., 2013), grouping and aggregation queries (Ciucanu et al., 2018), equijoins (Dolev et al., 2016), and matrix multiplication (Bultel et al., 2017). The general goal of these works is to execute MapReduce computations such that the public cloud cannot learn any information on the input or output data.

In this paper, we focus on the matrix multiplication computation. Recently, (Bultel et al., 2017) secured the two standard MapReduce algorithms for matrix multiplication using one and two MapReduce rounds as found in Chapter 2 of (Leskovec et al., 2014). For each algorithm, they proposed two different approaches called SP for *Secure-Private* and CRSP for *Collision-Resistance Secure-Private*. The two approaches are based on the Paillier-like somewhat homomorphic cryptosystem. Contrary to the CRSP approach, the SP approach assumes that different nodes of the cloud do not collude.In our paper, we assume that all nodes of the public cloud can collude, hence we consider the public cloud as only one entity; hence our secure protocol S2M3 can be considered as a CRSP approach. We show that the matrix multiplication is performed faster using the Strassen-Winograd algorithm with MapReduce than with the standard matrix multiplication with MapReduce for the no-secure and the secure approaches.

Distributed matrix multiplication has been thoroughly investigated in the secure multi-party computation model (MPC) (Du and Atallah, 2001; Dumas et al., 2016; Dumas et al., 2017; Amirbekyan and Estivill-Castro, 2007; Wang et al., 2009), 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 (Dean and Ghemawat, 2004) 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) while the MPC model relies on more complex functions than map and reduce. Moreover, generic MPC protocols (Ma and Deng, 2008; Cramer et al., 2001) allow several nodes to securely evaluate any function such that matrix multiplication computation. 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 Strassen-Winograd algorithm with MapReduce.

Moreover, (Li et al., 2011) and (ul Hassan Khan et al., 2016) study optimizations to compute Strassen-Winograd matrix multiplication. However, they do not consider any privacy issues and do not propose a

[1] https://hal.archives-ouvertes.fr/hal-02129149

[2] https://hadoop.apache.org

secure approach of the algorithm.

To the best of our knowledge, we are the first to secure the Strassen-Winograd algorithm with the MapReduce paradigm.

Outline. In Section 2, we outline the SW algorithm and the cryptographic tools on which we rely. Then, we present the Strassen-Winograd algorithm with MapReduce (SM3) in Section 3, the secure approach (S2M3) in Section 4, and experimental results in Section 5.

2 PRELIMINARIES

We start by recalling the Winograd's variant of Strassen algorithm (Gathen and Gerhard, 2013) called Strassen-Winograd algorithm and denoted SW in the rest of the paper. Then we give cryptographic tools used in the rest of the paper.

2.1 Strassen-Winograd Algorithm

Let A and B be two square matrices of size $d \times d$ where $d = 2^k$ with $k \in \mathbb{N}^\star$. The standard matrix multiplication algorithm computes the product $C = A \times B$ in $O(d^3)$ while SW computes C in $O(d^{2.807})$ (Strassen, 1969).

First, the SW algorithm splits matrices A and B into four quadrants of equal dimensions such that $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ and $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$.

Using these submatrices, SW computes 8 additions, 7 recursive multiplications, and 7 final additions as follow:

$$
\begin{array}{ll}
S_1 = A_{21} + A_{22}, & T_1 = B_{12} - B_{11}, \\
S_2 = S_1 - A_{11}, & T_2 = B_{22} - T_1, \\
S_3 = A_{11} - A_{21}, & T_3 = B_{22} - B_{12}, \\
S_4 = A_{12} - S_2, & T_4 = T_2 - B_{21}, \\
R_1 = A_{11} \times B_{11}, & C_1 = R_1 + R_2, \\
R_2 = A_{12} \times B_{21}, & C_2 = R_1 + R_6, \\
R_3 = S_4 \times B_{22}, & C_3 = C_2 + R_7, \\
R_4 = A_{22} \times T_4, & C_4 = C_2 + R_5, \\
R_5 = S_1 \times T_1, & C_5 = C_4 + R_3, \\
R_6 = S_2 \times T_2, & C_6 = C_3 - R_4, \\
R_7 = S_3 \times T_3, & C_7 = C_3 + R_5.
\end{array}
$$

The final result: $A \times B = \begin{bmatrix} C_1 & C_5 \\ C_6 & C_7 \end{bmatrix}$.

