Secure and Efficient 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
`radu.ciucanu@insa-cvl.fr`
[2] Université Clermont Auvergne, LIMOS CNRS UMR 6158, Aubière, France
{`matthieu.giraud, pascal.lafourcade`}`@uca.fr`
[3] Harbin Institute of Technology, China
`16s003041@stu.hit.edu.cn`

Abstract. MapReduce is one of the most popular distributed programming paradigms that allows processing big data sets in parallel on a cluster. MapReduce users often outsource data and computations to a public cloud, which yields inherent security concerns. In this paper, we consider the problem of matrix multiplication and one of the most efficient matrix multiplication algorithms: the Strassen-Winograd (SW) algorithm. Our first contribution is a distributed MapReduce algorithm based on SW. Then, we tackle the security concerns that occur when outsourcing matrix multiplication computation to a honest-but-curious cloud i.e., that executes tasks dutifully, but tries to learn as much information as possible. Our main contribution is a secure distributed MapReduce algorithm called S2M3 (**S**ecure **S**trassen-Winograd **M**atrix **M**ultiplication with **M**apReduce) that enjoys security guarantees such as: none of the cloud nodes can learn the input or the output data. We formally prove the security properties of S2M3 and we present an empirical evaluation devoted to show its efficiency.

Keywords: Cloud security, Privacy-preserving cloud computations, MapReduce, Matrix multiplication, Strassen-Winograd algorithm.

1 Introduction

Matrix multiplication is a mathematical tool useful for solving various problems spanning over a plethora of domains e.g., statistical analysis, medicine, image processing, machine learning or web ranking. Indeed, Markov chains applications on genetics and sociology [6], or applications such that computation of shortest paths [28,32], convolutional neural network [21] deal with data processed as matrix multiplication. In such applications, the size of the matrices to be multiplied is often very large. The matrix multiplication is also the original purpose for which the Google implementation of MapReduce was created. Indeed, such multiplications are needed by Google in the computation of the PageRank algorithm [11]. Whereas a naive matrix multiplication algorithm has cubic complexity, many research efforts have been made to propose more efficient algorithms.

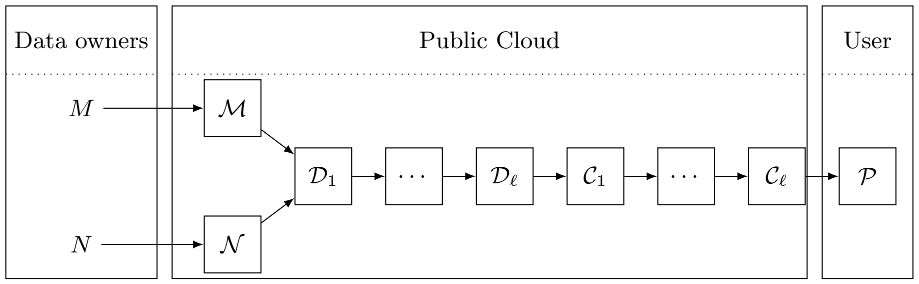

Fig. 1. Architecture of SW matrix multiplication with MapReduce.

One of the most efficient algorithms is Strassen-Winograd [29,25] (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 [17] 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 tackle the problem of distributing the SW algorithm using the MapReduce paradigm and we address the inherent security concerns that occur when outsourcing data and computations to a public cloud. Indeed, many cloud providers offer an important amount of data storage and computation power at a reasonable price e.g., Google Cloud Platform, Amazon Web Services, Microsoft Azure. Despite these benefits, cloud providers do not usually address the fundamental problem of protecting the security of the data, which in our case consists of input and output matrices. The outsourced data can be communicated over some network and processed on some machines where malicious cloud admins could learn and leak sensitive data.

Problem Statement. Two distinct data owners respectively hold compatible square matrices M and N of dimension $d \in \mathbb{N}^*$. A user (who does not know the matrices M and N) wants their product $P := MN$. Matrices M and N are sent to the distributed file system of some public cloud provider. We assume that the matrices M (resp. N) is initially spread over a set $\mathcal{M}$ (resp. $\mathcal{N}$) of nodes of the public cloud storing a chunk of M (resp. N), i.e., a set of elements of M (resp. N). The final result P is computed over sets of nodes $\mathcal{D}_1, \ldots, \mathcal{D}_\ell, \mathcal{C}_1, \ldots, \mathcal{C}_\ell$ before it is sent to the user's nodes $\mathcal{P}$, where ℓ depends on the dimension d. Moreover, we assume that data owners (resp. the user) cannot collude with the public cloud and the user (resp. the public cloud and data owners). We illustrate the architecture of SW matrix multiplication with MapReduce in Fig. 1.

We expect the following security properties:

1. the user cannot learn any information about input matrices M and N,
2. cloud nodes cannot learn any information about matrices M, N, and P.

Related Work. Chapter 2 of [22] presents an introduction to the MapReduce paradigm. The security and privacy concerns of MapReduce have been summarized in a recent survey [12]. The state-of-the-art techniques for secure execution of MapReduce computations focus on problems such as word search [3], information retrieval [26], grouping and aggregation [7], equijoins [13,5], set intersection [8], and matrix multiplication [4]. The goal of these works is similar to ours i.e., execute MapReduce computations such that the public cloud cannot learn any information on the input or output data.

In this paper, we focus on matrix multiplication. Recently, [4] secured the two standard MapReduce algorithms for matrix multiplication using one and two MapReduce rounds cf. Chapter 2 of [22]. For each algorithm, they proposed two approaches: SP (*Secure-Private*) and CRSP (*Collision-Resistance Secure-Private*). The two approaches are based on the Paillier partially homomorphic cryptosystem [27]. Contrary to the CRSP approach, the SP approach assumes that cloud nodes do not collude. In this paper, we assume that all cloud nodes can collude, hence our secure protocol S2M3 can be considered as a CRSP approach. We show that MapReduce matrix multiplication can be done faster using the SW algorithm, compared to the standard MapReduce matrix multiplication for both no-secure and secure approaches.

Distributed matrix multiplication has been thoroughly investigated in the secure multi-party computation model (MPC) [14,15,1,30], 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 [11] 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 [24,10] 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 a secure and efficient MapReduce protocol based on the SW algorithm.

Summary of Contributions and Paper Organization. This paper is an extension of our previous paper [9], which is to the best of our knowledge the first one to propose a secure MapReduce-based protocol based on the SW algorithm. We next discuss the organization and contributions of this paper, while highlighting the differences with respect to [9].

- In Sect. 2, we introduce preliminary notions: crypographic tools and the basic SW algorithm.

  The cryptographic tools presented here go beyond [9] because we need to introduce more notions to be able in Sect. 6 to formally prove our security results that are not part of [9].

- In Sect. 3, we present our first contribution, that is a MapReduce version of the SW matrix multiplication algorithm. We call our protocol SM3 (**S**trassen-**W**inograd Matrix Multiplication with **M**apReduce).

  In our initial version of the paper [9], we present only the basic case where the matrix dimension is a power of 2 and we briefly discuss the intuition behind generalizing the algorithm to arbitrary dimensions. On the other hand, we provide here pseudocode and we discuss in detail two different flavors (padding and peeling) for generalizing the basic algorithm to arbitrary dimensions.
- In Sect. 4, we present our secure protocol for SW matrix multiplication with MapReduce, which is the main contribution of the paper. Our secure protocol S2M3 (**S**ecure **S**trassen-Winograd **M**atrix **M**ultiplication with **M**apReduce) relies on the MapReduce paradigm and on Paillier's public-key cryptosystem. The public 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 public cloud cannot learn none of the input or output matrices.

  Similarly to the previous point, in this extended version, we also include secure algorithms for multiplying matrices whose dimension is not necessarily a power of 2. We achieve this generalization by using padding and peeling techniques, which were not part of our first version of the paper [9].
- In Sect. 5, we present an experimental evaluation of our two MapReduce protocols (SM3 and S2M3) using Hadoop [16], the Apache MapReduce implementation.
- In Sect. 6, we formally prove, using a standard security model, that the S2M3 protocol satisfies the security properties from the problem statement, which hold even in the presence of collusions among the cloud's nodes.

  All theorems and lemmas from this paper are new with respect to the first version of our paper [9].
- In Sect. 7, we discuss conclusions and future work.

To sum up, the main novel content of this paper w.r.t. [9] consists of new non-trivial algorithms and proofs, which significantly add more than 30% of new material (the volume of the paper actually increased from 8 to 25 pages). Moreover, as required in the instructions, we have enriched the paper structure, and avoided as much as possible the verbatim repetitions e.g., we have changed the title, abstract, introduction, as well as the presentation of the subset of material that is common to [9]. Moreover, among the 25 figures in this paper, only the 2 from Sect. 5 are common to [9].

2 Preliminaries

We present cryptographic tools in Sect. 2.1 and the SW algorithm in Sect. 2.2

2.1 Cryptographic Tools

Notations. We define some notations that we use throughout the paper.

$a\|\|b$	Concatenation of two strings a and b
$x \xleftarrow{\$} E$	Uniformly random choice of a value x from the set E
$\mathrm{lcm}(a, b)$	Least common multiple of two integers a and b
$a := b + c$	Set the result of $b + c$ into a
$S^{a\times b}$	Matrix of a rows and b columns with elements in S
$A \times B$	Cartesian product between A and B
ε	Empty string

Negligible Function. A function $\mu : \mathbb{N} \to \mathbb{R}^+$ is called *negligible* if for every positive polynomial $p(\cdot)$ there exists $N_p \in \mathbb{N}$ such that for all integers $x > N_p$, we have $\mu(x) < 1/p(x)$.

Security Parameter and Adversaries. In order to formalize security notions, we need to bound the computing power of an adversary. Indeed, an arbitrary adversary can always break cryptosystems using a large enough computer and spending an exponential amount of time. We first define a *polynomial-time* algorithm: Let $\lambda \in \mathbb{N}$. An algorithm $\mathcal{A}$ is said to run in *polynomial-time* if there exists a polynomial $p(\cdot)$ such that for every input $x \in \{0,1\}^\lambda$, the execution time of $\mathcal{A}(x)$ is bounded by $p(\lambda)$ steps.

In cryptography, we restrict cryptosystems protection against a *reasonable* adversary represented by an algorithm $\mathcal{A}$. To do so, we use the notion of *security parameter*, denoted $\lambda \in \mathbb{N}$. The security parameter is passed as input to the adversary, under its unary form and indicates that the running time of the adversary is polynomial in λ and whose computation success probability is non-negligible in λ, i.e., significantly high.

Moreover, when a polynomial-time algorithm $\mathcal{A}$ is allowed to "throw coins", we said that $\mathcal{A}$ is a *probabilistic polynomial-time* algorithm. In the following, $\textsc{ppt}(\lambda)$ denotes the set of probabilistic algorithms that are bounded in the security parameter λ.

Experiments. Security property of a cryptosystem can be proven using an *experiment* (or game). We call an experiment, an algorithm that proposes some *challenge* to an adversary (i.e., a probabilistic polynomial-time algorithm). The challenge can be considered as an algorithmic problem that the adversary tries to solve. If the adversary successfully resolves the challenge, we say that the adversary *wins the experiment.*

In order to have concrete adversary model, the adversary may have access to black-box algorithms (sometimes with some restrictions), called *oracles.* An oracle allows the adversary to learn some information that she cannot obtain by herself in order to solve the experiment. For instance, an oracle can be an algorithm that decrypts some ciphertexts using a key that the adversary does not know. We denote by $\mathcal{A}^{\mathsf{Oracle}}$ to mean that the adversary $\mathcal{A}$ has access to the oracle denoted Oracle.

When there is no such an adversary that wins the experiment with a non-negligible probability in polynomial-time, then we say that the cryptosystem is secure according to the considered property.

Computational Indistinguishability [23]. We first recall that a *distribution ensemble* is a sequence of random variables indexed by a countable set. In the context of secure computation, this sequence of random variables are indexed by $I \in \mathcal{I}$ where $\mathcal{I}$ is the set of all inputs of parties, and by the security parameter $\lambda \in \mathbb{N}$, i.e., $X_0 := \{X(I, \lambda)\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}}$.

Let $X_0 := \{X(I, \lambda)\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}}$ and $Y_0 := \{Y(I, \lambda)\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}}$ be two distribution ensembles. We say that X_0 and Y_0 are *computationally indistinguishable* if for every probabilistic polynomial algorithm D outputting a single bit, there exists a negligible function $\mu(\cdot)$ such that for every $I \in \mathcal{I}$ and every $\lambda \in \mathbb{N}$, we have

$$\big| \Pr[D(X(I, \lambda)) = 1] - \Pr[D(Y(I, \lambda) = 1] \big| \leq \mu(\lambda) \ .$$

We call the algorithm D a *distinguisher*, and we denote by $X_0 \overset{c}{\equiv} Y_0$ two computationally indistinguishable distribution ensembles.

Simulation-based Proofs. The security proofs from this paper follow the ideal/real simulation paradigm [19]. In other terms, proofs are based on the indistinguishability of two different distribution ensembles X^0 and X^1. However, in many cases it is infeasible to directly prove this indistinguishability. Instead, we use the *hybrid argument* consisting in the construction of *simulators* that generate a sequence of distributions ensembles, starting with X^0, and ending with X^1. Then, we prove that consecutive distribution ensembles are indistinguishable. The indistinguishability between X^0 and X^1 is therefore obtained by transitivity.

Secure Multiparty Computation. Some cryptographic protocols involve several participants, called *parties*, in order to jointly compute a function over their inputs while keeping those inputs private. This model is called *multiparty computation* [31]. Unlike traditional cryptosystems where the adversary is outside of the system and tries, for instance, to break the confidentiality or the integrity of communication, the adversary in this model controls one of the parties. We consider *semi-honest* (or *honest-but-curious*) adversaries [18]. Such an adversary controls one of the parties and follows the protocol specification exactly. However, it may try to learn more information than allowed by looking at the transcript of message that it received and its internal state. A protocol that is secure in the presence of semi-honest adversaries does guarantee that there is no *inadvertent leakage* of information.

Intuitively, a multiparty protocol is secure if whatever can be computed by a party participating in the protocol can be computed based on its input and output *only*. This idea is formalized according to the simulation paradigm by requiring the existence of a *simulator* who generates the *view* of a party, i.e., all values received, computed, and sent by this party during an execution of the protocol. More formally, the view is defined as follows: Let $\lambda \in \mathbb{N}$ be a security parameter, and π be a n-party protocol. The view of the party P_i, for all $i \in \llbracket 1, n \rrbracket$, during an execution of π on $I = (I_i)_{i \in \llbracket 1,n \rrbracket}$ is denoted $\mathsf{view}^{\pi}_{P_i}(I, \lambda)$ and equals (I_i, M_i, O_i), where I_i is the input of P_i, M_i represents messages sent by other parties and received by P_i, and O_i is the output of P_i computed from I_i and M_i during the protocol execution. We denote by $\mathsf{view}^{\pi}_{P_i,P_j}(I, \lambda) = (\mathsf{view}^{\pi}_{P_i}(I, \lambda), \mathsf{view}^{\pi}_{P_j}(I, \lambda))$, with $i, j \in \llbracket 1, n \rrbracket$, the joint view of a collusion between parties P_i and P_j.

