

Secure Strassen-Winograd Matrix Multiplication with MapReduce

Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, Lihua Ye

▶ **To cite this version:**

Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, Lihua Ye. Secure Strassen-Winograd Matrix Multiplication with MapReduce. International Conference on Security and Cryptography (SECRYPT), Jul 2019, Prague, Czech Republic. pp.220-227. hal-02129149

HAL Id: hal-02129149
https://hal.science/hal-02129149

Submitted on 14 May 2019

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

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Secure Strassen-Winograd Matrix Multiplication with MapReduce

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, Université Clermont Auvergne, Aubière, France*
[2]*Harbin Institute of Technology, China*
radu.ciucanu@insa-cvl.fr, {matthieu.giraud, pascal.lafourcade}@uca.fr, 16s003041@stu.hit.edu.cn

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 nave 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 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, 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[1] 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 fact, in the SP approach one node of the cloud knows plain values of one of the matrix while in the CRSP approach both matrices are encrypted. In our paper, we consider the public cloud as only one entity in that we assume that all nodes of the cloud can collude; 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., 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.

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

[1] `https://hadoop.apache.org`

with MapReduce (SM3) in Section 3, the secure approach (S2M3) in Section 4, experimental results in Section 5, and we prove in Section 6 that our S2M3 protocol satisfies the desired security properties.

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 negligible function, the definition of a public-key encryption scheme and the security requirements. We also recall the Paillier cryptosystem used in our secure protocol.

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

Definition 2 (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.*

Let $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a PKE scheme, $\mathcal{A}$ be a probabilistic polynomial-time adversary. For $b \in \{0,1\}$, we define the ind-cpa-b experiment where $\mathcal{A}$ has access to the oracle $\mathcal{E}_{pk}(LR_b(\cdot,\cdot))$ taking (m_0, m_1) as input and returns $\mathcal{E}_{pk}(m_0)$ if $b = 0$, $\mathcal{E}_{pk}(m_1)$ otherwise. $\mathcal{A}$ tries to guess the bit b chosen in the experiment. We define the advantage of $\mathcal{A}$ against the IND-CPA experiment by:

$$
\mathsf{Adv}^{\text{ind-cpa}}_{\Pi,\mathcal{A}}(\eta) = \left| \Pr[1 \leftarrow \mathsf{Exp}^{\text{ind-cpa-1}}_{\Pi,\mathcal{A}}(\eta)] - \Pr[1 \leftarrow \mathsf{Exp}^{\text{ind-cpa-0}}_{\Pi,\mathcal{A}}(\eta)] \right|.
$$

We said that Π is IND-CPA if this advantage is negligible for any probabilistic polynomial-time $\mathcal{A}$.

In the following, we require an additive homomorphic encryption scheme to secure the computation of SW using MapReduce. There exists 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 Cryptosystem

Paillier cryptosystem is an 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 *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_{ij} and b_{ij} 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.

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

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.

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 $(0, (\mathtt{A}, d, i, j, a_{ij}))$ and $(0, (\mathtt{B}, d, i, j, b_{ij}))$ to the cloud, we encrypt each element of matrices and send key-value pairs of the form $(0, (\mathtt{A}, d, i, j, \mathcal{E}_{pk}(a_{ij})))$ and $(0, (\mathtt{B}, d, i, j, \mathcal{E}_{pk}(b_{ij})))$. Note that pk is the public key of the user that queried the matrix multiplication to the data owner.

Input: (key, values).
// key: a tag t.
// values: collection of $(\mathtt{R}_i, \delta, j, k, r_{jk})$.
for $1 \leq i \leq 7$ **do**
  $R_i \leftarrow (r_{jk})_{1 \leq j,k \leq \delta}$ such that $(\mathtt{R}_i, \delta, j, k, r_{jk})$ 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**
  for $1 \leq i \leq 2 \cdot \delta$ **do**
    for $1 \leq j \leq 2 \cdot \delta$ **do**
      emit $(t[:-1], (\mathtt{R}_{t[-1]}, 2 \cdot \delta, i, j, c_{ij}))$;
else
  for $1 \leq i \leq d$ **do**
    for $1 \leq j \leq d$ **do**
      emit $(-, (\mathtt{C}, i, j, c_{ij}))$.

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

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 Paillier.Add and Paillier.Sub. Function Paillier.Add(A, B) (resp. Paillier.Sub(A, B)) computes $\mathcal{E}_{pk}(a_{ij}) \cdot \mathcal{E}_{pk}(b_{ij})$ (resp. $\mathcal{E}_{pk}(a_{ij}) / \mathcal{E}_{pk}(b_{ij})$) for each pair (i, j); due to Paillier homomorphic properties, these multiplications (resp. divisions) of elements is equal to $\mathcal{E}_{pk}(a_{ij} + b_{ij})$ (resp. $\mathcal{E}_{pk}(a_{ij} - b_{ij})$).

If $\delta > 2$, i.e., the dimension of matrices formed by elements $\mathcal{E}_{pk}(a_{ij})$ and $\mathcal{E}_{pk}(b_{ij})$ 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 Paillier.Interactive. Then results are sent to the combination phase.

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

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

4.2 Combination Phase

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

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 $(-, (\mathtt{C}', i, j, c_{ij}))$ 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.

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

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

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. We also present how to use dynamic padding and dynamic peeling with our SM3 and S2M3 algorithms.

Dynamic Padding. For each round of the deconstruction phase, we check the dimension of considered matrices. If matrices already have an even number of rows and columns, we do not need to do any-

thing, as we can simply split it into four blocks. However, if the number of rows or the number of columns is odd, i.e., the dimension is odd, we add an extra row or column (or both, if needed). We initialize this row or column to zeros for SM3 protocol and $\mathcal{E}_{pk}(1)$ for S2M3 protocol. Then, we perform the protocol as normal. Once we have multiplied these two matrices, we remove the extra row or column, and return our result.