Since, we assume that $d = 2^k$ with $k \in \mathbb{N}^\star$, we can iterate this process k times (recursively) until the submatrices have a size equal to 1×1. The SW algorithm is generalizable to matrices for wich d is not a power of 2, as we point out in Section 4.3.

2.2 Cryptographic tools

We recall the definition of a public-key encryption scheme and the the Paillier's cryptosystem used in our secure protocol.

Definition 1 (Public-Key Encryption)**.** *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)*.*

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

$\mathcal{D}_{sk}$**:** *returns the plaintext m.*

In the following, we require an additive homomorphic encryption scheme to secure the computation of SW using MapReduce. There exist several schemes that have this property (Okamoto and Uchiyama, 1998; Paillier, 1999; Damgård and Jurik, 2001; Naccache and Stern, 1998). We choose Paillier cryptosystem (Paillier, 1999) to illustrate specific required homomorphic properties. Our protocols and proofs are generic, since any other encryption schemes having such properties can be used.

2.3 Paillier's Cryptosystem

Paillier's cryptosystem is an *indistinguishable under chosen plaintext attack* (IND-CPA) scheme (Sako, 2011), 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 $\mathbb{Z}_n^\times$ the set of inversible elements of $\mathbb{Z}_n$. The public key pk of Paillier cryptosystem is (n, g), where $g \in \mathbb{Z}_{n^2}^\times$ and $n = p \cdot q$ 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 $L(x) = (x-1)/n$.

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

Decryption algorithm. Let c be a ciphertext such that $c \in \mathbb{Z}_{n^2}^\times$. We denote by $\mathcal{D}_{sk}(\cdot)$ the decryption function of c with the secret key $sk = (\lambda, \mu)$ defined as follows: $m = L\left(c^\lambda \mod n^2\right) \cdot \mu \mod n$.

2.4 Homomorphic properties

Paillier cryptosystem is a partial homomorphic encryption scheme. We present its 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 $pk = (n, g)$, denoted $c_1 = \mathcal{E}_{pk}(m_1) = g^{m_1} \cdot r_1^n \mod n^2$ and $c_2 = \mathcal{E}_{pk}(m_2) = g^{m_2} \cdot r_2^n \mod n^2$, is the encryption of the sum of m_1 and m_2, i.e., $\mathcal{E}_{pk}(m_1) \cdot \mathcal{E}_{pk}(m_2) = \mathcal{E}_{pk}(m_1 + m_2 \mod n)$. We also remark that: $\mathcal{E}_{pk}(m_1)/\mathcal{E}_{pk}(m_2) = \mathcal{E}_{pk}(m_1 - m_2)$.

Specific Homomorphic Multiplication of Plaintexts. Let m_1 and m_2 be two plaintexts in $\mathbb{Z}_n$ and $c_1 \in \mathbb{Z}_{n^2}^{\times}$ be the ciphertext of m_1 with the public key pk. With Paillier cryptosystem, c_1 raised to the power of m_2 is the encryption of the product of the two plaintexts m_1 and m_2, i.e., $\mathcal{E}_{pk}(m_1)^{m_2} = \mathcal{E}_{pk}(m_1 \cdot m_2 \mod n)$.

Interactive homomorphic multiplication of ciphertexts. Cramer et al. (Cramer et al., 2001) show that an interactive protocol makes possible to perform multiplication over ciphertexts using additive homomorphic encryption schemes. More precisely, Bob knows two ciphertexts $c_1, c_2 \in \mathbb{Z}_{n^2}^{\times}$ 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. In order to do this, Bob has to interact with Alice. Bob first picks two randoms δ_1 and δ_2 and sends to Alice $\alpha_1 = c_1 \cdot \mathcal{E}_{pk_A}(\delta_1)$ and $\alpha_2 = c_2 \cdot \mathcal{E}_{pk_A}(\delta_2)$. By decrypting respectively α_1 and α_2, Alice recovers respectively $m_1 + \delta_1$ and $m_2 + \delta_2$. She sends to Bob $\beta = \mathcal{E}_{pk_A}((m_1 + \delta_1) \cdot (m_2 + \delta_2))$. Then, Bob can deduce the value of $\mathcal{E}(m_1 \cdot m_2)$ by computing: $\beta / \mathcal{E}_{pk_A}(\delta_1 \cdot \delta_2) \cdot c_1^{\delta_1} \cdot c_2^{\delta_2}$, since $\mathcal{E}_{pk_A}((m_1 + \delta_1) \cdot (m_2 + \delta_2)) = \mathcal{E}_{pk_A}(m_1 \cdot m_2) \cdot \mathcal{E}_{pk_A}(m_1 \cdot \delta_2) \cdot \mathcal{E}_{pk_A}(m_2 \cdot \delta_1) \cdot \mathcal{E}_{pk_A}(\delta_1 \cdot \delta_2)$.