Since the parties have input and output, the simulator must be given a party's input and output in order to generate its view. Thus, the security is formalized by saying that there exists a simulator that simulates a party's view in a protocol execution given its *input* and *output*. The formalization implies that a party cannot extract any information from her view during the protocol *execution* beyond what they can derive from their input and prescribed output.

We now formally define the security of a multiparty protocol with respect to static semi-honest adversaries [23]: Let π be a n-party protocol that computes the function $f = (f_i)_{i \in \llbracket 1,n \rrbracket}$ for parties $(P_i)_{i \in \llbracket 1,n \rrbracket}$ using inputs $I = (I_i)_{i \in \llbracket 1,n \rrbracket} \in \mathcal{I}$ and security parameter $\lambda \in \mathbb{N}$. We say that π *securely computes f in the presence of static semi-honest adversaries* if there exists, for each party P_i with $i \in \llbracket 1, n \rrbracket$, a probabilistic polynomial-time simulator $\mathcal{S}_i$ such that $\mathcal{S}_{P_i}(1^\lambda, I_i, f_i(I)) \overset{\mathrm{c}}{\equiv} \mathsf{view}^{\pi}_{P_i}(I, \lambda)$. We say that π is *secure against collusions* between parties P_i and P_j with $i, j \in \llbracket 1, n \rrbracket$, if there exists probabilistic polynomial-time simulators $\mathcal{S}_{P_i,P_j}$ such that $\mathcal{S}_{P_i,P_j}((1^\lambda, I_i, f_i(I)), (1^\lambda, I_j, f_j(I))) \overset{\mathrm{c}}{\equiv} \mathsf{view}^{\pi}_{P_i,P_j}(I, \lambda)$.

Asymmetric Encryption. Let $\lambda \in \mathbb{N}$ be a security parameter. An *asymmetric encryption* scheme is a triple of polynomial-time algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ such that

- $\mathcal{G}(1^\lambda)$ is a probabilistic algorithm that takes as input the security parameter λ, and outputs a private key sk from the secret key space $\mathcal{K}_s$, a public key pk from the public key space $\mathcal{K}_p$, a plaintext space $\mathcal{M}$, and a ciphertext message $\mathcal{C}$.
- $\mathcal{E}(pk, m)$ is a deterministic or probabilistic algorithm that takes as input a public key $pk \in \mathcal{K}_p$ and a plaintext $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$.
- $\mathcal{D}(sk, c)$ is a deterministic algorithm that takes as input a secret key $sk \in \mathcal{K}_s$ and a ciphertext $c \in \mathcal{C}$, and outputs either a plaintext $m \in \mathcal{M}$ or a special reject value (distinct from all messages).

In the following, we only consider *correct* asymmetric encryption schemes, that is schemes such that, for any $\lambda \in \mathbb{N}$, we have

$$\Pr\left[(sk, pk, \mathcal{M}, \mathcal{C}) := \mathcal{G}(1^\lambda); m \xleftarrow{\$} \mathcal{M}; m' := \mathcal{D}(sk, m) \colon m = m'\right] = 1 .$$

Indistinguishability Under Chosen Plaintext Attack. The *indistinguishability under chosen plaintext attack* is a fundamental security property for asymmetric encryption schemes. Intuitively, this security notion implies that two different plaintexts can be distinguished by their respective ciphertext. In other terms, a ciphertext leaks no information about its corresponding plaintext.

Consider an adversary that chooses a couple of plaintexts (m_0, m_1), and that receives the encryption of one of the two plaintexts. If such an adversary is not able to guess the chosen message with a non-negligible probability, then the asymmetric encryption scheme achieves the indistinguishability security under chosen plaintext attack.

The basic definition of indistinguishability under chosen plaintext attack allows one adversary to submit a couple of plaintexts only one time. However, it is know that the indistinguishability security under chosen plaintext attack is equivalent to the indistinguishability security under *multiple chosen plaintexts* attack in a multi-user setting [2]. That means the adversary can receive several public keys and choose several couples of plaintexts (m_0, m_1). For each of these couples, the adversary receives the encryption of m_b for the different public keys, where the same $b \in \{0, 1\}$ is used each time.

More precisely, we use the Left-Or-Right definition [2]. Let $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme, $\mathcal{A} \in \text{PPT}(\lambda)$, and $(\alpha, \beta) \in \mathbb{N}^2$. For all $i \in [\![1, \beta]\!]$, the oracle $\mathcal{E}(pk_i, (\mathsf{LoR}_b(\cdot,\cdot), \alpha))$ takes as input a couple of plaintexts (m_0, m_1), and returns $\mathcal{E}(pk_i, m_b)$. Moreover, this oracle cannot be called more than α times.

We define the (α, β)-indistinguishability under chosen plaintext attack (IND-CPA) experiment, denoted $\mathsf{Exp}^{\text{indcpa-}b_{\alpha,\beta}}_{\Pi,\mathcal{A}}$ for the adversary $\mathcal{A}$ against Π in Fig. 2.

```
Experiment: $\mathsf{Exp}^{\text{indcpa-}b_{\alpha,\beta}}_{\Pi,\mathcal{A}}(\lambda)$

foreach $i \in [\![1, \beta]\!]$ do
    $(sk_i, pk_i, \mathcal{M}_i, \mathcal{C}_i) := \mathcal{G}(\lambda)$
$b_* := \mathcal{A}^{\mathcal{E}(pk_1,(\mathsf{LoR}_b(\cdot,\cdot),\alpha)),\ldots,\mathcal{E}(pk_\beta,(\mathsf{LoR}_b(\cdot,\cdot),\alpha))}(\lambda)$
return $b_*$
```

Fig. 2. IND-CPA experiment.

We define the advantage of the adversary $\mathcal{A}$ with respect to Π as follows

$$\mathsf{Adv}^{\text{indcpa}_{\alpha,\beta}}_{\Pi,\mathcal{A}}(\lambda) := \left| \Pr\left[\mathsf{Exp}^{\text{indcpa-}1_{\alpha,\beta}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{indcpa-}0_{\alpha,\beta}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] \right| .$$

Indistinguishability under multiple chosen plaintexts attack [2] Let $\lambda \in \mathbb{N}$ be a security parameter. An *asymmetric encryption scheme Π* achieves the (α, β)-indistinguishability security under multiple chosen plaintexts attack, if there exists a negligible function $\mu(\cdot)$ such that

$$\max_{\mathcal{A} \in \text{PPT}(\lambda)} \left\{ \mathsf{Adv}^{\text{indcpa-}b_{\alpha,\beta}}_{\Pi,\mathcal{A}}(\lambda) \right\} \leq \mu(\lambda) .$$

In the sequel, we denote by $\mathsf{Exp}^{\text{indcpa-}b}_{\Pi,\mathcal{A}}$ the IND-CPA experiment.

Paillier's cryptosystem. Paillier's cryptosystem is an asymmetric encryption scheme. It is well known due to its homomorphic properties described next. Let $\lambda \in \mathbb{N}$ be a security parameter. The Paillier cryptosystem is an asymmetric encryption scheme defined by a triple of polynomial-time algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ such that:

- $\mathcal{G}(1^\lambda)$ generates two prime numbers p and q according to the security parameter λ, sets $n := p \cdot q$ and $\Lambda := \mathrm{lcm}(p-1, q-1)$, generates the group $(\mathbb{Z}^*_{n^2}, \cdot)$, randomly picks $g \in \mathbb{Z}^*_{n^2}$ such that $M := (L(g^\Lambda \mod n^2))^{-1} \mod n$ exists, with $L(x) := (x-1)/n$. It sets $sk := (\Lambda, M)$, $pk := (n, g)$, $\mathcal{M} := \mathbb{Z}_n$, and $\mathcal{C} := \mathbb{Z}^*_{n^2}$. Finally, it outputs $((sk, pk), \mathcal{M}, \mathcal{C})$.
- $\mathcal{E}(pk, m)$ randomly picks $r \in \mathbb{Z}^*_n$, computes $c := g^m \cdot r^n \mod n^2$, and outputs c.
- $\mathcal{D}(sk, c)$ computes $m := L(c^\Lambda \mod n^2) \cdot M \mod n$, and outputs m.

Paillier's cryptosystem achieves the indistinguishability security against chosen plaintext attack under the *DCR* assumption [27].

Next, we present the homomorphic properties of Paillier's cryptosystem.

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. Indeed, we have:

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

We also remark that $\mathcal{E}(pk, m_1) \cdot \mathcal{E}(pk, m_2)^{-1} = \mathcal{E}(pk, m_1 - m_2)$.

Specific Homomorphic Multiplication of Plaintexts. Let m_1 and m_2 be two plaintexts in $\mathbb{Z}_n$ and $c_1 \in \mathbb{Z}^*_{n^2}$ be the ciphertext of m_1 with the public key pk, i.e., $c_1 := \mathcal{E}(pk, m_1)$. With Paillier's 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. Indeed, we have:

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

Interactive Homomorphic Multiplication of Ciphertexts. Cramer et al. [10] show that a two-party protocol makes possible to perform multiplication over ciphertexts using additive homomorphic encryption schemes as Paillier's cryptosystem. More precisely, P_1 knows two ciphertexts $c_1, c_2 \in \mathbb{Z}^*_{n^2}$ of the plaintexts $m_1, m_2 \in \mathbb{Z}_n$ encrypted using the public key pk of P_2, she wants to obtain the ciphertext corresponding to $m_1 \cdot m_2$ without revealing to P_2 the plaintexts m_1 and m_2. In order to do that, P_1 has to interact with P_2 as described in Fig. 3. First, P_1 randomly picks $\delta_1, \delta_2 \in \mathbb{Z}_n$ and sends to Alice

$$\begin{array}{rcl}
P_2 & & P_1 \\
 & & c_1 := \mathcal{E}(pk, m_1) \\
 & & c_2 := \mathcal{E}(pk, m_2) \\
 & & \delta_1, \delta_2 \xleftarrow{\$} \mathbb{Z}_n \\
 & & \alpha_1 := c_1 \cdot \mathcal{E}(pk, \delta_1) \\
\mathcal{D}(sk, \alpha_1) = m_1 + \delta_1 \mod n & \xleftarrow{\alpha_1, \alpha_2} & \alpha_2 := c_2 \cdot \mathcal{E}(pk, \delta_2) \\
\mathcal{D}(sk, \alpha_2) = m_2 + \delta_2 \mod n & & \\
c := \mathcal{E}(pk, (m_1 + \delta_1) \cdot (m_2 + \delta_2) \mod n) & \xrightarrow{c} & \mathcal{E}(pk, m_1 \cdot m_2 \mod n)
\end{array}$$

Fig. 3. Paillier interactive multiplicative homomorphic protocol [10].

$\alpha_1 := c_1 \cdot \mathcal{E}(pk, \delta_1)$ and $\alpha_2 := c_2 \cdot \mathcal{E}(pk, \delta_2)$. By decrypting respectively α_1 and α_2, P_2 recovers respectively $m_1 + \delta_1 \mod n$ and $m_2 + \delta_2 \mod n$. She sends to P_1 $c := \mathcal{E}(pk, (m_1 + \delta_1) \cdot (m_2 + \delta_2) \mod n)$. Then, P_1 can deduce the value of $\mathcal{E}(pk, m_1 \cdot m_2 \mod n)$ by computing $c \cdot (\mathcal{E}(pk, \delta_1 \cdot \delta_2 \mod n) \cdot c_1^{\delta_1} \cdot c_2^{\delta_2})^{-1}$.

Indeed, $\mathcal{E}(pk, (m_1+\delta_1)\cdot(m_2+\delta_2) \mod n) = \mathcal{E}(pk, m_1 \cdot m_2 \mod n) \cdot \mathcal{E}(pk, m_1 \cdot \delta_2 \mod n) \cdot \mathcal{E}(pk, m_2 \cdot \delta_1 \mod n) \cdot \mathcal{E}(pk, \delta_1 \cdot \delta_2 \mod n)$.

2.2 Strassen-Winograd Algorithm

Let M and N two compatible matrices such that $M \in \mathbb{R}^{a \times b}$ and $N \in \mathbb{R}^{b \times c}$ with $(a, b, c) \in (\mathbb{N}^*)^3$. We denote by $m_{i,j}$ the element of the matrix M which is in the i-th row and the j-th column with $i \in [\![1, a]\!]$ and $j \in [\![1, b]\!]$. In the same way, we denote by $n_{j,k}$ the element of the matrix N which is in the j-th row and k-th column with $j \in [\![1, b]\!]$ and $k \in [\![1, c]\!]$. Moreover, we denote by P the product MN, and by $p_{i,k}$ the element of the matrix P which is in the i-th row and the k-th column with $i \in [\![1, a]\!]$ and $k \in [\![1, c]\!]$.

Strassen-Winograd Algorithm for 2-Power Size Matrices. The Strassen-Winograd matrix multiplication algorithm is denoted SW. It works with two square matrices of same dimension. We assume that $M, N \in \mathbb{R}^{d \times d}$ where $d := 2^k$ and $k \in \mathbb{N}^*$.

First, the SW algorithm splits matrices M and N into four quadrants of equal dimension such that

$$M := \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \quad \text{and} \quad N := \begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix}.$$

Using these 8 quadrants, SW performs the computation presented below.

- 8 additions

$$\begin{array}{ll}
S_1 := M_{21} + M_{22}\ , & T_1 := N_{12} - N_{11}\ , \\
S_2 := S_1 - M_{11}\ , & T_2 := N_{22} - T_1\ , \\
S_3 := M_{11} - M_{21}\ , & T_3 := N_{22} - N_{12}\ , \\
S_4 := M_{12} - S_2\ , & T_4 := T_2 - N_{21}\ .
\end{array}$$

– 7 recursive SW matrix multiplications

$$\begin{array}{ll} R_1 := M_{11}N_{11} \ , & R_5 := S_1T_1 \ , \\ R_2 := M_{12}N_{21} \ , & R_6 := S_2T_2 \ , \\ R_3 := S_4N_{22} \ , & R_7 := S_3T_3 \ . \\ R_4 := M_{22}T_4, & \end{array}$$

– 7 final additions

$$\begin{array}{ll} P_1 := R_1 + R_2 \ , & P_5 := P_4 + R_3 \ , \\ P_2 := R_1 + R_6 \ , & P_6 := P_3 - R_4 \ , \\ P_3 := P_2 + R_7 \ , & P_7 := P_3 + R_5 \ . \\ P_4 := P_2 + R_5, & \end{array}$$

Then, the final result is $P := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$.

This algorithm works only if the dimension of M and N is equal to a 2-power integer. However, two methods exist to use SW algorithm with any dimension.

Padding and Peeling: On a Quest for All Dimensions. Three methods, namely *static padding*, *dynamic padding* and *dynamic peeling* [20] allow to perform Strassen-Winograd matrix multiplication with two compatible square matrices of arbitrary dimension, i.e., dimension that is equal to a 2-power integer or not.

Static Padding. The *static padding* method checks if the dimension of original matrix M and N is even or not. If not, it pads both matrices with zeros to obtain matrix order that is equal to a 2-power integer. Hence SW can run on these two padded matrices. At the end of the computation, extra rows and columns of zeros are removed.