Using dynamic padding avoids huge memory allocations. In fact, a *static padding* will pad matrices to obtain a number of rows and columns equal to a power of 2.

Dynamic Peeling. Contrary to the previous method, the dynamic peeling method splits a square matrix A (resp. B) having an odd dimension d into four blocks A_1, A_2, A_3, and A_4 (resp. B_1, B_2, B_3, and B_4). Block A_1 (resp. B_1) has dimension $(d-1)$, matrix A_2 (resp. B_2) has dimension $(d-1)\times 1$, matrix A_3 (resp. B_3) has dimension $1\times(d-1)$, and matrix A_4 (resp. B_4) has dimension 1. We illustrate this method in Fig. 5.

$$A = \left[\begin{array}{ccc|c} a_{11} & \dots & a_{1,d-1} & a_{1,d} \\ \vdots & \ddots & \vdots & \vdots \\ a_{d-1,1} & \dots & a_{d-1,d-1} & a_{d-1,d} \\ \hline a_{d,1} & \dots & a_{d,d-1} & a_{d,d} \end{array}\right],$$

$$B = \left[\begin{array}{ccc|c} b_{11} & \dots & b_{1,d-1} & b_{1,d} \\ \vdots & \ddots & \vdots & \vdots \\ b_{d-1,1} & \dots & b_{d-1,d-1} & b_{d-1,d} \\ \hline b_{d,1} & \dots & b_{d,d-1} & b_{d,d} \end{array}\right].$$

Figure 5: Dynamic peeling for matrices A and B of dimension d with d an odd number.

Hence, for each round of the SM3 and S2M3 protocols deconstruction phase (where matrices A and B have an odd order), the reduce function splits both matrices according to the dynamic peeling method. Obtained blocks A_1 and B_1 are then considered as new matrices A and B, and the multiplication is performed following the considered protocol. Other blocks multiplications are computed using the blocks multiplication algorithm and used in the combination phase. We stress that blocks multiplications can be performed using Paillier cryptosystem due to its additive homomorphic property and to the interactive multiplication.

5 Experimental Results

We present the experimental results for our SM3 and S2M3 protocols using the Hadoop [2] implementation of MapReduce. We have done all computations on a cluster running on Ubuntu Server 14.04 with Vanilla Hadoop 2.7.1 using Java 1.7.0. The cluster is composed of one master node and of six data nodes. The master node has six CPU cadenced to 2.4GHz, 160Gb of disk, and 32Gb of RAM. The six data nodes have of two CPU cadenced to 2.4GHz, 40Gb of disk, and 4Gb of RAM.

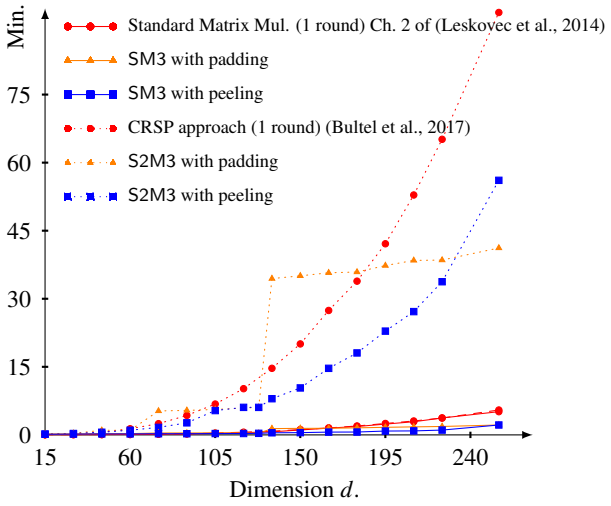


Figure 6: Running-time on SM3, S2M3 protocols with padding and peeling methods, and comparisons with protocols from Bultel et al. (Bultel et al., 2017).

For each $15 \le d \le 256$, we generate two random square matrices A and B of order d and composed of integers in $[0,100]$. For each order d, we perform matrix multiplication between A and B with SM3 and S2M3 protocols using the dynamic padding and the dynamic peeling methods. We also run the standard matrix multiplication with 1 round of MapReduce cf. Chapter 2 of (Leskovec et al., 2014) and the secure CRSP approach approach presented by (Bultel et al., 2017). We use Paillier cryptosystem with the public key $pk = (n, g)$ where n is 2048-bits long for our S2M3 protocol and the CRSP approach.

Scalability. We present in Fig. 6 the running time for our SM3 and S2M3 protocols using dynamic padding and dynamic peeling methods. We report average times over five runs. Without any security, S2M3 with padding and peeling methods perform the

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

matrix multiplication a bit faster than the standard matrix multiplication for the largest dimensions. For secure protocols, the curve of S2M3 with padding grows step by step, the running-time is very similar at the beginning and at the end of each step compared with other protocols. When the dimension d is of the form 2^k or 2^k-1 (with $k \in \mathbb{N}^\star$), the S2M3 with padding is better that other secure protocols. The curve of S2M3 with peeling grows gradually with dimension d. When $2^k \leq d < 2^k - 1$, S2M3 with peeling is the faster algorithm. Moreover, S2M3 with peeling is always faster than the secure approach of the standard matrix multiplication.

6 Security Proof

We provide a formal security proof of our protocol S2M3 in the standard model considering semi-honest adversaries.