3 STRASSEN-WINOGRAD MATRIX MULTIPLICATION

We present our protocol SM3 for Strassen-Winograd Matrix Multiplication with MapReduce. The aim is to compute the multiplication of two square matrices A and B of order $d = 2^k$ (where $k \in \mathbb{N}^{\star}$) using SW algorithm with MapReduce.

The cloud runs two different MapReduce phases: the *deconstruction* phase and the *combination* phase, each of them being repeated $k = \log_2(d)$ times. At the end of the combination phase, the matrix $C = A \times B$ is sent to the user.

We can think of each element $a_{ij} \in A$ (resp. $b_{ij} \in B$) as a tuple $(\mathtt{A}, d, i, j, a_{ij})$ (resp. $(\mathtt{B}, d, i, j, b_{ij})$). Note that A and B are not the matrices themselves but the names of the matrices. In order to run the SW algorithm with MapReduce, a key called *tag* initialized to 0 is added to each tuple. Hence tuples are key-value pairs of the form $(0, (\mathtt{A}, d, i, j, a_{ij}))$ and $(0, (\mathtt{B}, d, i, j, b_{ij}))$. All these key-value pairs establish a relation that is outsourced to the cloud.

3.1 Deconstruction Phase

We present the deconstruction phase of SM3. This phase computes recursively all the needed submatrices of dimension 2×2 to multiply in order to construct the final matrix.

The Map Function. This function is the identity. For every input element with key t and value v, it produces the key-value pair (t, v).

The Reduce Function. This function is presented in Fig. 1. If $\delta > 2$, i.e., the dimension of matrices formed by elements $a_{i,j}$ and $b_{i,j}$ associated to the key t, it produces 7 couples of submatrices of dimension $\delta/2$. These couples of submatrices correspond to the recursive multiplications in SW algorithm and are keyed with a different tag in order to distribute the compution using MapReduce. When $\delta = 2$, the 7 multiplications are between integers (and not between matrices), results are sent to the combination phase.

3.2 Combination Phase

We present the combination phase of SM3 which is executed k times where $k = \log_2(d)$. This phase constructs the matrix $C = A \times B$ using all results received from the deconstruction phase.

The Map Function. This function is the identity. For every input element with key t and value v, it produces the key-value pair (t, v).

The Reduce Function. This function is presented in Fig. 2. At each round, each key is associated to elements forming 7 submatrices of dimension δ. Following SW algorithm, these submatrices are used to construct a matrix of dimension $2 \cdot \delta$. At the end of the combination phase (after k rounds), the reduce function sends to the user key-value pairs of the form $(-, (\mathtt{C}, i, j, c_{ij}))$ forming matrix $C = A \times B$. We do not specify key value since all these elements are part of the final matrix.