Dynamic Padding. The *dynamic padding* method checks if the dimension of original matrices M and N is even or not. If not, it pads matrices with an extra column and and extra row of zeros. In that way, SW is able to produce the four quadrants of equal dimension and the 8 additions as described in Sect. 2.2.

For each of the 7 recursive multiplications, this method checks the parity of matrices to multiply. If the matrices already have an even dimension, then nothing is done since the algorithm can split them in four quadrants. However, if matrices have an odd dimension, then an extra row column and an extra row of zeros are added to them before to split the matrix in four quadrants.

Once the computation of the multiplication of padded matrices is done, the extra row and column are removed.

Dynamic Peeling. Instead of adding an extra row and an extra column to make matrices even sized as in the dynamic padding method, the *dynamic peeling* method removes a row and a column. Let M and N be two square matrices of size d. If d is an odd number, the dynamic peeling builds four quadrants for each matrix as illustrated in Fig. 4.

Since d is an odd number, quadrants $M_{11} := (m_{i,j})_{i,j \in [\![1,d-1]\!]}$ and $N_{11} := (n_{i,j})_{i,j \in [\![1,d-1]\!]}$ are square matrices of even size. Moreover, we define $M_{12} :=$

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

Fig. 4. Dynamic peeling for matrices M and N.

$(m_{i,d})_{i\in[\![1,d-1]\!]}$, $M_{21} := (m_{d,j})_{j\in[\![1,d-1]\!]}$, $M_{22} := m_{d,d}$, $N_{12} := (n_{i,d})_{i\in[\![1,d-1]\!]}$, $N_{21} := (n_{d,j})_{j\in[\![1,d-1]\!]}$, and $N_{22} := n_{d,d}$. Hence, the multiplication of M with N is given using blocks multiplication

$$MN := \begin{bmatrix} M_{11}N_{11} + M_{12}N_{21} & M_{11}N_{12} + M_{12}N_{22} \\ M_{21}N_{11} + M_{22}N_{21} & M_{21}N_{12} + M_{22}N_{22} \end{bmatrix},$$

where the product $M_{11}N_{11}$ is computed using the SW and the dynamic peeling method if needed, while other block multiplications are computed using standard matrix multiplication.

3 Strassen-Winograd Matrix Multiplication with MapReduce

We present our three MapReduce protocols that compute the multiplication of square matrices M and N using the Strassen-Winograd algorithm. The first one is the Strassen-Winograd matrix multiplication, denoted SM3, and assumes that matrices' dimension is a 2-power integer. The second one (resp. third one) denoted SM3-Pad (resp. SM3-Peel) is the Strassen-Winograd matrix multiplication using the *dynamic padding* (resp. *dynamic peeling*) method and considers square matrices of any dimension.

Each protocol is decomposed in two phases: (i) the *deconstruction* phase, and (ii) the *combination* phase. The aim of the deconstruction phase is to divide recursively M and N until the recursive Strassen-Winograd matrix multiplications have an order that is equal to 1. The aim of the combination phase is to combine all results of scalar multiplications to build $P := MN$. Each phase is composed of a Map function and of a Reduce function. Due to the recursive nature of the Strassen-Winograd algorithm, each phase is run several times depending on the protocol. At the last round of the combination phase of each protocol, the public cloud obtains $P := MN$ and sends it to the user.

3.1 Strassen-Winograd MapReduce Protocol

The Strassen-Winograd matrix multiplication protocol, denoted SM3, assumes that M and N are two matrices such that $M, N \in \mathbb{R}^{d\times d}$ and $\ell := \log_2(d) \in \mathbb{N}^*$.

Deconstruction Phase. We present the deconstruction phase of SM3. The Map function (resp. the Reduce function) of the deconstruction phase is presented in Fig. 5 (resp. Fig. 6).

```
Map function:

Input: $(key, value)$
// $key$: id of a chunk of $M$ or $N$
// $value$: collection of $(i, j, m_{i,j})$ or $(k, l, n_{k,l})$
foreach $(i, j, m_{i,j}) \in value$ do
    $\texttt{emit}_{\mathcal{M}\to\mathcal{D}_1}(0, (\texttt{M}, i, j, m_{i,j}, d))$
foreach $(k, l, n_{k,l}) \in value$ do
    $\texttt{emit}_{\mathcal{N}\to\mathcal{D}_1}(0, (\texttt{N}, k, l, n_{k,l}, d))$
```

Fig. 5. Map function for the deconstruction phase of the SM3 protocol.

- *The Map Function.* It is run only during the first MapReduce round of the deconstruction phase by sets of nodes $\mathcal{M}$ and $\mathcal{N}$. It consists in rewriting each matrix element sent by data owners in the form of key-value pair such that they share the same key initialized to 0. Hence, when the set of nodes $\mathcal{M}$ receives chunks of M from the owner, the Map function creates for each matrix element $m_{i,j}$ the key-value pair $(0, (\texttt{M}, i, j, m_{i,j}, d))$, where d is the dimension of M. Likewise, when the set of nodes $\mathcal{N}$ receives chunks of N from the owner, the Map function creates for each matrix element $n_{k,l}$ the key-value pair $(0, (\texttt{N}, k, l, n_{k,l}, d))$. We stress that `M` and `N` in the values are the names of matrices, that can be encoded with a single bit, and not the matrices themselves. During other rounds of the deconstruction phase, the Map function is the identity function.
- *The Reduce Function.* It is executed by nodes $\mathcal{D}_s$, with $s \in [\![1, \ell]\!]$. Each key is associated to two matrices sent from previous nodes. When $s = 1$, matrices are M and N and are sent by nodes $\mathcal{M}$ and $\mathcal{N}$. When $s \in [\![2, \ell]\!]$, the two matrices correspond to a recursive matrix multiplication and are sent by $\mathcal{D}_{s-1}$. The Reduce function follows the Strassen-Winograd algorithm using these two matrices. Since the Strassen-Winograd algorithm needs to compute 7 recursive matrix multiplications, the Reduce function produces key-value pairs for 7 different keys where each key is associated to a pair of submatrices to multiply. These key-value pairs are sent to the next nodes of the deconstruction phase $\mathcal{D}_{s+1}$. For the last round of the deconstruction phase, i.e., when $s = \ell$, matrix multiplications are degenerated into scalar multiplications. Hence, the Reduce function produces key-values pairs with the result of scalar multiplications and sends them to the set of nodes $\mathcal{C}_1$.

Combination Phase. We present the combination phase of SM3. In this phase, the Map function is just the identity function. The Reduce function of the combination phase is presented in Fig. 7.

```
Reduce function:

Input: $(key, values)$
// $key$: $t \in \{0,7\}^\ell$
// $values$: collection of $(\mathtt{M}, i, j, m_{i,j}, \delta)$ or $(\mathtt{N}, k, l, n_{k,l}, \delta)$

// Build $M$ and $N$ from values
$M := (m_{i,j})_{(\mathtt{M},i,j,m_{i,j},\delta) \in values}$
$N := (n_{k,l})_{(\mathtt{N},k,l,n_{k,l},\delta) \in values}$

// Split $M$ and $N$ into four quadrants of equal dimension
$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} := M \quad , \quad \begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix} := N$

// Build submatrices according to the Strassen-Winograd algorithm
$S_1 := M_{21} + M_{22}$    $S_3 := M_{11} - M_{21}$    $T_1 := N_{12} - N_{11}$    $T_3 := N_{22} - N_{12}$
$S_2 := S_1 - M_{11}$       $S_4 := M_{12} - S_2$       $T_2 := N_{22} - T_1$       $T_4 := T_2 - N_{21}$

// Create a list $L$ containing couple of matrices
$L := \big[[M_{11}, N_{11}], [M_{12}, N_{21}], [S_4, N_{22}], [M_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$

if $\delta > 2$ then
    $\delta' := \delta/2$
    $\ell' := \log_2(d/\delta')$
    foreach $u \in [\![1,7]\!]$ do
        $(m'_{v,w})_{v,w \in [\![1,\delta']\!]} := L[u][0]$
        $(n'_{v,w})_{v,w \in [\![1,\delta']\!]} := L[u][1]$
        foreach $(v,w) \in [\![1,\delta']\!]^2$ do
            $\mathtt{emit}_{\mathcal{D}_{\ell'-1} \to \mathcal{D}_{\ell'}}(t \| u, (\mathtt{M}, v, w, m'_{v,w}, \delta'))$
            $\mathtt{emit}_{\mathcal{D}_{\ell'-1} \to \mathcal{D}_{\ell'}}(t \| u, (\mathtt{N}, v, w, n'_{v,w}, \delta'))$
else
    foreach $u \in [\![1,7]\!]$ do
        $\mathtt{emit}_{\mathcal{D}_\ell \to \mathcal{C}_1}(t, (u, 1, 1, L[u][0] \cdot L[u][1], 1))$
```

Fig. 6. Reduce function for the deconstruction phase of the SM3 protocol.

- *The Map Function.* The Map function corresponds to the identity function.

- *The Reduce Function.* It is executed by each set of nodes $\mathcal{C}_s$ with $s \in [\![1, \ell]\!]$. For the first round of the combination phase, i.e., when $s = 1$, each key is associated to 7 values corresponding to the scalar multiplications sent by $\mathcal{D}_\ell$. The Reduce function follows the Strassen-Winograd algorithm and combines all these values to build matrices of dimension 2 that is sent to the next nodes $\mathcal{C}_2$. Other rounds of the combination phase work in the same way but in this case, each key is associated to 7 matrices of dimension δ and produces a matrix of dimension $2 \cdot \delta$. At the last round, i.e., $s = \ell$, the Reduce function produces key-value pairs corresponding to the final result $P = MN$ and send them to the user's set of nodes $\mathcal{P}$.

```
Reduce function:

Input: $(key, values)$
// $key$: $t_0 \dots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 7]\!]$ for $z \in [\![0, e]\!]$
// $values$: collection of $(u, i, j, r_{i,j}, \delta)$ such that $u \in [\![1, 7]\!]$ and $i, j \in [\![1, \delta]\!]$

// Build matrices $R_u$ from values with $u \in [\![1, 7]\!]$
foreach $u \in [\![1, 7]\!]$ do
    $R_u := (r_{i,j})_{(u,i,j,r_{i,j},\delta) \in values}$

$P_1 := R_1 + R_2$    $P_3 := P_2 + R_7$    $P_5 := P_4 + R_3$    $P_7 := P_3 - R_5$
$P_2 := R_1 + R_6$    $P_4 := P_2 + R_5$    $P_6 := P_3 - R_4$

$(p_{v,w})_{v,w \in [\![1, 2\cdot\delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$
if $\delta < d$ then
    $\delta' := 2 \cdot \delta$
    $\ell' := \log_2(\delta')$
    foreach $(v, w) \in [\![1, 2 \cdot \delta']\!]^2$ do
        emit$_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t_0 \dots t_{e-1}, (t_e, i, j, p_{v,w}, 2 \cdot \delta'))$
else
    foreach $(v, w) \in [\![1, d]\!]^2$ do
        emit$_{\mathcal{D}_\ell \to \mathcal{P}}((v, w), p_{v,w})$
```

Fig. 7. Reduce function for the combination phase of the SM3 protocol.

3.2 Strassen-Winograd MapReduce Protocol with the Dynamic Padding Method

The Strassen-Winograd matrix multiplication protocol with dynamic padding using the MapReduce paradigm is denoted SM3-Pad. It assumes that M and N are two square matrices such that $M, N \in \mathbb{R}^{d \times d}$ and $d \in \mathbb{N}^*$. In other terms, SM3-Pad consider two compatible square matrices of arbitrary dimension.

Deconstruction Phase. We present the deconstruction phase of SM3-Pad. The difference compared to the SM3 protocol is the use of the dynamic padding in the Reduce function. Hence, each time it is required, the Reduce function adds an extra column and an extra row of zeros to both matrices in order they have an even size dimension. The Map function (resp. the Reduce function) of the deconstruction phase is presented in Fig. 8 (resp. Fig. 9).

– *The Map Function.* It is executed only during the first MapReduce round of the deconstruction phase by the set of nodes $\mathcal{D}_1$. The only difference with the Map function of SM3 is the adding of the padding flag denoted pad and initialized to the empty string ε. For other rounds, the Map function is the identity function.
– *The Reduce Function.* It is executed during each MapReduce round of the deconstruction phase. The difference with the Reduce function of SM3 is that before to split both matrices formed from received key-value pairs, it checks if the matrices' dimension is odd or not. If that is the case, the padding flag

```
Map function:
Input: $(key, value)$
// $key$: id of a chunk of $M$ or $N$
// $value$: collection of $(i, j, m_{i,j})$ or $(k, l, n_{k,l})$
pad := $\varepsilon$                                   // $\varepsilon$ denotes the empty string
foreach $(i, j, m_{i,j}) \in value$ do
    emit$_{\mathcal{M}\to\mathcal{D}_1}(0, (\mathtt{M}, i, j, m_{i,j}, d, \varepsilon))$
foreach $(k, l, n_{k,l}) \in value$ do
    emit$_{\mathcal{N}\to\mathcal{D}_1}(0, (\mathtt{N}, k, l, n_{k,l}, d, \varepsilon))$
```

Fig. 8. Map function for the deconstruction phase of the SM3-Pad protocol.

pad is updated, i.e., it concatenates the character P (for padding), otherwise it concatenated the character E (for even). Moreover, when the matrices' dimension is odd, the Reduce function adds an extra column and an extra row of zeros to both matrices. Hence, matrices have an even dimension and can be split as in the Reduce function of SM3. Since, the Reduce function adds an extra dimensions to matrices when their dimension is odd, the deconstruction phase runs on $\lceil \log_2(d) \rceil$ MapReduce rounds.

Combination Phase. We present the combination phase of SM3-Pad. This phase deals with the extra column and the extra row added during the deconstruction phase using the padding flag pad introduced in the deconstruction phase. Indeed, when two padded matrices are multiplied, the result has also an extra column and an extra row. In this phase, the Map function is just the identity function. The Reduce function of the combination phase is presented in Fig. 10.

– *The Reduce Function.* It works as the Reduce function of SM3, however the Reduce function checks at each round the value of the last character of the padding tag pad. If it is equal to P it means that the obtained matrix is padded. Hence, the Reduce function removes the extra column and the extra row. Moreover, it updates the padding tag by removing the last character. Since matrices are padded when their dimension is odd, the Reduce function is performed $\lceil \log_2(d) \rceil$ times.

3.3 Strassen-Winograd MapReduce Protocol with the Dynamic Peeling Method

The Strassen-Winograd matrix multiplication protocol with dynamic peeling using the MapReduce paradigm is denoted SM3-Peel. As for SM3-Pad, it considers two compatible square matrices of arbitrary dimension, i.e., $M, N \in \mathbb{R}^{d\times d}$ and $d \in \mathbb{N}^*$.