6.1 Defining Security for Semi-Honest Adversaries

Secure Computation. We use the standard multi-party computations definition of security against semi-honest adversaries (Ma and Deng, 2008) to prove the security of our protocol S2M3. We consider several entities that run a secure protocol in order to evaluate a multivariate function g. For example, consider two parties E_1 and E_2 using respectively inputs I_{E_1} and I_{E_2} that run a secure two-party protocol to evaluate the multivariate function $g = (g_{E_1}, g_{E_2})$. At the end of the protocol, E_1 learns $g_{E_1}(I_{E_1}, I_{E_2})$ and E_2 learns $g_{E_2}(I_{E_1}, I_{E_2})$. Such a protocol is *secure* when E_1 (resp. E_2) learns nothing else than $g_{E_1}(I_{E_1}, I_{E_2})$ about I_{E_2} (resp. $g_{E_2}(I_{E_1}, I_{E_2})$ about I_{E_1}). We consider *semi-honest* adversaries in the sense that E_1 and E_2 run *honestly* the protocols, but they try to exploit all intermediate information that they have received during the protocol.

We model S2M3 with three entities: $\mathcal{O}$ for the data owner, $\mathcal{C}$ for the public cloud, and $\mathcal{U}$ for the user. We assume that these three entities do not collude. They use respective inputs $I = (I_\mathcal{O}, I_\mathcal{C}, I_\mathcal{U})$, and a function $g = (g_\mathcal{O}, g_\mathcal{C}, g_\mathcal{U})$ such that:

- $\mathcal{O}$ has the input $I_\mathcal{O} = (A, B, pk)$ where A and B are matrices of dimension $d = 2^k$ (with $k \in \mathbb{N}^\star$) to multiply and pk is a Paillier public key. It returns $g_\mathcal{O}(I) = \perp$ because $\mathcal{O}$ does not learn anything.
- $\mathcal{C}$ has the input $I_\mathcal{C} = pk$ where pk is a Paillier public key, and returns $g_\mathcal{C}(I) = d$ because $\mathcal{C}$ learns the dimension d of matrices A and B.
- $\mathcal{U}$ has the input $I_\mathcal{U} = (pk, sk)$ where (pk, sk) is a Paillier key pair, and returns $g_\mathcal{U}(I) = C$, where $C = A \times B$.

We start by formally defining the *computational indistinguishability* and the *view* of an entity before formally presenting the security of our protocol.

Definition 3 (Computational indistinguishability (Lindell, 2017))**.** *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 by $X_\eta \overset{c}{\equiv} Y_\eta$, if for every probabilistic polynomial-time algorithm $\mathcal{D}$ there exists a negligible function $\varepsilon(\cdot)$ such that:*

$$\left|\Pr[x \leftarrow X_\eta : 1 \leftarrow D(x)] - \Pr[y \leftarrow Y_\eta : 1 \leftarrow D(y)]\right| \leq \varepsilon(\eta) .$$

Definition 4 (The view)**.** *Let P be a ρ-parties protocol that computes the function $g = (g_i)_{1 \leq i \leq \rho}$ for the entities $(E_i)_{1 \leq i \leq \rho}$ using inputs $I = (I_i)_{1 \leq i \leq \rho}$. The view of a party E_i (where $1 \leq i \leq \rho$) during an execution of P, denoted $\mathsf{view}^P_{E_i}(I)$, is the set of all values sent and received by E_i during the execution of the protocol.*

To prove that a party E learns nothing during the execution of the protocol, we show that E can run a polynomial *simulator* algorithm that simulates the protocol, such that E 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 P be a ρ-parties protocol that computes the function $g = (g_i)_{1 \leq i \leq \rho}$ for entities $(E_i)_{1 \leq i \leq \rho}$ using inputs $I = (I_i)_{1 \leq i \leq \rho}$. We say that P* securely computes *g in the presence of semi-honest adversaries if for each E_i (where $1 \leq i \leq \rho$) there exists a probabilistic polynomial-time simulator $\mathcal{S}_{E_i}$ such that:*

$$\mathcal{S}_{E_i}(I_i, g_{E_i}(I)) \overset{c}{\equiv} \mathsf{view}^P_{E_i}(I) .$$

6.2 Proof

We prove the following theorem:

Theorem 1. *Assume Paillier cryptosystem is IND-CPA. Then, our protocol* S2M3 *securely computes the matrix multiplication in the presence of semi-honest adversaries.*

The security proof is decomposed in Lemma 1 for $\mathcal{O}$, Lemma 2 for $\mathcal{C}$, and Lemma 3 for $\mathcal{U}$.

Lemma 1. *There exists a probabilistic polynomial-time simulator $\mathcal{S}_\mathcal{O}$ such that for all $I = (I_\mathcal{O}, I_\mathcal{C}, I_\mathcal{U})$, we have $\mathcal{S}_\mathcal{O}(I_\mathcal{O}, g_\mathcal{O}(I)) \overset{c}{\equiv} \mathsf{view}^{\mathsf{S2M3}}_\mathcal{O}(I)$.*

Proof. Consider that the data owner O is corrupted. Observe that O knows the two matrices A and B, and the Paillier public key pk of the user. The view of O only contains the encryption of matrices A and B that are sent to $\mathcal{C}$. We build the simulator $\mathcal{S}_O$ as follows:

1. $\mathcal{S}_O$ creates two matrices A' and B' such that $A' = (\mathcal{E}_{pk}(a_{ij}))_{1\leq i,j\leq d}$ and $B' = (\mathcal{E}_{pk}(b_{ij}))_{1\leq i,j\leq d}$, where $a_{ij} \in A$ and $b_{ij} \in B$ for $1 \leq i, j \leq d$.
2. $\mathcal{S}_O$ returns $\mathsf{view} = (A', B')$.