Input: (key, values).
// key: a tag t.
// values: collection of $(\mathtt{A}, \delta, i, j, a_{i,j})$
// or $(\mathtt{B}, \delta, i, j, b_{i,j})$.
$A \leftarrow (a_{i,j})_{1 \leq i,j \leq \delta}$; $B \leftarrow (b_{i,j})_{1 \leq i,j \leq \delta}$;
$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = A$; $\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = B$;
$S_1 \leftarrow A_{21} + A_{22}$; $S_2 \leftarrow S_1 - A_{11}$; $S_3 \leftarrow A_{11} - A_{21}$;
$S_4 \leftarrow A_{12} - S_2$; $T_1 \leftarrow B_{12} - B_{11}$; $T_2 \leftarrow B_{22} - T_1$;
$T_3 \leftarrow B_{22} - B_{12}$; $T_4 \leftarrow T_2 - B_{21}$;
$\mathsf{L} \leftarrow \big[[A_{11}, B_{11}], [A_{12}, B_{21}], [S_4, B_{22}], [A_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$;
if $\delta \neq 2$ **then**
  for $1 \leq u \leq 7$ **do**
    $(a^{\star}_{i,j})_{1 \leq i,j \leq \delta/2} = A^{\star} \leftarrow \mathsf{L}[u-1][0]$;
    $(b^{\star}_{i,j})_{1 \leq i,j \leq \delta/2} = B^{\star} \leftarrow \mathsf{L}[u-1][1]$;
    foreach $(v, w) \in [\![1, \delta/2]\!]$ **do**
      emit $(t \| u, (\mathtt{A}, \delta/2, v, w, a^{\star}_{v,w}))$;
      emit $(t \| u, (\mathtt{B}, \delta/2, v, w, b^{\star}_{v,w}))$;
else
  for $1 \leq u \leq 7$ **do**
    $r \leftarrow \mathsf{L}[u-1][0] \cdot \mathsf{L}[u-1][1]$;
    emit $(t, (\mathtt{R}_u, 1, 1, 1, r))$.

Figure 1: Reduce function of the deconstruction phase for SM3.

Input: (key, values).
// key: a tag t.
// values: collection of $(\mathtt{R}_i, \delta, j, k, r_{j,k})$.
for $1 \leq i \leq 7$ **do**
  $R_i \leftarrow (r_{j,k})_{1 \leq j,k \leq \delta}$ such that $(\mathtt{R}_i, \delta, j, k, r_{j,k})$ is in values;
$C_1 \leftarrow R_1 + R_2$; $C_2 \leftarrow R_1 + R_6$; $C_3 \leftarrow C_2 + R_7$;
$C_4 \leftarrow C_2 + R_5$; $C_5 \leftarrow C_4 + R_3$; $C_6 \leftarrow C_3 - R_4$;
$C_7 \leftarrow C_3 + R_5$;
$C = \begin{bmatrix} C_1 & C_5 \\ C_6 & C_7 \end{bmatrix}$;
if $\delta \neq d$ **then**
  foreach $(i, j) \in [\![1, 2 \cdot \delta]\!]$ **do**
    emit $(t[:-1], (\mathtt{R}_{t[-1]}, 2 \cdot \delta, i, j, c_{i,j}))$;
else
  foreach $(i, j) \in [\![1, d]\!]$ **do**
    emit $(-, (\mathtt{C}, i, j, c_{i,j}))$.

Figure 2: Reduce function of the combination phase for SM3.

4 SECURE STRASSEN-WINOGRAD MATRIX MULTIPLICATION

We now present our secure approach, called S2M3 (for Secure Strassen-Winograd Matrix Multiplication with Mapreduce), in order to ensure privacy of elements of matrices. Similarly to SM3, the secured version S2M3 considers matrices A and B as a relation. However, instead of sending key-value pairs of the form $(0, (\mathtt{A}, d, i, j, a_{i,j}))$ and $(0, (\mathtt{B}, d, i, j, b_{i,j}))$ to the cloud, we encrypt each element of matrices and send key-value pairs of the form $(0, (\mathtt{A}, d, i, j, \mathcal{E}_{pk}(a_{i,j})))$ and $(0, (\mathtt{B}, d, i, j, \mathcal{E}_{pk}(b_{i,j})))$. Note that pk is the public key of the user that queried the matrix multiplication to the data owner.

4.1 Deconstruction Phase

We present the deconstruction phase of S2M3. This phase computes recursively all the needed submatrices of dimension 2×2 to multiply in order to construct the final matrix.

The Map Function. This function is the identity. For every input element with key t and value v, it produces the key-value pair (t, v).

The Reduce Function. This function is presented in Fig. 3. It computes 8 submatrices $S_1, \ldots, S_4, T_1, \ldots, T_4$ using functions P.Add and P.Sub. Function P.Add(A, B) (resp. P.Sub(A, B)) computes $\mathcal{E}_{pk}(a_{i,j}) \cdot \mathcal{E}_{pk}(b_{i,j})$ (resp. $\mathcal{E}_{pk}(a_{i,j}) \cdot \mathcal{E}_{pk}(b_{i,j})^{-1}$) for each pair (i, j); due to Paillier homomorphic properties, these multiplications (resp. divisions) of elements is equal to $\mathcal{E}_{pk}(a_{i,j} + b_{i,j})$ (resp. $\mathcal{E}_{pk}(a_{i,j} - b_{i,j})$).