Deconstruction Phase. We present the deconstruction phase of SM3-Peel. When it is required, i.e., when matrices' dimension is odd, it uses the dynamic

```
Reduce function:
Input: $(key, values)$
// $key$: $t_0 \dots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 7]\!]$ for $z \in [\![0, e]\!]$
// $values$: collection of $(\mathtt{M}, i, j, m_{i,j}, \delta, \mathsf{pad})$ or $(\mathtt{N}, k, l, n_{k,l}, \delta, \mathsf{pad})$

// Build $M$ and $N$ from values
$M := (m_{i,j})_{(\mathtt{M},i,j,m_{i,j},\delta,\mathsf{pad}) \in values}$
$N := (n_{k,l})_{(\mathtt{N},k,l,n_{k,l},\delta,\mathsf{pad}) \in values}$

// Apply dynamic padding if dimension is odd
if $\delta \not\equiv 0 \pmod 2$ then
    $\mathsf{pad} := \mathsf{pad} \| \mathtt{P}$
    $\delta := \delta + 1$
    $M' := \left[\begin{array}{c|c} M & \begin{matrix}0\\ \vdots\end{matrix} \\ \hline 0 \cdots & 0 \end{array}\right]$ ,   $N' := \left[\begin{array}{c|c} N & \begin{matrix}0\\ \vdots\end{matrix} \\ \hline 0 \cdots & 0 \end{array}\right]$
else
    $\mathsf{pad} := \mathsf{pad} \| \mathtt{E}$
    $M' := M$ ,   $N' := N$

// Split $M'$ and $N'$ into four quadrants of equal dimension
$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} := M'$ ,   $\begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix} := N'$
// Build submatrices according to the Strassen-Winograd algorithm
$S_1 := M_{21} + M_{22}$   $S_3 := M_{11} - M_{21}$   $T_1 := N_{12} - N_{11}$   $T_3 := N_{22} - N_{12}$
$S_2 := S_1 - M_{11}$   $S_4 := M_{12} - S_2$   $T_2 := N_{22} - T_1$   $T_4 := T_2 - N_{21}$
// Create a list $L$ containing couple of matrices
$L := \big[[M_{11}, N_{11}], [M_{12}, N_{21}], [S_4, N_{22}], [M_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$
if $\delta > 2$ then
    $\delta' := \delta/2$
    $\ell' := \log_2(d/\delta')$
    foreach $u \in [\![1, 7]\!]$ do
        $(m'_{v,w})_{v,w \in [\![1,\delta']\!]} := L[u][0]$
        $(n'_{v,w})_{v,w \in [\![1,\delta']\!]} := L[u][1]$
        foreach $(v, w) \in [\![1, \delta']\!]^2$ do
            $\mathtt{emit}_{\mathcal{D}_{\ell'-1} \to \mathcal{D}_{\ell'}}(t \| u, (\mathtt{M}, v, w, m'_{v,w}, \delta', \mathsf{pad}))$
            $\mathtt{emit}_{\mathcal{D}_{\ell'-1} \to \mathcal{D}_{\ell'}}(t \| u, (\mathtt{N}, v, w, n'_{v,w}, \delta', \mathsf{pad}))$
else
    foreach $u \in [\![1, 7]\!]$ do
        $\mathtt{emit}_{\mathcal{D}_\ell \to \mathcal{C}_1}(t, (u, 1, 1, L[u][0] \cdot L[u][1], 1, \mathsf{pad}))$
```

Fig. 9. Reduce function for the deconstruction phase of the SM3-Pad protocol.

peeling to split the two matrices. The Map function is exactly the same than the Map function of the deconstruction phase of the SM3 protocol and is presented in Fig. 5. The Reduce function of the deconstruction phase is presented in Fig. 17.

```
Reduce function:

Input: $(key, values)$
// $key$: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 7]\!]$ for $z \in [\![0, e]\!]$
// $values$: collection of $(u, i, j, r_{i,j}, \delta, p_0 \dots p_s)$ such that $u \in [\![1, 7]\!]$, $i, j \in [\![1, \delta]\!]$, $s \in [\![0, \ell]\!]$
//           and $p_s \in \{\mathtt{P}, \mathtt{E}\}$

// Build matrices $R_u$ from values with $u \in [\![1, 7]\!]$
foreach $u \in [\![1, 7]\!]$ do
    $R_u := (r_{i,j})_{(u,i,j,r_{i,j},\delta,p_0 \dots p_s) \in values}$

$P_1 := R_1 + R_2$    $P_3 := P_2 + R_7$    $P_5 := P_4 + R_3$    $P_7 := P_3 - R_5$
$P_2 := R_1 + R_6$    $P_4 := P_2 + R_5$    $P_6 := P_3 - R_4$

$(p_{k,l})_{k,l \in [\![1, 2 \cdot \delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$
if $p_s = \mathtt{P}$ then
    $\delta' := 2 \cdot \delta - 1$
else
    $\delta' := 2 \cdot \delta$
if $\delta' < d$ then
    $\ell' := \log_2(2 \cdot \delta)$
    foreach $(k, l) \in [\![1, \delta']\!]^2$ do
        $\mathtt{emit}_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t_0 \cdots t_{e-1}, (t_e, k, l, p_{k,l}, \delta', p_0 \dots p_{s-1}))$
else
    foreach $(k, l) \in [\![1, d]\!]^2$ do
        $\mathtt{emit}_{\mathcal{D}_\ell \to \mathcal{P}}((k, l), p_{k,l})$
```

Fig. 10. Reduce function for the combination phase of the SW-Pad protocol.

- *The Map Function.* It is executed only during the first MapReduce round of the deconstruction phase, i.e., by set of nodes $\mathcal{D}_1$. It works as the Map function of SM3 protocol. The Map function for other rounds of the deconstruction phase is the identity function.
- *The Reduce Function.* It is executed during each MapReduce round of the deconstruction phase, i.e., by sets of nodes $\mathcal{D}_s$ with $s \in [\![1, \ell]\!]$ where $\ell := \lfloor \log_2(d) \rfloor$. When $\mathcal{D}_s$ receives for a certain key the two matrices to multiply denoted M_0 and N_0, it checks the parity of their dimension. If it is odd, the Reduce function splits M_0 and N_0 using the dynamic peeling method and obtains
$$M_0 := \begin{bmatrix} M' & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \;, \quad N_0 := \begin{bmatrix} N' & N_{12} \\ N_{21} & N_{22} \end{bmatrix} \;.$$
  Then, it follows the Strassen-Winograd algorithm for the multiplication between M' and N' blocks, and compute standard matrix multiplication for other blocks.
  Otherwise, the Reduce function follows the Strassen-Winograd algorithm with matrices M_0 and N_0.

```
Reduce function:

Input: $(key, values)$
// $key$: $t_0 \cdots t_e$ such that $e \in [\![0,\ell]\!]$ and $t_z \in [\![0,6]\!]$ for $z \in [\![0,e]\!]$
// $values$: collection of $(\mathtt{M}, i, j, m_{ij}, \delta)$ or $(\mathtt{N}, j, k, n_{jk}, \delta)$ or $(\mathtt{P}, j, k, p_{jk}, \delta_p)$
foreach $(\mathtt{P}, i, j, p_{ij}, \delta_p) \in values$ do
    emit$(t_0 \cdots t_e, (\mathtt{Q}, i, j, p_{ij}, \delta_p))$
$M := (m_{ij})_{(\mathtt{M},i,j,m_{ij},\delta) \in values}$
$N := (n_{jk})_{(\mathtt{N},j,k,n_{jk},\delta) \in values}$
if $\delta \not\equiv 0 \pmod 2$ then
    $\begin{bmatrix} M' & M_{12} \\ M_{21} & M_{22} \end{bmatrix} := M$ ,   $\begin{bmatrix} N' & N_{12} \\ N_{21} & N_{22} \end{bmatrix} := N$ ,
    such that $\begin{cases} M' := (m_{ij})_{i,j \in [\![1,\delta-1]\!]} \\ M_{12} := (m_{ij})_{i \in [\![1,\delta-1]\!], j=\delta} \\ M_{21} := (m_{ij})_{i=\delta, j \in [\![1,\delta-1]\!]} \\ M_{22} := (m_{ij})_{i=\delta, j=\delta} \end{cases}$ , and $\begin{cases} N' := (n_{ij})_{i,j \in [\![1,\delta-1]\!]} \\ N_{12} := (n_{ij})_{i \in [\![1,\delta-1]\!], j=\delta} \\ N_{21} := (n_{ij})_{i=\delta, j \in [\![1,\delta-1]\!]} \\ N_{22} := (n_{ij})_{i=\delta, j=\delta} \end{cases}$
    $(q_{i,j})_{i,j \in [\![1,\delta]\!]} := \begin{bmatrix} M_{12}N_{21} & M'N_{12} + M_{12}N_{22} \\ M_{21}N' + M_{22}N_{21} & M_{21}N_{22} + M_{22}N_{21} \end{bmatrix}$
    $\delta' := (\delta - 1)/2$
    foreach $(i,j) \in [\![1,\delta]\!]^2$ do
        emit$(t_0 \ldots t_e, (\mathtt{Q}, i, j, q_{i,j}, \delta'))$
else
    $M' := M$ ,   $N' := N$
    $\delta' := \delta/2$
// Split $M'$ and $N'$ into four quadrants of equal dimension
$\begin{bmatrix} M'_{11} & M'_{12} \\ M'_{21} & M'_{22} \end{bmatrix} := M'$   ,   $\begin{bmatrix} N'_{11} & N'_{12} \\ N'_{21} & N'_{22} \end{bmatrix} := N'$
$S_1 := M'_{21} + M'_{22}$   $S_3 := M'_{11} - M'_{21}$   $T_1 := N'_{12} - N'_{11}$   $T_3 := N'_{22} - N'_{12}$
$S_2 := S_1 - M'_{11}$   $S_4 := M'_{12} - S_2$   $T_2 := N'_{22} - T_1$   $T_4 := T_2 - N'_{21}$
$L := \big[[M'_{11}, N'_{11}], [M'_{12}, N'_{21}], [S_4, N'_{22}], [M'_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$
if $\delta > 2$ then
    $\ell' := \log_2(d/\delta')$
    foreach $u \in [\![0,6]\!]$ do
        $(m'_{v,w})_{i,j \in [\![1,\delta']\!]} := L[u][0]$
        $(n'_{v,w})_{j,k \in [\![1,\delta']\!]} := L[u][1]$
        foreach $(v,w) \in [\![1,\delta']\!]^2$ do
            emit$_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t \| u, (\mathtt{M}, v, w, m'_{v,w}, \delta'))$
            emit$_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t \| u, (\mathtt{N}, v, w, n'_{v,w}, \delta'))$
else
    foreach $u \in [\![0,6]\!]$ do
        emit$_{\mathcal{D}_\ell \to \mathcal{C}_1}(t, (u, 1, 1, L[u][0] \cdot L[u][1], 1))$
```

Fig. 11. Reduce function for the deconstruction phase of the SW-Peel protocol.

Combination Phase. We present the combination phase of SM3-Peel. This phase combines results of recursive matrix multiplications to compute matrix $P := MN$. As for the combination phase of the SM3-Pad protocol, it has to deal with the dynamic peeling method used during the deconstruction phase. In this phase, the Map function is just the identity function. The Reduce function of the combination phase is presented in Fig. 12.

– *The Reduce Function.* It is executed by nodes $\mathcal{C}_s$ with $s \in [\![1, \ell]\!]$ where $\ell := \lfloor \log_2(d) \rfloor$. For the same key $t \in \{0, 7\}^\ell$, three different cases are possible depending on the associated values.
  1. If values are only of the form $(u, v, w, r_{v,w}, \delta)$ where $u \in [\![1, 7]\!]$, $r_{v,w} \in \mathbb{R}$, $\delta \in \mathbb{N}^*$, $v, w \in [\![1, \delta]\!]$, then the Reduce function combines values to build matrices $R_1, \ldots, R_7$ sent by $\mathcal{C}_{s-1}$ if $s \neq 1$, $\mathcal{D}_\ell$ otherwise, and follows the Strassen-Winograd algorithm. It emits key-value pairs consisting in the elements of the result of the recursive matrix multiplication.
  2. If values are only of the form $(\mathtt{Q}, i, j, q_{i,j}, \delta_q)$ where $\delta_q \in \mathbb{N}^*$, $i, j \in [\![1, \delta_q]\!]$, and $q_{i,j} \in \mathbb{R}$, then the Reduce function consists in the identity function and sends key-value pairs of the form $(\mathtt{Q}, i, j, q_{i,j}, \delta_q)$ to next set of nodes. Note that it is impossible to have this case during the last round of the combination phase, i.e., for the set of nodes $\mathcal{C}_\ell$.
  3. If values are of the form $(u, v, w, r_{v,w}, \delta)$ and of the form $(\mathtt{Q}, i, j, q_{i,j}, \delta_q)$, it means that the dynamic peeling has been applied and must be considered to compute the result of the recursive matrix multiplication. First, the Reduce function combines values of the form $(u, v, w, r_{v,w}, \delta)$ to build matrices $R_1, \ldots, R_7$ and follows the Strassen-Winograd algorithm. Moreover, it uses values of the form $(\mathtt{Q}, i, j, q_{i,j}, \delta_q)$ corresponding to matrix blocks multiplication to obtain the result of the recursive matrix multiplication. Finally, all elements of the obtained matrix are sent to the next set of nodes under the form of key-value, as in SM3.

4 Secure Strassen-Winograd Matrix Multiplication with MapReduce

Protocols SM3, SM3-Pad, and SM3-Peel presented in the previous Section reveal both matrices, intermediate results, and the product of M by N to the public cloud. For instance, nodes $\mathcal{M}$ and $\mathcal{N}$ learn respectively M and N, while the last set of nodes of the combination phase learns $P := MN$. Below, we describe these protocols with a secure approach.