We remark that $\mathcal{S}_O$ uses exactly the same algorithm as the real protocol of S2M3, then it describes exactly the same distribution as $\mathsf{view}_O^{\mathsf{S2M3}}(I)$, which concludes the proof. □

Lemma 2. *There exists a probabilistic polynomial-time simulator $\mathcal{S}_C$ such that for all $I = (I_O, I_C, I_{\mathcal{U}})$, we have $\mathcal{S}_C(I_C, g_C(I)) \overset{\mathsf{c}}{\equiv} \mathsf{view}_C^{\mathsf{S2M3}}(I)$.*

Proof. Consider that the public cloud $\mathcal{C}$ is corrupted. Observe that $\mathcal{C}$ knows the Paillier public key pk of the user and the dimension d of the two matrices. The view of $\mathcal{C}$ contains the two encrypted matrices sent by O, the encryption of the matrix $C = A \times B$ sent to $\mathcal{U}$, and all couples of ciphertexts $(x_i^{(\ell)}, y_i^{(\ell)})$ sent to $\mathcal{U}$ and all corresponding ciphertexts $z_i^{(\ell)}$ returned by $\mathcal{U}$ to compute multiplication on encrypted coefficients, for $1 \leq i \leq 7$ and $1 \leq \ell \leq 7^k$. Formally, $\mathcal{S}_C$ is given pk and n and works as follows:

1. $\mathcal{S}_C$ creates two square matrices of dimension $d = 2^k$ (with $k \in \mathbb{N}^\star$) $A = (a_{ij})_{1\leq i,j\leq d}$ and $B = (b_{ij})_{1\leq i,j\leq d}$ where a_{ij} and b_{ij} are randomly chosen in $\mathbb{Z}_n$, the plaintext space of Paillier cryptosystem.
2. $\mathcal{S}_C$ encrypts each element of A (resp. B) using the Paillier cryptosystem and the public key pk, and obtains encrypted matrix A' (resp. B'). We denote by a'_{ij} (resp. b'_{ij}) elements of A' (resp. B').
3. $\mathcal{S}_C$ parses A' and B' such that:

$$A' = \begin{bmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{bmatrix} \quad \text{and} \quad B' = \begin{bmatrix} B'_{11} & B'_{12} \\ B'_{21} & B'_{22} \end{bmatrix},$$

   where A'_{ij} and B'_{ij} for $1 \leq i, j \leq 2$ have the same dimension, i.e., $d/2 \times d/2$. It also computes the following submatrices as in S2M3:
   - $S_1 \leftarrow (a'_{ij} \cdot b'_{ij})_{1\leq i,j\leq d/2}$ s.t. $a'_{ij} \in A'_{21}, b'_{ij} \in A'_{22}$.
   - $S_2 \leftarrow (a'_{ij}/b'_{ij})_{1\leq i,j\leq d/2}$ s.t. $a'_{ij} \in S_1, b'_{ij} \in A'_{11}$.
   - $S_3 \leftarrow (a'_{ij}/b'_{ij})_{1\leq i,j\leq d/2}$ s.t. $a'_{ij} \in A'_{11}, b'_{ij} \in A'_{21}$.
   - $S_4 \leftarrow (a'_{ij}/b'_{ij})_{1\leq i,j\leq d/2}$ s.t. $a'_{ij} \in A'_{11}, b'_{ij} \in S_2$.
   - $T_1 \leftarrow (a'_{ij}/b'_{ij})_{1\leq i,j\leq d/2}$ s.t. $a'_{ij} \in B'_{12}, b'_{ij} \in B'_{11}$.
   - $T_2 \leftarrow (a'_{ij}/b'_{ij})_{1\leq i,j\leq d/2}$ s.t. $a'_{ij} \in B'_{22}, b'_{ij} \in T_1$.
   - $T_3 \leftarrow (a'_{ij}/b'_{ij})_{1\leq i,j\leq d/2}$ s.t. $a'_{ij} \in B'_{22}, b'_{ij} \in B'_{12}$.
   - $T_4 \leftarrow (a'_{ij}/b'_{ij})_{1\leq i,j\leq d/2}$ s.t. $a'_{ij} \in T_2, b'_{ij} \in B'_{21}$.
4. - If $\dim(A') \neq 2$, then $\mathcal{S}_C$ runs recursively step 3. for each matrices pair (A', B') of $\{(A'_{11}, B'_{11}), (A'_{12}, B'_{21}), (S_4, B'_2), (A'_{22}, T_4), (S_1, T_1), (S_2, T_2), (S_3, T_3)\}$.
   - Otherwise, i.e., if $\dim(A') = 2$, we denote by $(A'^{(\ell)}, B'^{(\ell)})$, for $1 \leq \ell \leq 7^{k-1}$, all couples of encrypted matrices of dimension 2×2 that $\mathcal{S}_C$ obtains. We denote by ${a'_{ij}}^{(\ell)}$ (resp. ${b'_{ij}}^{(\ell)}$) elements of matrix $A'^{(\ell)}$ (resp. $B'^{(\ell)}$) such that $1 \leq i, j \leq 2$. As in S2M3, $\mathcal{S}_C$ computes for $1 \leq i \leq 7$ and $1 \leq \ell \leq 7^{k-1}$:
     - $S_1^{(\ell)} = {a'_{21}}^{(\ell)} \cdot {a'_{22}}^{(\ell)}$,
     - $S_2^{(\ell)} = S_1^{(\ell)} \cdot {a'_{11}}^{(\ell)}$,
     - $S_3^{(\ell)} = {a'_{11}}^{(\ell)} \cdot {a'_{21}}^{(\ell)}$,
     - $S_4^{(\ell)} = {a'_{11}}^{(\ell)} \cdot S_2^{(\ell)}$,
     - $T_1^{(\ell)} = {b'_{12}}^{(\ell)} \cdot {b'_{11}}^{(\ell)}$,
     - $T_2^{(\ell)} = {b'_{22}}^{(\ell)} \cdot T_1^{(\ell)}$,
     - $T_3^{(\ell)} = {b'_{22}}^{(\ell)} \cdot {b'_{12}}^{(\ell)}$,
     - $T_4^{(\ell)} = T_2^{(\ell)} \cdot {b'_{21}}^{(\ell)}$.
5. $\mathcal{S}_C$ picks randomly $(r_i^{(\ell)}, s_i^{(\ell)}, t_i^{(\ell)}) \in (\mathbb{Z}_n)^3$ for $1 \leq i \leq 7$, $1 \leq \ell \leq 7^{k-1}$ to simulate the interactive homorphic encryption. It computes for $1 \leq i \leq 7$, and $1 \leq \ell \leq 7^{k-1}$:
   - $(x_1^{(\ell)}, y_1^{(\ell)}) = ({a'_{11}}^{(\ell)} \cdot \mathcal{E}_{pk}(r_1^{(\ell)}), {b'_{11}}^{(\ell)} \cdot \mathcal{E}_{pk}(s_1^{(\ell)}))$,
   - $(x_2^{(\ell)}, y_2^{(\ell)}) = ({a'_{12}}^{(\ell)} \cdot \mathcal{E}_{pk}(r_2^{(\ell)}), {b'_{21}}^{(\ell)} \cdot \mathcal{E}_{pk}(s_2^{(\ell)}))$,
   - $(x_3^{(\ell)}, y_3^{(\ell)}) = (S_4^{(\ell)} \cdot \mathcal{E}_{pk}(r_3^{(\ell)}), {b'_{22}}^{(\ell)} \cdot \mathcal{E}_{pk}(s_3^{(\ell)}))$,
   - $(x_4^{(\ell)}, y_4^{(\ell)}) = ({a'_{22}}^{(\ell)} \cdot \mathcal{E}_{pk}(r_4^{(\ell)}), T_4^{(\ell)} \cdot \mathcal{E}_{pk}(s_4^{(\ell)}))$,
   - $(x_5^{(\ell)}, y_5^{(\ell)}) = (S_1^{(\ell)} \cdot \mathcal{E}_{pk}(r_5^{(\ell)}), T_1^{(\ell)} \cdot \mathcal{E}_{pk}(s_5^{(\ell)}))$,
   - $(x_6^{(\ell)}, y_6^{(\ell)}) = (S_2^{(\ell)} \cdot \mathcal{E}_{pk}(r_6^{(\ell)}), T_2^{(\ell)} \cdot \mathcal{E}_{pk}(s_6^{(\ell)}))$,
   - $(x_7^{(\ell)}, y_7^{(\ell)}) = (S_3^{(\ell)} \cdot \mathcal{E}_{pk}(r_7^{(\ell)}), T_3^{(\ell)} \cdot \mathcal{E}_{pk}(s_7^{(\ell)}))$,
   - $z_i^{(\ell)} = \mathcal{E}_{pk}(t_i^{(\ell)})$.