If $\delta > 2$, i.e., the dimension of matrices formed by elements $\mathcal{E}_{pk}(a_{i,j})$ and $\mathcal{E}_{pk}(b_{i,j})$ associated to the key t, it produces 7 couples of submatrices of dimension $\delta/2$. When $\delta = 2$, we need to compute 7 multiplications between integers encrypted with Paillier cryptosystem. Hence we use Paillier interactive multiplication and denoted P.Interactive. Then results are sent to the combination phase.

4.2 Combination Phase

We present the combination phase of S2M3 run k times where $k = \log_2(d)$. This phase constructs the matrix $C' = (\mathcal{E}_{pk}(c_{i,j}))_{1 \leq i,j \leq d}$ with $c_{i,j} \in C = A \times B$ using all results received from the deconstruction phase.

The Map Function. This function is the identity. For every input element with key t and value v, it produces the key-value pair (t, v).

Input: (key, values).
// key: a tag t.
// values: collection of $(\mathtt{A}, \delta, i, j, \mathcal{E}_{pk}(a_{i,j}))$
// or $(\mathtt{B}, \delta, i, j, \mathcal{E}_{pk}(b_{i,j}))$.
$A \leftarrow (\mathcal{E}_{pk}(a_{i,j}))_{1 \leq i,j \leq \delta}$; $B \leftarrow (\mathcal{E}_{pk}(b_{i,j}))_{1 \leq i,j \leq \delta}$;
$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = A$; $\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = B$;
$S_1 \leftarrow \mathsf{P.Add}(A_{21}, A_{22})$; $T_1 \leftarrow \mathsf{P.Sub}(B_{12}, B_{11})$;
$S_2 \leftarrow \mathsf{P.Sub}(S_1, A_{11})$; $T_2 \leftarrow \mathsf{P.Sub}(B_{22}, T_1)$;
$S_3 \leftarrow \mathsf{P.Sub}(A_{11}, A_{21})$; $T_3 \leftarrow \mathsf{P.Sub}(B_{22}, B_{12})$;
$S_4 \leftarrow \mathsf{P.Sub}(A_{11}, S_2)$; $T_4 \leftarrow \mathsf{P.Sub}(T_2, B_{21})$;
$\mathsf{L} \leftarrow \big[[A_{11}, B_{11}], [A_{12}, B_{21}], [S_4, B_{22}], [A_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$;
if $\delta \neq 2$ **then**
  for $1 \leq u \leq 7$ **do**
    $(a^{\star}_{i,j})_{1 \leq i,j \leq \delta/2} = A^{\star} \leftarrow \mathsf{L}[u-1][0]$;
    $(b^{\star}_{i,j})_{1 \leq i,j \leq \delta/2} = B^{\star} \leftarrow \mathsf{L}[u-1][1]$;
    foreach $(v, w) \in [\![1, \delta/2]\!]$ **do**
      emit $(t \| u, (\mathtt{A}, \delta/2, v, w, a^{\star}_{v,w}))$;
      emit $(t \| u, (\mathtt{B}, \delta/2, v, w, b^{\star}_{v,w}))$;
else
  for $1 \leq u \leq 7$ **do**
    $r \leftarrow \mathsf{P.Interactive}(\mathsf{L}[u-1][0], \mathsf{L}[u-1][1])$;
    emit $(t, (\mathtt{R}_u, 1, 1, 1, r))$.

Figure 3: Reduce function of the deconstruction phase for S2M3.

The Reduce Function. This function is presented in Fig. 4. At each round, each key is associated to elements forming 7 submatrices of dimension δ. As for the deconstruction phase, we use Paillier homomorphic properties in order to construct a matrix of dimension $2 \cdot \delta$. At the end of the combination phase (after k rounds), the reduce function sends to the user key-value pairs of the form $(-, (\mathtt{C}', i, j, c_{i,j}))$ forming the encryption of matrix $C = A \times B$. We do not specify key value since all these elements are part of the final matrix.