We assume that the MapReduce's user has a Paillier public key denoted pk which is available to the data owners and the public cloud. Since we use Paillier's cryptosystem, the matrix multiplication is computed modulo n, where n is the modulo of pk.

```
Reduce function:
Input: $(key, values)$
// $key$: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 6]\!]$ for $z \in [\![0, e]\!]$
// $values$: collection of $(u, v, w, r_{v,w}, \delta)$ or $(\mathtt{Q}, i, j, q_{i,j}, \delta_\mathtt{q})$ such that $u \in [\![0, 6]\!]$, $i, j \in [\![1, \delta]\!]$
if $\exists\big((\mathtt{Q}, i, j, q_{i,j}, \delta_\mathtt{q}) \wedge (u, v, w, r_{v,w}, \delta)\big) \in values$ then
    $Q := (q_{i,j})_{(\mathtt{Q},i,j,q_{i,j},\delta_\mathtt{q}) \in values}$
    foreach $u \in [\![0, 6]\!]$ do
        $R_u := (r_{v,w})_{(u,v,w,r_{v,w},\delta) \in values}$
    $C_1 := R_0 + R_1$          $C_5 := C_4 + R_2$
    $C_2 := R_0 + R_5$          $C_6 := C_3 - R_3$
    $C_3 := C_2 + R_6$          $C_7 := C_3 - R_4$
    $C_4 := C_2 + R_4$
    $(c_{i,j})_{i,j \in [\![1, 2\cdot\delta]\!]} := \begin{bmatrix} C_1 & C_5 \\ C_6 & C_7 \end{bmatrix}$
    $c'_{i,j} := \begin{cases} c_{i,j} + q_{i,j} & \text{if} \quad i, j \in [\![1, 2\cdot\delta]\!] \\ q_{i,j} & \text{if} \quad \max_{i,j \in [\![1, 2\cdot\delta+1]\!]}(i, j) = 2\cdot\delta + 1 \end{cases}$
    $\delta' := 2\cdot\delta + 1$
    if $\delta' < d$ then
        foreach $(i, j) \in [\![1, \delta']\!]^2$ do
            $\mathtt{emit}_{\mathcal{C}\to\mathcal{C}}(t_0 \cdots t_{e-1}, (t_e, i, j, c'_{i,j}, \delta'))$
    else
        foreach $(i, j) \in [\![1, d]\!]^2$ do
            $\mathtt{emit}_{\mathcal{C}\to\mathcal{C}}((i, j), c'_{i,j})$
else if $\big\{\exists(\mathtt{Q}, i, j, q_{i,j}, \delta_\mathtt{q}) \in values\big\} \wedge \big\{\nexists(u, v, w, r_{v,w}, \delta) \in values\big\}$ then
    foreach $(\mathtt{Q}, i, j, q_{i,j}, \delta_\mathtt{q}) \in values$ do
        $\mathtt{emit}(t_0 \cdots t_e, (\mathtt{Q}, i, j, q_{i,j}, \delta_\mathtt{q}))$
else if $\big\{\nexists(\mathtt{Q}, i, j, q_{i,j}, \delta_q) \in values\big\} \wedge \big\{\exists(u, v, w, r_{v,w}, \delta) \in values\big\}$ then
    foreach $u \in [\![0, 6]\!]$ do
        $R_u := (r_{v,w})_{(u,v,w,r_{v,w},\delta) \in values}$
    $C_1 := R_0 + R_1$          $C_5 := C_4 + R_2$
    $C_2 := R_0 + R_5$          $C_6 := C_3 - R_3$
    $C_3 := C_2 + R_6$          $C_7 := C_3 - R_4$
    $C_4 := C_2 + R_4$
    $(c_{i,j})_{i,j \in [\![1, 2\cdot\delta]\!]} := \begin{bmatrix} C_1 & C_5 \\ C_6 & C_7 \end{bmatrix}$
    $\delta' := 2\cdot\delta$
    if $\delta' < d$ then
        foreach $(i, j) \in [\![1, \delta']\!]^2$ do
            $\mathtt{emit}_{\mathcal{C}\to\mathcal{C}}(t_0 \cdots t_{e-1}, (t_e, i, j, c'_{i,j}, \delta'))$
    else
        foreach $(i, j) \in [\![1, d]\!]^2$ do
            $\mathtt{emit}_{\mathcal{C}\to\mathcal{C}}((i, j), c'_{i,j})$
```

Fig. 12. Reduce function for the combination phase of the SW-Peel protocol.

4.1 Preprocessing for Secure Strassen-Winograd Matrix Multiplication

In order to avoid the public cloud from learning the content of the two matrices and the result of their product, each data owner performs a preprocessing on its own matrix. This preprocessing is done in a way allowing the public cloud to perform the same computation, as in protocols presented in the previous Section, in a partially homomorphic way while privacy constraints are satisfied. To run the preprocessing, data owners use the Paillier public key pk of the MapReduce's user where $pk := (n, g)$, and n being the product of two prime numbers generated according to a security parameter λ, and $g \in \mathbb{Z}^*_{n^2}$.

The preprocessing is simple. It consists in the encryption of each element of the matrix owned by the data owner using the Paillier's cryptosystem with the public key pk of the MapReduce's user. At the end of the encryption, it outputs the corresponding encrypted matrix. In the following, we denote by a star an encrypted matrix, i.e., M^* is the encrypted matrix associated to M. Moreover, elements of M^* are denoted $m^*_{i,j}$ for $i, j \in [\![1, d]\!]$, where d is the dimension of the square matrix M.

4.2 Secure Approach

The secure approach for SM3 protocol (resp. SM3-Pad, SM3-Peel) is denoted S2M3 (resp. S2M3-Pad, S2M3-Peel). The three secure protocols use the Paillier's cryptosystem and its partial homomorphic properties to ensure privacy of elements of matrices and to allow the public cloud to compute the matrix multiplication.

In our secure approach, we assume that the MapReduce's user and the public cloud do not collude, i.e., the public cloud does not know the secret key sk of the MapReduce's user. Indeed, if that is the case then the public cloud is able to decrypt all ciphertexts, and then to learn the content of both matrices and the result of the matrix multiplication.

The three secure protocols are similar to protocols presented in the previous Section. Each protocol is also decomposed into the deconstruction phase and the combination phase. Moreover, secure approaches have the same number of rounds for each phase than their plain version.

For the sake of clarity, we define the two following functions used in secure approaches.

- $\mathsf{Paillier.Add}(pk, A, B)$. This function takes matrices $A := (\mathcal{E}(pk, a_{i,j}))_{i,j \in [\![1,d]\!]}$ and $B := (\mathcal{E}(pk, b_{i,j}))_{i,j \in [\![1,d]\!]}$ as input such that $A, B \in (\mathbb{Z}^*_{n^2})^{d \times d}$. For each $(i, j) \in [\![1, d]\!]^2$, the function computes $c_{i,j} := \mathcal{E}(pk, a_{i,j}) \cdot \mathcal{E}(pk, b_{i,j})$ and outputs the encrypted matrix $C := (c_{i,j})_{i,j \in [\![1,d]\!]}$ that correspond to the encryption of the sum of A and B.
- $\mathsf{Paillier.Sub}(pk, A, B)$. This function takes matrices $A := (\mathcal{E}(pk, a_{i,j}))_{i,j \in [\![1,d]\!]}$ and $B := (\mathcal{E}(pk, b_{i,j}))_{i,j \in [\![1,d]\!]}$ as input such that $A, B \in (\mathbb{Z}^*_{n^2})^{d \times d}$. For each $(i, j) \in [\![1, d]\!]^2$, the function computes $c_{i,j} := \mathcal{E}(pk, a_{i,j}) \cdot \mathcal{E}(pk, b_{i,j})^{-1}$ and

outputs the encrypted matrix $C := (c_{i,j})_{i,j \in [\![1,d]\!]}$ that correspond to the encryption of the subtraction of B to A.

Moreover, secure approaches use the Paillier interactive multiplicative homomorphic protocol denoted $\mathsf{Paillier.Inter}$ and presented in Fig. 3.

4.3 Secure Strassen-Winograd Matrix Multiplication Protocol

The secure Strassen-Winograd matrix multiplication protocol, denoted S2M3, assumes that M and N are two matrices such that $M, N \in \mathbb{Z}_n^{d \times d}$ and $\ell := \log_2(d) \in \mathbb{N}^*$.

Deconstruction Phase. We present the deconstruction phase of S2M3. The Map function is the same than for SM3 protocol presented in Fig. 5. The only difference is that it operates on encrypted matrices M^* and N^* sent by data owners after the preprocessing.

The Reduce function is presented in Fig. 13. Since it operates on encrypted matrices, we use functions $\mathsf{Paillier.Add}$ and $\mathsf{Paillier.Sub}$ to add or subtract two matrices. Moreover, it uses the Paillier interactive multiplicative homomorphic protocol $\mathsf{Paillier.Inter}$ during the last round of the decomposition phase to the encryption of the multiplication of two elements.

Combination Phase. As for SM3 protocol, the Map function of the combination phase is the identity function. The Reduce function of the combination phase is presented in Fig. 14. The only difference compared to S2M3 protocol is the use of $\mathsf{Paillier.Add}$ and $\mathsf{Paillier.Sub}$ functions for addition and subtraction of encrypted matrices.

4.4 Secure Strassen-Winograd Matrix Multiplication with the Dynamic Padding Method

The secure Strassen-Winograd matrix multiplication protocol with dynamic padding using the MapReduce paradigm is denoted S2M3-Pad. It consider two square matrices of same dimension M and N such that $M, N \in \mathbb{Z}_n^{d \times d}$ and $d \in \mathbb{N}^*$.

Deconstruction Phase. The Map function is the same than for SM3-Pad protocol presented in Fig. 8. The only difference is that it operates on encrypted matrices M^* and N^* sent by data owners after the preprocessing.

The Reduce function is presented in Fig. 15. Note that it pads matrices with encryption of zero instead of zero as for SM3-Pad protocol. Since it operates on encrypted matrices, we use $\mathsf{Paillier.Add}$ and $\mathsf{Paillier.Sub}$ functions to add or subtract two encrypted matrices. Moreover, it uses the Paillier interactive multiplicative homomorphic protocol $\mathsf{Paillier.Inter}$ during the last round of the deconstruction phase to the encryption of the multiplication of two elements.

Combination Phase. As for SM3-Pad, the Map function of the combination phase is the identity function. The Reduce function is presented in Fig. 16. It works as the Reduce function of the combination phase of SM3-Pad protocol.

```
Reduce function:

Input: $(key, values)$
// $key$: $t \in \{0,7\}^{\ell}$
// $values$: collection of $(\mathtt{M}, i, j, m^*_{i,j}, \delta)$ or $(\mathtt{N}, k, l, n^*_{k,l}, \delta)$

// Build $M^*$ and $N^*$ from values
$M^* := (m^*_{i,j})_{(\mathtt{M},i,j,m^*_{i,j},\delta) \in values}$
$N^* := (n^*_{k,l})_{(\mathtt{N},k,l,n^*_{k,l},\delta) \in values}$

// Split $M^*$ and $N^*$ into four quadrants of equal dimension
$\begin{bmatrix} M^*_{11} & M^*_{12} \\ M^*_{21} & M^*_{22} \end{bmatrix} := M^* \quad , \quad \begin{bmatrix} N^*_{11} & N^*_{12} \\ N^*_{21} & N^*_{22} \end{bmatrix} := N^*$

// Build submatrices according to the Strassen-Winograd algorithm
$S_1 := \mathsf{Paillier.Add}(pk, M^*_{21}, M^*_{22})$   $T_1 := \mathsf{Paillier.Sub}(pk, N^*_{12}, N^*_{11})$
$S_2 := \mathsf{Paillier.Sub}(pk, S_1, M^*_{11})$   $T_2 := \mathsf{Paillier.Sub}(pk, N^*_{22}, T_1)$
$S_3 := \mathsf{Paillier.Sub}(pk, M^*_{11}, M^*_{21})$   $T_3 := \mathsf{Paillier.Sub}(pk, N^*_{22}, N^*_{12})$
$S_4 := \mathsf{Paillier.Sub}(pk, M^*_{12}, S_2)$   $T_4 := \mathsf{Paillier.Sub}(pk, T_2, N^*_{21})$

// Create a list $L$ containing couple of matrices
$L := \big[[M^*_{11}, N^*_{11}], [M^*_{12}, N^*_{21}], [S_4, N^*_{22}], [M^*_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$

if $\delta > 2$ then
    $\delta' := \delta/2$
    $\ell' := \log_2(d/\delta')$
    foreach $u \in [\![1,7]\!]$ do
        $(m'_{v,w})_{v,w \in [\![1,\delta']\!]} := L[u][0]$
        $(n'_{v,w})_{v,w \in [\![1,\delta']\!]} := L[u][1]$
        foreach $(v,w) \in [\![1,\delta']\!]^2$ do
            $\mathtt{emit}_{\mathcal{D}_{\ell'-1} \to \mathcal{D}_{\ell'}}(t \| u, (\mathtt{M}, v, w, m'_{v,w}, \delta'))$
            $\mathtt{emit}_{\mathcal{D}_{\ell'-1} \to \mathcal{D}_{\ell'}}(t \| u, (\mathtt{N}, v, w, n'_{v,w}, \delta'))$
else
    foreach $u \in [\![1,7]\!]$ do
        $\mathtt{emit}_{\mathcal{D}_{\ell} \to \mathcal{C}_1}(t, (u, 1, 1, \mathsf{Paillier.Inter}(L[u][0], L[u][1]), 1))$
```

Fig. 13. Reduce function for the deconstruction phase of the S2M3 protocol.

```
Reduce function:

Input: $(key, values)$
// $key$: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 7]\!]$ for $z \in [\![0, e]\!]$
// $values$: collection of $(u, i, j, r^*_{i,j}, \delta)$ such that $u \in [\![1, 7]\!]$ and $i, j \in [\![1, \delta]\!]$

// Build matrices $R_u$ from values with $u \in [\![1, 7]\!]$
foreach $u \in [\![1, 7]\!]$ do
    $R_u := (r_{i,j})_{(u,i,j,r^*_{i,j},\delta) \in values}$

$P_1 := \mathsf{Paillier.Add}(pk, R_1, R_2)$        $P_5 := \mathsf{Paillier.Add}(pk, P_4, R_3)$
$P_2 := \mathsf{Paillier.Add}(pk, R_1, R_6)$        $P_6 := \mathsf{Paillier.Add}(pk, P_3, R_4)$
$P_3 := \mathsf{Paillier.Add}(pk, P_2, R_7)$        $P_7 := \mathsf{Paillier.Sub}(pk, P_3, R_5)$
$P_4 := \mathsf{Paillier.Add}(pk, P_2, R_5)$

$(p_{v,w})_{v,w \in [\![1, 2\cdot\delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$
if $\delta < d$ then
    $\delta' := 2 \cdot \delta$
    $\ell' := \log_2(\delta')$
    foreach $(v, w) \in [\![1, 2 \cdot \delta']\!]^2$ do
        $\mathtt{emit}_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t_0 \cdots t_{e-1}, (t_e, i, j, p_{v,w}, 2 \cdot \delta'))$
else
    foreach $(v, w) \in [\![1, d]\!]^2$ do
        $\mathtt{emit}_{\mathcal{D}_\ell \to \mathcal{P}}((v, w), p_{v,w})$
```

Fig. 14. Reduce function for the combination phase of the S2M3 protocol.

4.5 Secure Strassen-Winograd Matrix Multiplication with the Dynamic Peeling Method

The secure Strassen-Winograd matrix multiplication protocol with dynamic peeling using the MapReduce paradigm is denoted S2M3-Peel. It considers two square matrices M and N such that $M, N \in \mathbb{Z}_n^{d\times d}$ and $d \in \mathbb{N}^*$.

Deconstruction Phase. The Map function is the same than for SM3-Peel protocol presented in Fig. 5. The only difference is that it operates on encrypted matrices M^* and N^* sent by data owners after the preprocessing.

The Reduce function is presented in Fig. 17. Since it operates on encrypted matrices, it uses Paillier.Add and Paillier.Sub functions to add or subtract two encrypted matrices. Moreover, it uses the Paillier interactive multiplicative homomorphic protocol during the last round of the decomposition phase since the multiplication is performed over encrypted values.