   Couples of ciphertexts $(x_i^{(\ell)}, y_i^{(\ell)})$ correspond to ciphertexts sent by $\mathcal{C}$ to $\mathcal{U}$, and ciphertexts $z_i^{(\ell)}$ are the corresponding answers sent by $\mathcal{U}$ to $\mathcal{C}$. Using all $z_i^{(\ell)}$, $\mathcal{S}_C$ computes for $1 \leq i \leq 7$ and $1 \leq \ell \leq 7^{k-1}$:

$$w_i^{(\ell)} = z_i^{(\ell)} \Big/ \left(\left(x_1^{(\ell)^{r_1^{(\ell)}}}\right) \cdot \left(y_1^{(\ell)^{s_1^{(\ell)}}}\right) \cdot \mathcal{E}_{pk}\left(r_1^{(\ell)} \cdot s_1^{(\ell)}\right) \right)$$

   simulating the encryption of the plaintext product associated to $x_i^{(\ell)}$ and $y_i^{(\ell)}$. Then, it computes:

   - ${C'_1}^{(\ell)} = w_1^{(\ell)} \cdot w_2^{(\ell)}$,

- $C_2'^{(\ell)} = w_1^{(\ell)} \cdot w_6^{(\ell)}$,
- $C_3'^{(\ell)} = C_2'^{(\ell)} \cdot w_7^{(\ell)}$,
- $C_4'^{(\ell)} = C_2'^{(\ell)} \cdot w_5^{(\ell)}$,
- $C_5'^{(\ell)} = C_4'^{(\ell)} \cdot w_3^{(\ell)}$,
- $C_6'^{(\ell)} = C_3'^{(\ell)} / w_4^{(\ell)}$,
- $C_7'^{(\ell)} = C_3'^{(\ell)} \cdot w_5^{(\ell)}$,

and constructs the submatrix $C'^{(\ell)} = \begin{bmatrix} C_1'^{(\ell)} & C_5'^{(\ell)} \\ C_6'^{(\ell)} & C_7'^{(\ell)} \end{bmatrix}$.

6. $\mathcal{S}_C$ uses as S2M3 matrices $C'^{(\ell)}$ to construct recursively the matrix C', the encryption of $C = A \times B$.
7. Finally, $\mathcal{S}_C$ returns:
   $\mathsf{view} = (A', B', \{(x_i^{(\ell)}, y_i^{(\ell)}), z_i^{(\ell)}\}_{1 \le i \le 7, 1 \le \ell \le 7^{k-1}}, C')$.