4.3 Padding and Peeling: On a Quest for All Dimensions

Default SW algorithm works for square matrices of dimension $d = 2^k$ with $k \in \mathbb{N}^{\star}$. In this section, we recall the *dynamic padding* and the *dynamic peeling* (Huss-Lederman et al., 1996) methods allowing to perform matrix multiplication with SW algorithm for square matrices of arbitrary dimension.

Dynamic Padding. For each round of the deconstruction phase, the Reduce function checks the dimension's parity of the two considered matrices. If

Input: (key, values).
// key: a tag t.
// values: collection of $(\mathtt{R}_i, \delta, j, k, r_{j,k})$.
for $1 \leq i \leq 7$ **do**
  $R_i \leftarrow (r_{j,k})_{1 \leq j,k \leq \delta}$ such that $(\mathtt{R}_i, \delta, j, k, r_{j,k})$ is in values;
$C_1 \leftarrow \mathsf{P.Add}(R_1, R_2)$; $C_5 \leftarrow \mathsf{P.Add}(C_4, R_3)$;
$C_2 \leftarrow \mathsf{P.Add}(R_1, R_6)$; $C_6 \leftarrow \mathsf{P.Sub}(C_3, R_4)$;
$C_3 \leftarrow \mathsf{P.Add}(C_2, R_7)$; $C_7 \leftarrow \mathsf{P.Add}(C_3, R_5)$;
$C_4 \leftarrow \mathsf{P.Add}(C_2, R_5)$;
$C' = \begin{bmatrix} C_1 & C_5 \\ C_6 & C_7 \end{bmatrix}$;
if $\delta \neq d$ **then**
  foreach $(i, j) \in [\![1, 2 \cdot \delta]\!]$ **do**
    emit $(t[:-1], (\mathtt{R}_{t[-1]}, 2 \cdot \delta, i, j, c'_{i,j}))$;
else
  foreach $(i, j) \in [\![1, d]\!]$ **do**
    emit $(-, (\mathtt{C}', i, j, c_{i,j}))$.

Figure 4: Reduce function of the combination phase for S2M3.

matrices have an odd number of rows and columns, then it adds an extra row and an extra column of zeros for SM3 protocol, and of $\mathcal{E}_{pk}(0)$ for S2M3 protocol. In the other case, the Reduce function works as usual. Then, the Reduce function of the combination phase removes, when it is required, the extra row and the extra column.

Comparing to the static padding, the dynamic padding avoids huge memory allocations. Indeed, the *static padding* method pads matrices to obtain a number of rows and columns equal to a power-of-2.

Dynamic Peeling. As the previous method, the Reduce function checks for each round of the deconstruction phase the dimension's parity of the two considered matrices. If matrices have an odd number of rows and columns, then it splits each of the two considered matrices into four blocks as illustrated in Fig. 5. Then, for SM3 and S2M3 protocols, it uses the two square blocks for the recursive multiplication, while other blocks are used for the block matrix multiplication. We note that the block matrix multiplication can be performed in the encrypted domain due to the homomorphic properties of Paillier's cryptosystem. Otherwise, the Reduce function works as usual. Then, the Reduce function of the combination phase combines the resulted matrices to build the final result.

$$M = \left[\begin{array}{ccc|c} m_{1,1} & \cdots & m_{1,d-1} & m_{1,d} \\ \vdots & \ddots & \vdots & \vdots \\ m_{d-1,1} & \cdots & m_{d-1,d-1} & m_{d-1,d} \\ \hline m_{d,1} & \cdots & m_{d,d-1} & m_{d,d} \end{array}\right].$$

Figure 5: Splitting of matrix M of dimension d according the dynamic peeling method, with d an odd number.

5 EXPERIMENTAL RESULTS

We present the experimental results for our SM3 and S2M3 protocols using the Hadoop implementation of MapReduce. All computations have been done on a computer running on Ubuntu 16.04 with Hadoop 3.2.0. The computer has an Intel® Core™ i7-4790 CPU cadenced at 3.60GHz, and 16Gb of RAM.