Combination Phase. The Map function of the combination phase for the S2M3-Peel protocol is the identity function. The Reduce function of S2M3-Peel protocol is presented in Fig. 18. It works as the Reduce function of the SM3-Peel protocol.

```
Reduce function:
Input: $(key, values)$
// $key$: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 6]\!]$ for $z \in [\![0, e]\!]$
// $values$: collection of $(\mathtt{M}, i, j, m^*_{i,j}, \delta, \mathsf{pad})$ or $(\mathtt{N}, k, l, n^*_{k,l}, \delta, \mathsf{pad})$

// Build $M$ and $N$ from values
$M^* := (m^*_{i,j})_{(\mathtt{M},i,j,m^*_{i,j},\delta,\mathsf{pad}) \in values}$
$N^* := (n^*_{k,l})_{(\mathtt{N},k,l,n^*_{k,l},\delta,\mathsf{pad}) \in values}$

// Apply dynamic padding if dimension is odd
if $\delta \not\equiv 0 \pmod 2$ then
    $\mathsf{pad} := \mathsf{pad} \| \mathsf{P}$
    $\delta := \delta + 1$
    $M' := \left[\begin{array}{c|c} M^* & \begin{matrix}\mathcal{E}(pk,0) \\ \vdots\end{matrix} \\ \hline \mathcal{E}(pk,0) \cdots & \mathcal{E}(pk,0)\end{array}\right]$ , $N' := \left[\begin{array}{c|c} N^* & \begin{matrix}\mathcal{E}(pk,0) \\ \vdots\end{matrix} \\ \hline \mathcal{E}(pk,0) \cdots & \mathcal{E}(pk,0)\end{array}\right]$
else
    $\mathsf{pad} := \mathsf{pad} \| \mathsf{E}$
    $M' := M^*$ , $N' := N^*$

// Split $M'$ and $N'$ into four quadrants of equal dimension
$\begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} := M'$ , $\begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix} := N'$
// Build submatrices according to the Strassen-Winograd algorithm
$S_1 := \mathsf{Paillier.Add}(pk, M_{21}, M_{22})$   $T_1 := \mathsf{Paillier.Sub}(pk, N_{12}, N_{11})$
$S_2 := \mathsf{Paillier.Sub}(pk, S_1, M_{11})$   $T_2 := \mathsf{Paillier.Sub}(pk, N_{22}, T_1)$
$S_3 := \mathsf{Paillier.Sub}(pk, M_{11}, M_{21})$   $T_3 := \mathsf{Paillier.Sub}(pk, N_{22}, N_{12})$
$S_4 := \mathsf{Paillier.Sub}(pk, M_{12}, S_2)$   $T_4 := \mathsf{Paillier.Sub}(pk, T_2, N_{21})$

// Create a list $L$ containing couple of matrices
$L := [[M_{11}, N_{11}], [M_{12}, N_{21}], [S_4, N_{22}], [M_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]]$
if $\delta > 2$ then
    $\delta' := \delta/2$
    $\ell' := \log_2(d/\delta')$
    foreach $u \in [\![1, 7]\!]$ do
        $(m'_{v,w})_{v,w \in [\![1,\delta']\!]} := L[u][0]$
        $(n'_{v,w})_{v,w \in [\![1,\delta']\!]} := L[u][1]$
        foreach $(v, w) \in [\![1, \delta']\!]^2$ do
            $\mathtt{emit}_{\mathcal{D}_{\ell'-1} \to \mathcal{D}_{\ell'}}(t \| u, (\mathtt{M}, v, w, m'_{v,w}, \delta', \mathsf{pad}))$
            $\mathtt{emit}_{\mathcal{D}_{\ell'-1} \to \mathcal{D}_{\ell'}}(t \| u, (\mathtt{N}, v, w, n'_{v,w}, \delta', \mathsf{pad}))$
else
    foreach $u \in [\![1, 7]\!]$ do
        $\mathtt{emit}_{\mathcal{D}_\ell \to \mathcal{C}_1}(t, (u, 1, 1, \mathsf{Paillier.Inter}(L[u][0], L[u][1]), 1, \mathsf{pad}))$
```

Fig. 15. Reduce function for the deconstruction phase of the S2M3-Pad protocol.

5 Experimental Results

We present the experimental results for our SM3 and S2M3 protocols.

```
Reduce function:

Input: $(key, values)$
// $key$: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 7]\!]$ for $z \in [\![0, e]\!]$
// $values$: collection of $(u, i, j, r_{i,j}, \delta, p_0 \dots p_s)$ such that $u \in [\![1, 7]\!]$, $i, j \in [\![1, \delta]\!]$, $s \in [\![0, \ell]\!]$
//            and $p_s \in \{\mathtt{P}, \mathtt{E}\}$

// Build matrices $R_u$ from values with $u \in [\![1, 7]\!]$
foreach $u \in [\![1, 7]\!]$ do
    $R_u := (r_{i,j})_{(u,i,j,r_{i,j},\delta,p_0 \dots p_s) \in values}$

$P_1 :=$ Paillier.Add$(pk, R_1, R_2)$        $P_5 :=$ Paillier.Add$(pk, P_4, R_3)$
$P_2 :=$ Paillier.Add$(pk, R_1, R_6)$        $P_6 :=$ Paillier.Add$(pk, P_3, R_4)$
$P_3 :=$ Paillier.Add$(pk, P_2, R_7)$        $P_7 :=$ Paillier.Sub$(pk, P_3, R_5)$
$P_4 :=$ Paillier.Add$(pk, P_2, R_5)$

$(p_{k,l})_{k,l \in [\![1, 2 \cdot \delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$
if $p_s = \mathtt{P}$ then
    $\delta' := 2 \cdot \delta - 1$
else
    $\delta' := 2 \cdot \delta$
if $\delta' < d$ then
    $\ell' := \log_2(2 \cdot \delta)$
    foreach $(k, l) \in [\![1, \delta']\!]^2$ do
        emit$_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t_0 \cdots t_{e-1}, (t_e, k, l, p_{k,l}, \delta', p_0 \dots p_{s-1}))$
else
    foreach $(k, l) \in [\![1, d]\!]^2$ do
        emit$_{\mathcal{D}_\ell \to \mathcal{P}}((k, l), p_{k,l})$
```

Fig. 16. Reduce function for the combination phase of the S2M3-Pad protocol.

5.1 Dataset and Settings

For each experiment, we generate two random square matrices and of order d such that $240 \leq d \leq 450$ for no-secure protocols, and $90 \leq d \leq 300$ for secure protocols. Elements of both matrices are in $[\![0, 10]\!]$. For each order d, we perform matrix multiplication with SM3 and S2M3 protocols using static padding, dynamic padding, and dynamic peeling methods. We also compare the results to the standard matrix multiplication using one MapReduce round [22] and the secure approach denoted CRSP-1R [4]. For each experiment, we stop the Strassen-Winograd recursive matrix multiplication when the dimension of matrices is less than 16. Then, we use the MM-1R protocol for the no-secure approach, and the CRSP-1R protocol for the secure approach. Our secure protocols are based on the Paillier's cryptosystem. We use Gaillier[4], a Go implementation of the

[4] `https://github.com/actuallyachraf/gomorph`

Paillier's cryptosystem. Note that Gaillier is not an optimized implementation. Hence, we use it with a 64-bit RSA modulus as proof of concept.

5.2 Results

In Fig. 19, we present CPU times for the no-secure SM3 protocols using the static padding method denoted SM3-sPad, the dynamic padding method denoted SM3-Pad, or the dynamic peeling method denoted SM3-Peel. Moreover, we compare them to the standard matrix multiplication using one MapReduce round [22], denoted MM-1R.

First we observe that without any security, our SM3-Pad and SM3-Peel protocols perform the matrix multiplication faster than the standard matrix multiplication for the largest dimensions. This trend can be seen when matrices' dimension is larger than 300. Moreover, we remark that our protocol SM3-sPad is more efficient than the state-of-the-art protocol MM-1R when matrices' dimension tend to a 2-power integer. Indeed, we note that SM3-sPad is faster than MM-1R when matrices' dimension is between 450 and $512 = 2^8$.

We show in Fig. 20, CPU times for secure protocols computing the SW matrix multiplication. We compare them to a state-of-the-art secure standard matrix multiplication protocol CRSP-1R using one MapReduce round [4].

Same observations than no-secure protocols can be done for secure protocols. Indeed, we also remark that our S2M3-Pad and S2M3-Peel protocols perform the matrix multiplication faster than our protocol CRSP-1R.

Finally, for both no-secure and secure protocols, we remark that the protocol using the dynamic peeling method is always faster than the protocol using the dynamic padding method. As we have seen previously, the deconstruction phase and the combination phase use $\lceil \log_2(d) \rceil$ MapReduce rounds for the dynamic padding method, while they use $\lfloor \log_2(d) \rfloor$ MapReduce rounds for the dynamic peeling method, where d is matrices' dimension.

6 Security Proofs

We provide formal security proof for S2M3, S2M3-Pad, and S2M3-Peel protocols. We use the standard multiparty computations definition of security against semi-honest adversaries [23].

6.1 Security Proof for S2M3 Protocol

S2M3 protocol assumes that the public cloud's nodes may collude, hence in a security point of view, all sets of nodes are considered as a unique set of nodes when they collude.

We model S2M3 protocol with four parties P_M, P_N, $P_\mathcal{C}$, and $P_\mathcal{P}$ using respective inputs $I := (I_M, I_N, I_\mathcal{C}, I_\mathcal{P}) \in \mathcal{I}$, and a function $g := (g_M, g_N, g_\mathcal{C}, g_\mathcal{P})$ such that:

- P_M is the data owner of M. It has the input $I_M := (M, pk)$, where M is its private matrix and pk is the Paillier's public key of the MapReduce's user. P_M returns $g_M(I) := \perp$ because it does not learn anything.
- P_N is the data owner of N. It has the input $I_N := (N, pk)$, where N is its private matrix and pk is the Paillier's public key of the MapReduce's user. P_N returns $g_N(I) := \perp$ because it does not learn anything.
- $P_{\mathcal{C}}$ is the public cloud's nodes that represents the collusion between all sets of nodes of the deconstruction phase and of the combination phase. It has the input $I_{\mathcal{C}} := (pk)$, where pk is the Paillier's public key of the user. $P_{\mathcal{C}}$ returns $g_{\mathcal{C}}(I) := d \in \mathbb{N}^*$ because it learns matrices dimensions.
- $P_{\mathcal{P}}$ is the set of nodes $\mathcal{P}$ of the MapReduce's user. It has the input $I_{\mathcal{P}} := (pk, sk)$, where (pk, sk) is the Paillier's key pair of the MapReduce's user $P_{\mathcal{P}}$ returns $g_{\mathcal{P}}(I) := P$ because the user obtains the result of the matrix multiplication at the end of the protocol.

Note that for the sake of clarity, we consider that $P_{\mathcal{C}}$ sends the product of the encrypted matrices to $P_{\mathcal{P}}$ instead of storing them in a database.

The security of S2M3 protocol is given in Theorem 1.

Theorem 1. *Assume Paillier's cryptosystem is IND-CPA, then* S2M3 *securely computes the matrix multiplication in the presence of semi-honest adversaries even if public cloud's nodes collude.*

The security proof for S2M3 protocol (Theorem 1) is decomposed in Lemma 1 for parties P_M and P_N, Lemma 2 for party $P_{\mathcal{C}}$, and Lemma 3 for party $P_{\mathcal{P}}$.

Lemma 1. *There exists probabilistic polynomial-time simulators $\mathcal{S}_M^{\text{S2M3}}$ and $\mathcal{S}_N^{\text{S2M3}}$ such that:*

$$\left\{\mathcal{S}_M^{\text{S2M3}}(1^\lambda, I_M, g_M(I))\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}} \overset{c}{\equiv} \left\{\mathsf{view}_M^{\text{S2M3}}(I, \lambda)\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}},$$
$$\left\{\mathcal{S}_N^{\text{S2M3}}(1^\lambda, I_N, g_N(I))\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}} \overset{c}{\equiv} \left\{\mathsf{view}_N^{\text{S2M3}}(I, \lambda)\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}}.$$

Proof. The view of P_M contains M^* (the encryption of M) obtained from the preprocessing and that is sent to $P_{\mathcal{C}}$. Simulator $\mathcal{S}_M^{\text{S2M3}}$ has input (M, pk). It encrypts each element of M using pk to build M^*. Hence, $\mathcal{S}_M^{\text{S2M3}}$ performs exactly the same computation as S2M3 protocol and describes exactly the same distribution as $\mathsf{view}_M^{\text{S2M3}}(I, \lambda)$. Building the simulator $\mathcal{S}_N^{\text{S2M3}}$ in the same way, it describes exactly the same distribution as $\mathsf{view}_N^{\text{S2M3}}(I, \lambda)$.

Lemma 2. *Assume Paillier's cryptosystem is IND-CPA, then there exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}$ such that:*

$$\left\{\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}(1^\lambda, I_{\mathcal{C}}, g_{\mathcal{C}}(I))\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}} \overset{c}{\equiv} \left\{\mathsf{view}_{\mathcal{C}}^{\text{S2M3}}(I, \lambda)\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}}.$$

Proof. We recall that $P_{\mathcal{C}}$ is the collusion of sets of nodes of the public cloud, i.e., $\mathcal{M}$, $\mathcal{N}$, $\mathcal{D}_i$ and $\mathcal{C}_i$ for $i \in [\![1, \ell]\!]$. $P_{\mathcal{C}}$ receives M^* and N^* from the data owners.

Simulator of $P_{\mathcal{C}}$ is given in Fig. 23. Function $\mathsf{SW\text{-}Deconstruction}$ presented in Fig. 21 simulates the public cloud's view during the deconstruction phase. The view contains all submatrices corresponding the recursive matrices multiplications. Moreover, the last set of nodes of the deconstruction phase $\mathcal{D}_\ell$ sends couples of ciphertexts (x_i, y_i), with $i \in [\![1, 7^\ell]\!]$, to $P_{\mathcal{P}}$ and receives all corresponding ciphertexts $R_i^{(\ell)}$ returned by $P_{\mathcal{P}}$ to compute multiplication on encrypted coefficients.

Function $\mathsf{SW\text{-}Combination}$ presented in Fig. 22 simulates the public cloud's view during one round of the combination phase. For each round, it combines submatrices according the SW algorithm.

Let $\lambda \in \mathbb{N}$ be a security parameter. Assume there exists a polynomial-time distinguisher D such that for all inputs $I \in \mathcal{I}$, we have:

$$\left| \Pr[D(\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}(1^\lambda, I_{\mathcal{C}}, g_{\mathcal{C}}(I))) = 1] - \Pr[D(\mathsf{view}_{\mathcal{C}}^{\text{S2M3}}(I)) = 1] \right| = \mu(\lambda) ,$$

where μ is a non-negligible function in λ. We show how to build a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}$ has a non-negligible advantage to win the IND-CPA experiment on the Paillier's cryptosystem. Then we conclude the proof by contraposition. Adversary $\mathcal{A}$ is presented in Fig. 24. At the end of its execution, $\mathcal{A}$ uses the distinguisher D to compute the bit b^* before returning it. First, we remark that:

$$\Pr\left[\mathsf{Exp}_{\text{Paillier},\mathcal{A}}^{\text{indcpa-0}}(\lambda) = 1\right] = \Pr\left[D(\mathsf{view}_{\mathcal{C}}^{\text{S2M3}}(I, \lambda)) = 1\right] .$$

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

$$\Pr\left[\mathsf{Exp}_{\text{Paillier},\mathcal{A}}^{\text{indcpa-1}}(\lambda) = 1\right] = \Pr\left[D(\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}(1^\lambda, I_{\mathcal{C}}, g_{\mathcal{C}}(I))) = 1\right] .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}$. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}$. Finally, we evaluate the probability that $\mathcal{A}$ wins the IND-CPA experiment:

$$\begin{aligned}\mathsf{Adv}_{\text{Paillier},\mathcal{A}}^{\text{indcpa}}(\lambda) &= \left| \Pr\left[\mathsf{Exp}_{\text{Paillier},\mathcal{A}}^{\text{indcpa-1}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}_{\text{Paillier},\mathcal{A}}^{\text{indcpa-0}}(\lambda) = 1\right] \right| \\ &= \left| \Pr\left[D(\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}(1^\lambda, I_{\mathcal{C}}, g_{\mathcal{C}})) = 1\right] - \Pr\left[D\big(\mathsf{view}_{\mathcal{C}}^{\text{S2M3}}(I, \lambda) = 1\right] \right| \\ &= \mu(\lambda) ,\end{aligned}$$

which is non-negligible and concludes the proof by contradiction.