Assume by contradiction there exists a polynomial-time distinguisher D such that for all $I \in I$:

$$\Big| \Pr[s \leftarrow \mathsf{view}_C^{\mathsf{S2M3}}(I) \colon 1 \leftarrow D(s)] - \Pr[s \leftarrow \mathcal{S}_C(I_C, g_C(I)) \colon 1 \leftarrow D(s)] \Big| = \lambda(\eta)\,,$$

where λ is a non-negligible function in η. We construct a probabilistic polynomial-time algorithm $\mathcal{A}$ that uses D to win with a non-negligible advantage the IND-CPA experiment. Algorithm $\mathcal{A}$ is given pk and works as follows:

1. $\mathcal{A}$ creates two square matrices of dimension $d = 2^k$ (with $k \in \mathbb{N}^\star$) $A = (a_{ij})_{1 \le i,j \le d}$ and $B = (b_{ij})_{1 \le i,j \le d}$ where a_{ij} and b_{ij} are randomly chosen in $\mathbb{Z}_n$, the plaintext space of Paillier cryptosystem.
2. $\mathcal{A}$ picks $(\alpha_{ij}, \beta_{ij}) \xleftarrow{\$} (\mathbb{Z}_n)^2$, and constructs $A' = (\mathcal{E}_{pk}(LR_b(a_{ij}, \alpha_{ij})))_{1 \le i,j \le d}$ and $B' = (\mathcal{E}_{pk}(LR_b(b_{ij}, \beta_{ij})))_{1 \le i,j \le d}$. We denote by a'_{ij} (resp. b'_{ij}) elements of A' (resp. B').
3. $\mathcal{A}$ parses A' and B' such that:

$$A' = \begin{bmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{bmatrix} \quad \text{and} \quad B' = \begin{bmatrix} B'_{11} & B'_{12} \\ B'_{21} & B'_{22} \end{bmatrix},$$

   where A'_{ij} and B'_{ij} for $1 \le i, j \le 2$ have the same dimension, i.e., $d/2 \times d/2$. It also computes the following submatrices as in S2M3:
   - $S_1 \leftarrow (a'_{ij} \cdot b'_{ij})_{1 \le i,j \le d/2}$ s.t. $a'_{ij} \in A'_{21}$, $b'_{ij} \in A'_{22}$.
   - $S_2 \leftarrow (a'_{ij} / b'_{ij})_{1 \le i,j \le d/2}$ s.t. $a'_{ij} \in S_1$, $b'_{ij} \in A'_{11}$.
   - $S_3 \leftarrow (a'_{ij} / b'_{ij})_{1 \le i,j \le d/2}$ s.t. $a'_{ij} \in A'_{11}$, $b'_{ij} \in A'_{21}$.
   - $S_4 \leftarrow (a'_{ij} / b'_{ij})_{1 \le i,j \le d/2}$ s.t. $a'_{ij} \in A'_{11}$, $b'_{ij} \in S_2$.
   - $T_1 \leftarrow (a'_{ij} / b'_{ij})_{1 \le i,j \le d/2}$ s.t. $a'_{ij} \in B'_{12}$, $b'_{ij} \in B'_{11}$.
   - $T_2 \leftarrow (a'_{ij} / b'_{ij})_{1 \le i,j \le d/2}$ s.t. $a'_{ij} \in B'_{22}$, $b'_{ij} \in T_1$.
   - $T_3 \leftarrow (a'_{ij} / b'_{ij})_{1 \le i,j \le d/2}$ s.t. $a'_{ij} \in B'_{22}$, $b'_{ij} \in B'_{12}$.
   - $T_4 \leftarrow (a'_{ij} / b'_{ij})_{1 \le i,j \le d/2}$ s.t. $a'_{ij} \in T_2$, $b'_{ij} \in B'_{21}$.
4. - If $\dim(A') \neq 2$, then $\mathcal{A}$ runs recursively step 3. for each $(A', B') \in \{(A'_{11}, B'_{11}), (A'_{12}, B'_{21}), (S_4, B'_2), (A'_{22}, T_4), (S_1, T_1), (S_2, T_2), (S_3, T_3)\}$.
   - Otherwise, i.e., if $\dim(A') = 2$, we denote by $(A'^{(\ell)}, B'^{(\ell)})$, for $1 \le \ell \le 7^{k-1}$, all couples of encrypted matrices of dimension 2×2 that $\mathcal{A}$ obtains. We denote by $a'^{(\ell)}_{ij}$ (resp. $b'^{(\ell)}_{ij}$) elements of matrix $A'^{(\ell)}$ (resp. $B'^{(\ell)}$) such that $1 \le i, j \le 2$. As in S2M3, $\mathcal{A}$ computes for $1 \le i \le 7$ and $1 \le \ell \le 7^{k-1}$:
     - $S_1^{(\ell)} = a'^{(\ell)}_{21} \cdot a'^{(\ell)}_{22}$,
     - $S_2^{(\ell)} = S_1^{(\ell)} \cdot a'^{(\ell)}_{11}$,
     - $S_3^{(\ell)} = a'^{(\ell)}_{11} \cdot a'^{(\ell)}_{21}$,
     - $S_4^{(\ell)} = a'^{(\ell)}_{11} \cdot S_2^{(\ell)}$,
     - $T_1^{(\ell)} = b'^{(\ell)}_{12} \cdot b'^{(\ell)}_{11}$,
     - $T_2^{(\ell)} = b'^{(\ell)}_{22} \cdot T_1^{(\ell)}$,
     - $T_3^{(\ell)} = b'^{(\ell)}_{22} \cdot b'^{(\ell)}_{12}$,
     - $T_4^{(\ell)} = T_2^{(\ell)} \cdot b'^{(\ell)}_{21}$.
5. $\mathcal{A}$ picks randomly $(r_i^{(\ell)}, s_i^{(\ell)}, t_i^{(\ell)}) \in (\mathbb{Z}_n)^3$ for $1 \le i \le 7$, $1 \le \ell \le 7^{k-1}$ to compute the interactive homorphic encryption. It also computes for $1 \le i \le 7$, and $1 \le \ell \le 7^{k-1}$:
   - $(x_1^{(\ell)}, y_1^{(\ell)}) = (a'^{(\ell)}_{11} \cdot \mathcal{E}_{pk}(r_1^{(\ell)}), b'^{(\ell)}_{11} \cdot \mathcal{E}_{pk}(s_1^{(\ell)}))$,
   - $(x_2^{(\ell)}, y_2^{(\ell)}) = (a'^{(\ell)}_{12} \cdot \mathcal{E}_{pk}(r_2^{(\ell)}), b'^{(\ell)}_{21} \cdot \mathcal{E}_{pk}(s_2^{(\ell)}))$,
   - $(x_3^{(\ell)}, y_3^{(\ell)}) = (S_4^{(\ell)} \cdot \mathcal{E}_{pk}(r_3^{(\ell)}), b'^{(\ell)}_{22} \cdot \mathcal{E}_{pk}(s_3^{(\ell)}))$,
   - $(x_4^{(\ell)}, y_4^{(\ell)}) = (a'^{(\ell)}_{22} \cdot \mathcal{E}_{pk}(r_4^{(\ell)}), T_4^{(\ell)} \cdot \mathcal{E}_{pk}(s_4^{(\ell)}))$,
   - $(x_5^{(\ell)}, y_5^{(\ell)}) = (S_1^{(\ell)} \cdot \mathcal{E}_{pk}(r_5^{(\ell)}), T_1^{(\ell)} \cdot \mathcal{E}_{pk}(s_5^{(\ell)}))$,
   - $(x_6^{(\ell)}, y_6^{(\ell)}) = (S_2^{(\ell)} \cdot \mathcal{E}_{pk}(r_6^{(\ell)}), T_2^{(\ell)} \cdot \mathcal{E}_{pk}(s_6^{(\ell)}))$,
   - $(x_7^{(\ell)}, y_7^{(\ell)}) = (S_3^{(\ell)} \cdot \mathcal{E}_{pk}(r_7^{(\ell)}), T_3^{(\ell)} \cdot \mathcal{E}_{pk}(s_7^{(\ell)}))$,
   - $z_1^{(\ell)} = \mathcal{E}_{pk}(LR_b((a_{11}^{(\ell)} + r_1^{(\ell)}) \cdot (b_{11}^{(\ell)} + s_1^{(\ell)}), t_1^{(\ell)}))$,
   - $z_2^{(\ell)} = \mathcal{E}_{pk}(LR_b((a_{12}^{(\ell)} + r_2^{(\ell)}) \cdot (b_{21}^{(\ell)} + s_2^{(\ell)}), t_2^{(\ell)}))$,
   - $z_3^{(\ell)} = \mathcal{E}_{pk}(LR_b((S_4^{(\ell)} + r_3^{(\ell)}) \cdot (b_{22}^{(\ell)} + s_3^{(\ell)}), t_3^{(\ell)}))$,
   - $z_4^{(\ell)} = \mathcal{E}_{pk}(LR_b((a_{22}^{(\ell)} + r_4^{(\ell)}) \cdot (T_4^{(\ell)} + s_4^{(\ell)}), t_4^{(\ell)}))$,
   - $z_5^{(\ell)} = \mathcal{E}_{pk}(LR_b((S_1^{(\ell)} + r_5^{(\ell)}) \cdot (T_1^{(\ell)} + s_5^{(\ell)}), t_5^{(\ell)}))$,
   - $z_6^{(\ell)} = \mathcal{E}_{pk}(LR_b((S_2^{(\ell)} + r_6^{(\ell)}) \cdot (T_2^{(\ell)} + s_6^{(\ell)}), t_6^{(\ell)}))$,
   - $z_7^{(\ell)} = \mathcal{E}_{pk}(LR_b((S_3^{(\ell)} + r_7^{(\ell)}) \cdot (T_3^{(\ell)} + s_7^{(\ell)}), t_7^{(\ell)}))$.