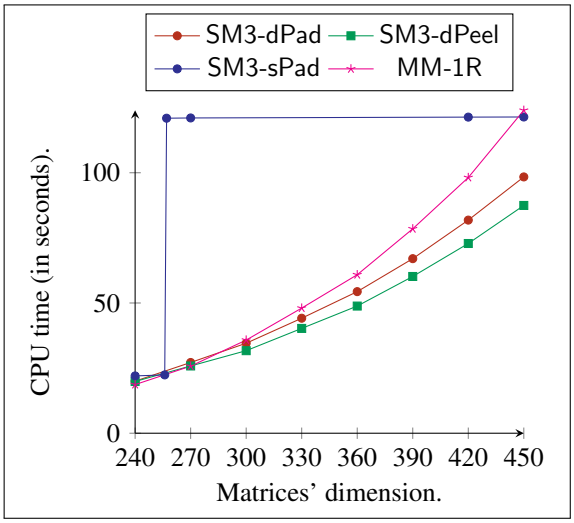

Figure 6: CPU time vs the matrices' dimension for SM3 protocol with the static padding (SM3-sPad), dynamic padding (SM3-dPad), and dynamic peeling (SM3-dPeel), and comparisons with the standard matrix multiplication using one MapReduce round (Leskovec et al., 2014) (MM-1R).

We generate two random square matrices A and B of order d such that $240 \leq d \leq 450$ for the no-secure approaches, and $90 \leq d \leq 300$. Moreover, we build A and B such that $A, B \in \mathbb{Z}_{10}^{d \times d}$. For each order d, we perform matrix multiplication between A and B with SM3 and S2M3 protocols using static padding, dynamic padding, and dynamic peeling methods. We also run the standard matrix multiplication using one MapReduce round (Leskovec et al., 2014) and the secure approach denoted CRSP-1R (Bultel et al., 2017). As proof of concept, we use Paillier's cryptosystem with the public key $pk = (n, g)$ where n is 64-bits long.

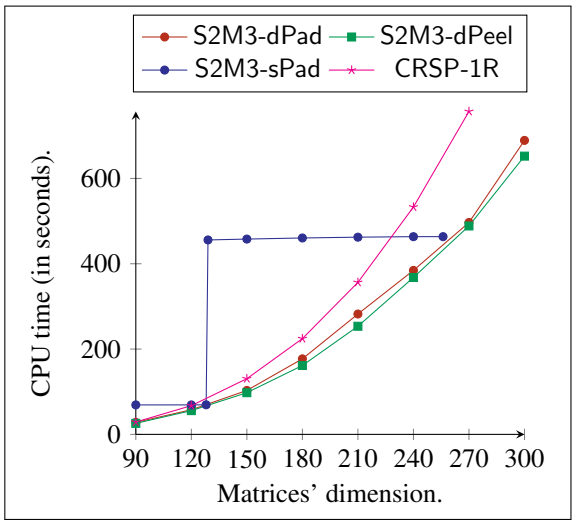

Figure 7: CPU time vs the matrices' dimension for S2M3 protocol with the static padding (S2M3-sPad) and dynamic padding (S2M3-dPad), and the dynamic peeling (S2M3-dPeel), and comparisons with the CRSP-1R protocol (Bultel et al., 2017).

Scalability. We present in Fig. 6 the CPU time for our SM3 protocol using static and dynamic padding methods, and dynamic peeling method. Without any security, our SM3 protocol using dynamic padding and peeling methods performs the matrix multiplication faster than the standard matrix multiplication for the largest dimensions.

For secure protocols, CPU times are presented in Fig. 7. We also remark that our S2M3 protocol using dynamic padding and peeling methods performs the matrix multiplication faster than the CRSP-1R protocol (Bultel et al., 2017).

6 CONCLUSION

We have presented SM3, an efficient algorithm to compute the Strassen-Winograd matrix multiplication using the MapReduce paradigm. We have also presented S2M3, a secure approach of SM3 that satisfies privacy guarantees such that the public cloud does not learn any information on input matrices and on the output matrix. To achieve our goal, we have relied on the well-known Paillier cryptosystem. We have compared our protocol S2M3 to the CRSP matrix multiplication with MapReduce proposed by Bultel et al. (Bultel et al., 2017) and shown that S2M3 is more efficient.

Looking forward to future work, we aim to investigate the matrix multiplication with privacy guarantees in different big data systems (e.g. Spark, Flink)

whose users also tend to outsource data and computations as MapReduce.