   Using all $z_i^{(\ell)}$, $\mathcal{A}$ computes for $1 \le i \le 7$ and $1 \le \ell \le 7^{k-1}$:

$$w_i^{(\ell)} = z_i^{(\ell)} \Big/ \left(\left(x_1^{(\ell)^{r_1^{(\ell)}}}\right) \cdot \left(y_1^{(\ell)^{s_1^{(\ell)}}}\right) \cdot \mathcal{E}_{pk}\left(r_1^{(\ell)} \cdot s_1^{(\ell)}\right) \right).$$

Then, it computes:

- $C_1'^{(\ell)} = w_1^{(\ell)} \cdot w_2^{(\ell)}$,
- $C_2'^{(\ell)} = w_1^{(\ell)} \cdot w_6^{(\ell)}$,
- $C_3'^{(\ell)} = C_2'^{(\ell)} \cdot w_7^{(\ell)}$,
- $C_4'^{(\ell)} = C_2'^{(\ell)} \cdot w_5^{(\ell)}$,
- $C_5'^{(\ell)} = C_4'^{(\ell)} \cdot w_3^{(\ell)}$,
- $C_6'^{(\ell)} = C_3'^{(\ell)} / w_4^{(\ell)}$,
- $C_7'^{(\ell)} = C_3'^{(\ell)} \cdot w_5^{(\ell)}$,

and constructs the submatrix $C'^{(\ell)} = \begin{bmatrix} C_1'^{(\ell)} & C_5'^{(\ell)} \\ C_6'^{(\ell)} & C_7'^{(\ell)} \end{bmatrix}$.