REFERENCES

Amirbekyan, A. and Estivill-Castro, V. (2007). A New Efficient Privacy-Preserving Scalar Product Protocol. In *Australasian Data Mining Conference AusDM*.

Blass, E., Pietro, R. D., Molva, R., and Önen, M. (2012). PRISM - privacy-preserving search in mapreduce. In *Privacy Enhancing Technologies Symposium, PETS*.

Bultel, X., Ciucanu, R., Giraud, M., and Lafourcade, P. (2017). Secure matrix multiplication with mapreduce. In *International Conference on Availability, Reliability and Security, ARES*.

Chartrand, G. (1985). *Introductory Graph Theory*. Dover.

Ciucanu, R., Giraud, M., Lafourcade, P., and Ye, L. (2018). Secure Grouping and Aggregation with MapReduce. In *International Joint Conference on e-Business and Telecommunications, SECRYPT*.

Cramer, R., Damgård, I., and Nielsen, J. B. (2001). Multiparty Computation from Threshold Homomorphic Encryption. In *EUROCRYPT, International Conference on the Theory and Application of Cryptographic Techniques*.

Damgård, I. and Jurik, M. (2001). A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography, PKC*.

Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation, OSDI*.

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

Dolev, S., Li, Y., and Sharma, S. (2016). Private and secure secret shared mapreduce (extended abstract) - (extended abstract). In *Data and Applications Security and Privacy XXX - 30th Annual IFIP*.

Du, W. and Atallah, M. J. (2001). Privacy-Preserving Cooperative Statistical Analysis. In *Annual Computer Security Applications Conference ACSAC*.

Dumas, J., Lafourcade, P., Orfila, J., and Puys, M. (2016). Private Multi-party Matrix Multiplication and Trust Computations. In *International Joint Conference on e-Business and Telecommunications SECRYPT*.

Dumas, J., Lafourcade, P., Orfila, J., and Puys, M. (2017). Dual protocols for private multi-party matrix multiplication and trust computations. *Computers & Security*, 71.

Gathen, J. v. z. and Gerhard, J. (2013). *Modern computer algebra*. Cambridge University Press, 3rd edition.

Huss-Lederman, S., Jacobson, E. M., Johnson, J. R., Tsao, A., and Turnbull, T. (1996). Implementation of strassen's algorithm for matrix multiplication. In *ACM/IEEE Conference on Supercomputing*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Annual Conference on Neural Information Processing Systems.*

Le Gall, F. (2014). Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC*.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of Massive Datasets, 2nd Ed*. Cambridge University Press.

Li, J., Ranka, S., and Sahni, S. (2011). Strassen's matrix multiplication on gpus. In *IEEE International Conference on Parallel and Distributed Systems, ICPADS*.

Ma, Q. and Deng, P. (2008). Secure Multi-party Protocols for Privacy Preserving Data Mining. In *Wireless Algorithms, Systems, and Applications Conference, WASA*.

Mayberry, T., Blass, E., and Chan, A. H. (2013). PIRMAP: efficient private information retrieval for mapreduce. In *Financial Cryptography and Data Security, FC*.

Naccache, D. and Stern, J. (1998). A new public key cryptosystem based on higher residues. In *Conference on Computer and Communications Security, CCS*.

Okamoto, T. and Uchiyama, S. (1998). A new public-key cryptosystem as secure as factoring. In *EUROCRYPT, International Conference on the Theory and Application of Cryptographic Techniques*.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT, International Conference on the Theory and Application of Cryptographic Techniques*.

Sako, K. (2011). Goldwasser-micali encryption scheme. In *Encyclopedia of Cryptography and Security, 2nd Ed.*

Shoshan, A. and Zwick, U. (1999). All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science, FOCS*.

Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4).

ul Hassan Khan, A., Al-Mouhamed, M., Fatayer, A., and Nazeeruddin, M. (2016). Optimizing the matrix multiplication using strassen and winograd algorithms with limited recursions on many-core. *International Journal of Parallel Programming*, 44(4).

Wang, I.-C., Shen, C.-H., Zhan, J., Hsu, T.-s., Liau, C.-J., and Wang, D.-W. (2009). Toward empirical aspects of secure scalar product. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(4).

Zwick, U. (1998). All pairs shortest paths in weighted directed graphs exact and almost exact algorithms. In *39th Annual Symposium on Foundations of Computer Science, FOCS*.