6. $\mathcal{A}$ uses as S2M3 matrices $C'^{(\ell)}$ to construct recursively the matrix C', the encryption of $C = A \times B$.
7. Finally, $\mathcal{A}$ sets: $\mathsf{view} = (A', B', \{(x_i^{(\ell)}, y_i^{(\ell)}), z_i^{(\ell)}\}_{1 \leq i \leq 7, 1 \leq \ell \leq 7^{k-1}}, C')$, runs $b_* \leftarrow D(\mathsf{view})$ and returns b_*.

We remark that:

$$\Pr[1 \leftarrow \mathsf{Exp}^{\text{ind-cpa-0}}_{\text{Paillier},\mathcal{A}}(\eta)] = \Pr[D(\mathsf{view}^{\mathsf{S2M3}}_{\mathcal{C}}(I)) = 1] .$$

Indeed, when $b = 0$ the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol S2M3. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the real protocol. On the other hand, we have:

$$\Pr[1 \leftarrow \mathsf{Exp}^{\text{ind-cpa-1}}_{\text{Paillier},\mathcal{A}}(\eta)] = \Pr[D(\mathcal{S}_{\mathcal{C}}(I_C, g_{I_C}(I))) = 1] .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathsf{S}_{\mathcal{C}}$. Then the probability that the experiment returns 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:

$$\begin{aligned} \mathsf{Adv}^{\text{ind-cpa}}_{\text{Paillier},\mathcal{A}}(\eta) &= \big|\Pr[1 \leftarrow \mathsf{Exp}^{\text{ind-cpa-1}}_{\text{Paillier},\mathcal{A}}(\eta)] \\ &\quad - \Pr[1 \leftarrow \mathsf{Exp}^{\text{ind-cpa-0}}_{\text{Paillier},\mathcal{A}}(\eta)]\big| \\ &= \big|\Pr[D(\mathsf{view}^{\mathsf{S2M3}}_{\mathcal{C}}(I)) = 1] \\ &\quad - \Pr[D(\mathcal{S}_{\mathcal{C}}(I_C, g_{I_C}(I))) = 1]\big| \\ &= \lambda(\eta) , \end{aligned}$$

which is non-negligible in η, and concludes the proof. □

Lemma 3. *There exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{U}}$ such that for all $I = (I_O, I_C, I_{\mathcal{U}})$, we have $\mathcal{S}_{\mathcal{U}}(I_{\mathcal{U}}, g_{\mathcal{U}}(I)) \overset{\mathsf{c}}{\equiv} \mathsf{view}^{\mathsf{S2M3}}_{\mathcal{U}}(I)$.*

Proof. Consider that the user $\mathcal{U}$ is corrupted. Observe that $\mathcal{U}$ knows the Paillier key pair (pk, sk) and the result of matrix multiplication $C = A \times B$. The view of $\mathcal{U}$ contains the couple of ciphertexts (x_i, y_i) sent by $\mathcal{C}$ and the corresponding answer z_i for $1 \leq i \leq 7^k$. The view of $\mathcal{U}$ also contains $C' = \mathcal{E}_{pk}(C)$ sent by $\mathcal{C}$. We build the simulator $\mathcal{S}_{\mathcal{U}}$ as follows:

1. $\mathcal{S}_{\mathcal{U}}$ picks randomly $(r_i, s_i) \in (\mathbb{Z}_n)^2$ and computes $x_i = \mathcal{E}_{pk}(r_i)$, $y_i = \mathcal{E}_{pk}(s_i)$, and $z_i = \mathcal{E}_{pk}(r_i \cdot s_i)$ for $1 \leq i \leq 7^k$.
2. $\mathcal{S}_{\mathcal{U}}$ computes $C' = (\mathcal{E}_{pk}(c_{ij}))_{1 \leq i,j \leq d}$ where $c_{ij} \in C$ for $1 \leq i, j \leq d$.
3. $\mathcal{S}_{\mathcal{U}}$ returns $\mathsf{view} = (\{(x_i, y_i), z_i\}_{1 \leq i \leq 7^k}, C')$.

We remark that $\mathcal{S}_{\mathcal{U}}$ describes exactly the same distribution as $\mathsf{view}^{\mathsf{S2M3}}_{\mathcal{U}}(I)$, which concludes the proof. □

7 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 *Data Mining and Analytics 2007, Proceedings of the Sixth 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 - 12th International Symposium, PETS*.

Bultel, X., Ciucanu, R., Giraud, M., and Lafourcade, P. (2017). Secure matrix multiplication with mapreduce. In *Proceedings of the 12th International Conference on Availability, Reliability and Security*.

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

Ciucanu, R., Giraud, M., Lafourcade, P., and Ye, L. (2018). Secure Grouping and Aggregation with MapReduce. In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications, ICETE 2018 - Volume 2: SECRYPT*.

Cramer, R., Damgård, I., and Nielsen, J. B. (2001). Multiparty Computation from Threshold Homomorphic Encryption. In *Advances in Cryptology - EUROCRYPT 2001, 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 *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01. Springer-Verlag.

Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *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 *17th Annual Computer Security Applications Conference ACSAC*.

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 *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems.*

Le Gall, F. (2014). Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14. ACM.

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

Lindell, Y. (2017). How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography.*

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

Mayberry, T., Blass, E., and Chan, A. H. (2013). PIRMAP: efficient private information retrieval for mapreduce. In *Financial Cryptography and Data Security - 17th International Conference, FC*.

Naccache, D. and Stern, J. (1998). A new public key cryptosystem based on higher residues. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, CCS '98, New York, NY, USA. ACM.

Okamoto, T. and Uchiyama, S. (1998). A new public-key cryptosystem as secure as factoring. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques*, pages 308–318.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*.

Sako, K. (2011). Goldwasser-micali encryption scheme. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, page 516.

Shoshan, A. and Zwick, U. (1999). All pairs shortest paths in undirected graphs with integer weights. In *FOCS*.

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