Lemma 3. *There exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}$ such that:*

$$\left\{\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}(1^\lambda, I_{\mathcal{P}}, g_{\mathcal{P}}(I))\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_{\mathcal{P}}^{\text{S2M3}}(I, \lambda)\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}} .$$

Proof. Simulator $\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}$ is presented in Fig. 25. The view of $P_{\mathcal{P}}$ contains the couple of ciphertexts (x_i, y_i) sent by the set of nodes $\mathcal{D}_\ell$ during the deconstruction phase run by $P_{\mathcal{C}}$ and the answer z_i sent by $P_{\mathcal{P}}$ to $P_{\mathcal{C}}$ that contains the encryption of the multiplication of x_i and y_i, for $i \in [\![1, 7^\ell]\!]$. Since x_i and y_i are randomized by $P_{\mathcal{C}}$, there are indistinguishable to random ciphertexts in the $P_{\mathcal{P}}$ point of view. The view of $P_{\mathcal{P}}$ also contains $P^* := (\mathcal{E}(pk, p_{i,j}))_{i,j \in [\![1,d]\!]}$ that is sent by $P_{\mathcal{C}}$. Finally, $\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}(1^\lambda, (pk, sk), P)$ describes exactly the same distribution as $\mathsf{view}_{\mathcal{P}}^{\text{S2M3}}(I, \lambda)$, which concludes the proof.

6.2 Security Proof for S2M3-Pad Protocol

S2M3-Pad protocol is modeled as the S2M3 protocol by parties P_M, P_N, $P_{\mathcal{C}}$, and $P_{\mathcal{P}}$. The security of S2M3-Pad protocol is given in Theorem 2.

Theorem 2. *Assume Paillier's cryptosystem is IND-CPA, then* S2M3-Pad *securely computes the matrix multiplication in the presence of semi-honest adversaries even if public cloud's nodes collude.*

The security proof for S2M3-Pad protocol (Theorem 2) is decomposed in Lemma 4 for parties P_M and P_N, Lemma 5 for party $P_{\mathcal{C}}$, and Lemma 6 for party $P_{\mathcal{P}}$.

Lemma 4. *There exists two probabilistic polynomial-time simulators $\mathcal{S}_M^{\text{S2M3-Pad}}$ and $\mathcal{S}_N^{\text{S2M3-Pad}}$ such that:*

$$\left\{\mathcal{S}_M^{\text{S2M3-Pad}}(1^\lambda, I_M, g_M(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{c}{\equiv} \left\{\mathsf{view}_M^{\text{S2M3-Pad}}(I, \lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}},$$
$$\left\{\mathcal{S}_N^{\text{S2M3-Pad}}(1^\lambda, I_N, g_N(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{c}{\equiv} \left\{\mathsf{view}_N^{\text{S2M3-Pad}}(I, \lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Proof. Since the view of P_M and P_N are exactly the same than for the S2M3 protocol, the proof is the same than Lemma 1.

Lemma 5. *Assume Paillier's cryptosystem is IND-CPA, then there exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3-Pad}}$ such that:*

$$\left\{\mathcal{S}_{\mathcal{C}}^{\text{S2M3-Pad}}(1^\lambda, I_{\mathcal{C}}, g_{\mathcal{C}}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{c}{\equiv} \left\{\mathsf{view}_{\mathcal{C}}^{\text{S2M3-Pad}}(I, \lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Proof. The only difference with the S2M3 protocol is the adding of the padding (with encryption of zeros) during the deconstruction phase, and the removing of the padding during the combination phase. In the security point of view, it is equivalent to deal with padded encrypted matrices or not. Hence, the proof is the same than Lemma 2.

Lemma 6. *There exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{P}}^{\text{S2M3-Pad}}$ such that:*

$$\left\{\mathcal{S}_{\mathcal{P}}^{\text{S2M3-Pad}}(1^\lambda, I_{\mathcal{P}}, g_{\mathcal{P}}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{c}{\equiv} \left\{\mathsf{view}_{\mathcal{P}}^{\text{S2M3-Pad}}(I, \lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Proof. The proof is exactly the same than Lemma 3.

6.3 Security Proof for S2M3-Peel Protocol

S2M3-Peel protocol is modeled as the S2M3 and S2M3-Pad protocols by parties P_M, P_N, $P_{\mathcal{C}}$, and $P_{\mathcal{P}}$. The security of S2M3-Peel protocol is given in Theorem 3.

Theorem 3. *Assume Paillier's cryptosystem is IND-CPA, then* S2M3-Peel *securely computes the matrix multiplication in the presence of semi-honest adversaries even if public cloud's nodes collude.*

The security proof for S2M3-Peel protocol (Theorem 3) is decomposed in Lemma 7 for parties P_M and P_N, Lemma 8 for party $P_{\mathcal{C}}$, and Lemma 9 for party $P_{\mathcal{P}}$.

Lemma 7. *There exists two probabilistic polynomial-time simulators* $\mathcal{S}_M^{\text{S2M3-Peel}}$ *and* $\mathcal{S}_N^{\text{S2M3-Peel}}$ *such that:*

$$\left\{\mathcal{S}_M^{\text{S2M3-Peel}}(1^\lambda, I_M, g_M(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_M^{\text{S2M3-Peel}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}},$$
$$\left\{\mathcal{S}_N^{\text{S2M3-Peel}}(1^\lambda, I_N, g_N(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_N^{\text{S2M3-Peel}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Proof. Since the view of P_M and P_N are exactly the same than for the S2M3 protocol, the proof is the same than Lemma 1.

Lemma 8. *Assume Paillier's cryptosystem is IND-CPA, then there exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{C}}^{\text{S2M3-Peel}}$ *such that:*

$$\left\{\mathcal{S}_{\mathcal{C}}^{\text{S2M3-Peel}}(1^\lambda, I_{\mathcal{C}}, g_{\mathcal{C}}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_{\mathcal{C}}^{\text{S2M3-Peel}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Proof. The only difference with the S2M3 protocol is that encrypted matrices are split according to the dynamic peeling during the deconstruction phase when it is required. Moreover, blocks multiplications are performed during the deconstruction phase and used during the combination phase to build the corresponding matrix. In the security point of view, these blocks multiplications do not give any information to the adversary. Hence we can consider only the blocks multiplication corresponding to the standard Strassen-Winograd matrix multiplication. Therefore, the security proof is the same than Lemma 2.

Lemma 9. *There exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{P}}^{\text{S2M3-Peel}}$ *such that:*

$$\left\{\mathcal{S}_{\mathcal{P}}^{\text{S2M3-Peel}}(1^\lambda, I_{\mathcal{P}}, g_{\mathcal{P}}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_{\mathcal{P}}^{\text{S2M3-Peel}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Proof. The proof is exactly the same than Lemma 3.

7 Conclusions and Future Work

We have presented SM3, a protocol to compute the Strassen-Winograd matrix multiplication using the MapReduce paradigm. Moreover, we extend this protocol to SM3-Pad and SM3-Peel protocols using respectively the dynamic padding and the dynamic peeling methods allowing square matrix multiplication of arbitrary dimension. We have also presented a secure approach for these three protocols denoted S2M3, S2M3-Pad, and S2M3-Peel satisfying 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's cryptosystem and on its homomorphic properties. We have compared our three no-secure protocols (resp. secure protocols) with the state-of-the-art MapReduce protocol of Leskovec et al. [22] (with the CRSP protocol proposed by Bultel et al. [4]) computing matrix multiplication, and shown that our protocols are 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

1. Amirbekyan, A., Estivill-Castro, V.: A New Efficient Privacy-Preserving Scalar Product Protocol. In: Proceedings of the 6th Australasian Data Mining Conference (AusDM). pp. 209–214 (2007)
2. Bellare, M., Boldyreva, A., Micali, S.: Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements. In: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT). pp. 259–274 (2000)
3. Blass, E., Pietro, R.D., Molva, R., Önen, M.: PRISM - Privacy-Preserving Search in MapReduce. In: Proceedings of the 12th International Symposium on Privacy Enhancing Technologies (PETS). pp. 180–200 (2012)
4. Bultel, X., Ciucanu, R., Giraud, M., Lafourcade, P.: Secure Matrix Multiplication with MapReduce. In: Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES). pp. 11:1–11:10 (2017)
5. Bultel, X., Ciucanu, R., Giraud, M., Lafourcade, P., Ye, L.: Secure Joins with MapReduce. In: Proceedings of the 11th International Symposium on Foundations and Practice of Security (FPS). pp. 78–94 (2018)
6. Chartrand, G.: Introductory Graph Theory. Dover Books on Mathematics, Dover Publications (2012)
7. Ciucanu, R., Giraud, M., Lafourcade, P., Ye, L.: Secure Grouping and Aggregation with MapReduce. In: Proceedings of the 15th International Joint Conference on e-Business and Telecommunications – Volume 2: SECRYPT (International Conference on Security and Cryptography). pp. 514–521 (2018)
8. Ciucanu, R., Giraud, M., Lafourcade, P., Ye, L.: Secure Intersection with MapReduce. In: Proceedings of the 16th International Joint Conference on e-Business and Telecommunications – Volume 2: SECRYPT (International Conference on Security and Cryptography). pp. 236–243 (2019)

9. Ciucanu, R., Giraud, M., Lafourcade, P., Ye, L.: Secure Strassen-Winograd Matrix Multiplication with MapReduce. In: Proceedings of the 16th International Joint Conference on e-Business and Telecommunications – Volume 2: SECRYPT (International Conference on Security and Cryptography). pp. 220–227 (2019)
10. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty Computation from Threshold Homomorphic Encryption. In: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT). pp. 280–299 (2001)
11. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI). pp. 137–150 (2004)
12. Derbeko, P., Dolev, S., Gudes, E., Sharma, S.: Security and Privacy Aspects in MapReduce on Clouds: A Survey. Computer Science Review **20**, 1–28 (2016)
13. Dolev, S., Li, Y., Sharma, S.: Private and Secure Secret Shared MapReduce (Extended Abstract). In: Proceedings of the 30th Annual Conference on Data and Applications Security and Privacy (DBSec). pp. 151–160 (2016)
14. Du, W., Atallah, M.J.: Privacy-Preserving Cooperative Statistical Analysis. In: Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC). pp. 102–110 (2001)
15. Dumas, J., Lafourcade, P., Orfila, J., Puys, M.: Dual Protocols for Private Multi-Party Matrix Multiplication and Trust Computations. Computers & Security **71**, 51–70 (2017)
16. Foundation, A.S.: Apache Hadoop (release 3.2.0). `https://hadoop.apache.org/` (Jan 2019)
17. Gall, F.L.: Powers of Tensors and Fast Matrix Multiplication. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC). pp. 296–303 (2014)
18. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2001)
19. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC). pp. 218–229 (1987)
20. Huss-Lederman, S., Jacobson, E.M., Johnson, J.R., Tsao, A., Turnbull, T.: Implementation of Strassen's Algorithm for Matrix Multiplication. In: Proceedings of the ACM/IEEE Conference on Supercomputing. p. 32 (1996)
21. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM **60**(6), 84–90 (2017)
22. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets, 2nd Ed. Cambridge University Press (2014)
23. Lindell, Y.: How to Simulate It - A Tutorial on the Simulation Proof Technique. In: Tutorials on the Foundations of Cryptography, pp. 277–346 (2017)
24. Ma, Q., Deng, P.: Secure Multi-party Protocols for Privacy Preserving Data Mining. In: Proceedings of the 3rd International Conference on Wireless Algorithms, Systems, and Applications (WASA). pp. 526–537 (2008)
25. Macedo, H.D.: Gaussian Elimination is Not Optimal. Journal of Logical and Algebraic Methods in Programming **85**(5), 999–1010 (2016)
26. Mayberry, T., Blass, E., Chan, A.H.: PIRMAP: Efficient Private Information Retrieval for MapReduce. In: Proceedings of the 17th International Conference on Financial Cryptography and Data Security (FC). pp. 371–385 (2013)

27. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT). pp. 223–238 (1999)
28. Shoshan, A., Zwick, U.: All Pairs Shortest Paths in Undirected Graphs with Integer Weights. In: Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS). pp. 605–615 (1999)
29. Strassen, V.: Gaussian Elimination Is Not Optimal. Numerische Mathematik **13**(4) (1969)
30. Wang, I., Shen, C., Zhan, J., Hsu, T., Liau, C., Wang, D.: Toward Empirical Aspects of Secure Scalar Product. IEEE Transactions on Systems, Man, and Cybernetics **39**(4), 440–447 (2009)
31. Yao, A.C.: Protocols for Secure Computations (Extended Abstract). In: Proceedings of the 23rd IEEE Annual Symposium on Foundations of Computer Science (FOCS). pp. 160–164 (1982)
32. Zwick, U.: All Pairs Shortest Paths in Weighted Directed Graphs Exact and Almost Exact Algorithms. In: Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS). pp. 310–319 (1998)

```
Reduce function:

Input: $(key, values)$
// $key$: $t_0 \cdots t_e$ such that $e \in [\![0,\ell]\!]$ and $t_z \in [\![0,7]\!]$ for $z \in [\![0,e]\!]$
// $values$: collection of $(\mathtt{M}, i, j, m^*_{i,j}, \delta)$ or $(\mathtt{N}, j, k, n^*_{j,k}, \delta)$ or $(\mathtt{P}, j, k, p^*_{a,z}, \delta_p)$
foreach $(\mathtt{P}, i, j, p^*_{a,z}, \delta_p) \in values$ do
    emit$(t_0 \cdots t_e, (\mathtt{P}, i, j, p^*_{a,z}, \delta_p))$
$M^* := (m^*_{i,j})_{(\mathtt{M},i,j,m_{i,j},\delta) \in values}$
$N^* := (n^*_{k,l})_{(\mathtt{N},k,l,n_{k,l},\delta) \in values}$
if $\delta \not\equiv 0 \pmod 2$ then
    $\begin{bmatrix} M' & M_{12} \\ M_{21} & M_{22} \end{bmatrix} := M^*$ ,   $\begin{bmatrix} N' & N_{12} \\ N_{21} & N_{22} \end{bmatrix} := N^*$ ,
    such that $\begin{cases} M' := (m_{ij})_{i,j \in [\![1,\delta-1]\!]} \\ M_{12} := (m_{ij})_{i \in [\![1,\delta-1]\!], j=\delta} \\ M_{21} := (m_{ij})_{i=\delta, j \in [\![1,\delta-1]\!]} \\ M_{22} := (m_{ij})_{i=\delta, j=\delta} \end{cases}$ , and $\begin{cases} N' := (n_{ij})_{i,j \in [\![1,\delta-1]\!]} \\ N_{12} := (n_{ij})_{i \in [\![1,\delta-1]\!], j=\delta} \\ N_{21} := (n_{ij})_{i=\delta, j \in [\![1,\delta-1]\!]} \\ N_{22} := (n_{ij})_{i=\delta, j=\delta} \end{cases}$
    $(q_{i,j})_{i,j \in [\![1,\delta]\!]} := \begin{bmatrix} M_{12}N_{21} & M'N_{12} + M_{12}N_{22} \\ M_{21}N' + M_{22}N_{21} & M_{21}N_{22} + M_{22}N_{21} \end{bmatrix}$
    $\delta' := (\delta - 1)/2$
    foreach $(a, z) \in [\![1,\delta]\!]^2$ do
        emit$(t_0 \cdots t_e, (\mathtt{P}, i, j, q^*_{a,z}, \delta'))$
else
    $M' := M^*$ ,   $N' := N^*$
    $\delta' := \delta/2$
// Split $M'$ and $N'$ into four quadrants of equal dimension
$\begin{bmatrix} M'_{11} & M'_{12} \\ M'_{21} & M'_{22} \end{bmatrix} := M'$ ,   $\begin{bmatrix} N'_{11} & N'_{12} \\ N'_{21} & N'_{22} \end{bmatrix} := N'$
$S_1 :=$ Paillier.Add$(pk, M'_{21}, M'_{22})$    $T_1 :=$ Paillier.Sub$(pk, N'_{12}, N'_{11})$
$S_2 :=$ Paillier.Sub$(pk, S_1, M'_{11})$      $T_2 :=$ Paillier.Sub$(pk, N'_{22}, T_1)$
$S_3 :=$ Paillier.Sub$(pk, M'_{11}, M'_{21})$    $T_3 :=$ Paillier.Sub$(pk, N'_{22}, N'_{12})$
$S_4 :=$ Paillier.Sub$(pk, M'_{12}, S_2)$      $T_4 :=$ Paillier.Sub$(pk, T_2, N'_{21})$
$L := \big[[M'_{11}, N'_{11}], [M'_{12}, N'_{21}], [S_4, N'_{22}], [M'_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$
if $\delta > 2$ then
    $\ell' := \log_2(d/\delta')$
    foreach $u \in [\![0,6]\!]$ do
        $(a'_{ij})_{i,j \in [\![1,\delta']\!]} := L[u][0]$
        $(b'_{jk})_{j,k \in [\![1,\delta']\!]} := L[u][1]$
        foreach $(v, w) \in [\![1,\delta']\!]^2$ do
            emit$_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t \| u, (\mathtt{M}, v, w, a'_{ij}, \delta'))$
            emit$_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t \| u, (\mathtt{N}, v, w, b'_{ij}, \delta'))$
else
    foreach $u \in [\![0,6]\!]$ do
        emit$_{\mathcal{D}_\ell \to \mathcal{C}_1}(t, (u, 1, 1,$ Paillier.Sub$(L[u][0], L[u][1]), 1))$
```

Fig. 17. Reduce function for the deconstruction phase of the S2M3-Peel protocol.

```
Reduce function:
Input: $(key, values)$
// $key$: $t_0 \cdots t_e$ such that $e \in [\![0,\ell]\!]$ and $t_z \in [\![0,7]\!]$ for $z \in [\![0,e]\!]$
// $values$: collection of $(u,v,w,r^*_{v,w},\delta)$ or $(\mathtt{P},a,z,p^*_{a,z},\delta_p)$ such that $u \in [\![1,7]\!]$
if $\exists\big((\mathtt{P},a,z,p^*_{a,z},\delta_p) \wedge (u,v,w,r^*_{v,w},\delta)\big) \in values$ then
    $P := (p^*_{a,z})_{(\mathtt{P},a,z,p^*_{a,z},\delta_p)\in values}$
    foreach $u \in [\![1,7]\!]$ do
        $R_u := (r^*_{v,w})_{(u,v,w,r^*_{v,w},\delta)\in values}$
    $P_1 := \mathsf{Paillier.Add}(pk, R_0, R_1)$        $P_5 := \mathsf{Paillier.Add}(pk, P_4, R_2)$
    $P_2 := \mathsf{Paillier.Add}(pk, R_0, R_5)$        $P_6 := \mathsf{Paillier.Sub}(pk, P_3, R_3)$
    $P_3 := \mathsf{Paillier.Add}(pk, P_2, R_6)$        $P_7 := \mathsf{Paillier.Sub}(pk, P_3, R_4)$
    $P_4 := \mathsf{Paillier.Add}(pk, P_2, R_4)$
    $(\rho_{i,j})_{i,j\in[\![1,2\cdot\delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$
    $\rho'_{i,j} := \begin{cases} \rho_{i,j}+p^*_{i,j} & \text{if } \; i,j \in [\![1,2\cdot\delta]\!] \\ p^*_{i,j} & \text{if } \; \max_{i,j\in[\![1,2\cdot\delta+1]\!]}(i,j) = 2\cdot\delta+1 \end{cases}$
    $\delta' := 2\cdot\delta+1$
    if $\delta' < d$ then
        foreach $(i,j) \in [\![1,\delta']\!]^2$ do
            $\mathtt{emit}_{\mathcal{C}\to\mathcal{C}}(t_0\cdots t_{e-1},(t_e,i,j,\rho'_{i,j},\delta'))$
    else
        foreach $(i,j) \in [\![1,d]\!]^2$ do
            $\mathtt{emit}_{\mathcal{C}\to\mathcal{C}}((i,j),\rho'_{i,j})$
else if $\{\exists(\mathtt{P},a,z,p^*_{a,z},\delta_p) \in values\} \wedge \{\nexists(u,v,w,r^*_{v,w},\delta) \in values\}$ then
    foreach $(\mathtt{P},a,z,p^*_{a,z},\delta_p) \in values$ do
        $\mathtt{emit}(t_0\cdots t_e,(\mathtt{P},a,z,p^*_{a,z},\delta_p))$
else if $\{\nexists(\mathtt{P},a,z,p^*_{a,z},\delta_p) \in values\} \wedge \{\exists(u,v,w,r^*_{v,w},\delta) \in values\}$ then
    foreach $u \in [\![1,7]\!]$ do
        $R_u := (r^*_{v,w})_{(u,v,w,r^*_{v,w},\delta)\in values}$
    $P_1 := \mathsf{Paillier.Add}(pk, R_1, R_2)$        $P_5 := \mathsf{Paillier.Add}(pk, P_4, R_3)$
    $P_2 := \mathsf{Paillier.Add}(pk, R_1, R_6)$        $P_6 := \mathsf{Paillier.Sub}(pk, P_3, R_4)$
    $P_3 := \mathsf{Paillier.Add}(pk, P_2, R_7)$        $P_7 := \mathsf{Paillier.Sub}(pk, P_3, R_5)$
    $P_4 := \mathsf{Paillier.Add}(pk, P_2, R_5)$
    $(\rho_{i,j})_{i,j\in[\![1,2\cdot\delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$
    $\delta' := 2\cdot\delta$
    if $\delta' < d$ then
        foreach $(i,j) \in [\![1,\delta']\!]^2$ do
            $\mathtt{emit}_{\mathcal{C}\to\mathcal{C}}(t_0\cdots t_{e-1},(t_e,i,j,\rho_{i,j},\delta'))$
    else
        foreach $(i,j) \in [\![1,d]\!]^2$ do
            $\mathtt{emit}_{\mathcal{C}\to\mathcal{C}}((i,j),\rho_{i,j})$
```

Fig. 18. Reduce function for the combination phase of the S2M3-Peel protocol.

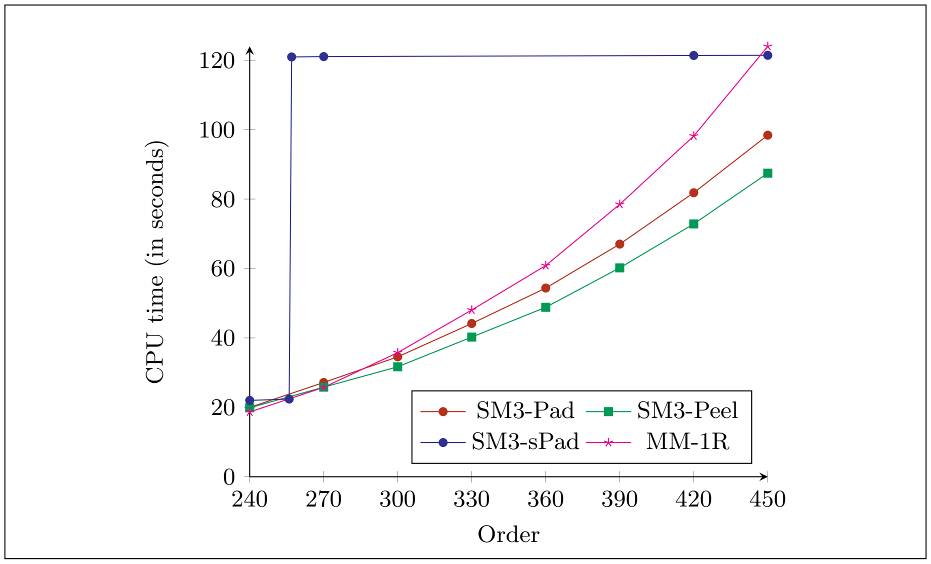

Fig. 19. CPU time vs order of matrices for the state-of-the-art MM-1R protocol using one MapReduce round [22] and for our SM3 protocols using static and dynamic padding methods, and dynamic peeling method.

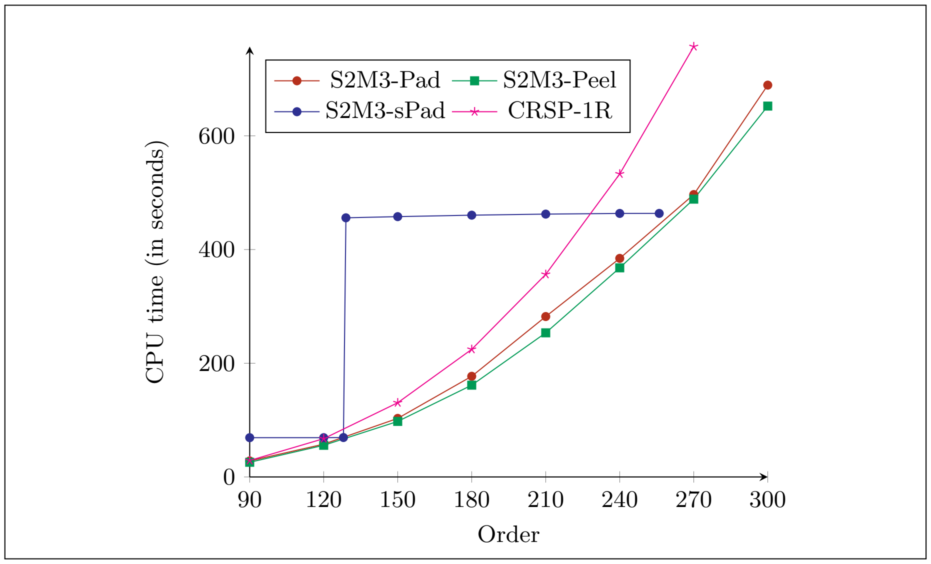

Fig. 20. CPU time vs order of matrices for the CRSP-1R protocol [4] and for our S2M3 protocols using static and dynamic padding methods, and dynamic peeling method.

```
Function: SW-Deconstruction
SW-Deconstruction($A, B, U, \mathsf{view}, pk$):
$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} := A\,, \quad \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} := B$
$S_1 :=$ Paillier.Add($pk, A_{21}, A_{22}$)     $T_1 :=$ Paillier.Sub($pk, B_{12}, B_{11}$)
$S_2 :=$ Paillier.Sub($pk, S_1, A_{11}$)       $T_2 :=$ Paillier.Sub($pk, B_{22}, T_1$)
$S_3 :=$ Paillier.Sub($pk, A_{11}, A_{21}$)     $T_3 :=$ Paillier.Sub($pk, B_{22}, B_{12}$)
$S_4 :=$ Paillier.Sub($pk, A_{12}, S_2$)       $T_4 :=$ Paillier.Sub($pk, T_2, B_{21}$)
$L := \big[[A_{11}, B_{11}], [A_{12}, B_{21}], [S_4, B_{22}], [A_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$
foreach $[A_0, B_0] \in L$ do
    if $dim(A_0) = dim(B_0) = 1$ then
        $U := U \cup \{[A_0, B_0]\}$
    else
        $\mathsf{view} := \mathsf{view} \cup \{[A_0, B_0]\}$
        SW-Deconstruction($A_0, B_0, U, \mathsf{view}, pk$)
```

Fig. 21. Function SW-Deconstruction for simulator $\mathcal{S}_{\mathcal{C}}^{\mathrm{S2M3}}$ presented in Fig. 23.

```
Function: SW-Combination
SW-Combination($R_1, \ldots, R_7, \mathsf{view}, pk$):
$P_1 :=$ Paillier.Add($pk, R_1, R_2$)      $P_5 :=$ Paillier.Add($pk, P_4, R_3$)
$P_2 :=$ Paillier.Add($pk, R_1, R_6$)      $P_6 :=$ Paillier.Add($pk, P_3, R_4$)
$P_3 :=$ Paillier.Add($pk, P_2, R_7$)      $P_7 :=$ Paillier.Add($pk, P_3, R_5$)
$P_4 :=$ Paillier.Add($pk, P_2, R_5$)
$P := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$
$\mathsf{view} := \mathsf{view} \cup \{P\}$
return $(P, \mathsf{view})$
```

Fig. 22. Function SW-Combination for simulator $\mathcal{S}_{\mathcal{C}}^{\mathrm{S2M3}}$ presented in Fig. 23.

```
Simulator: $\mathcal{S}_{\mathcal{C}}^{\mathrm{S2M3}}(1^\lambda, pk, d)$

$U := \emptyset$
$\mathsf{view} := \emptyset$
foreach $(i,j) \in [\![1,d]\!]^2$ do
    $(\alpha_{i,j}, \beta_{i,j}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
$M^* := \big(\mathcal{E}(pk, \alpha_{i,j})\big)_{i,j \in [\![1,d]\!]}$ ,    $N^* := \big(\mathcal{E}(pk, \beta_{i,j})\big)_{i,j \in [\![1,d]\!]}$
$\mathsf{SW\text{-}Deconstruction}(M^*, N^*, U, \mathsf{view}, pk)$
foreach $i \in [\![1, 7^\ell]\!]$ do
    $(r_i, s_i, t_i) \xleftarrow{\$} (\mathbb{Z}_n)^3$
    $x_i := \mathcal{E}(pk, r_i)$
    $y_i := \mathcal{E}(pk, s_i)$
    $R_i^{(\ell)} := \mathcal{E}(pk, t_i)$
    $\mathsf{view} := \mathsf{view} \cup \{R_i^{(\ell)}\}$
foreach $k \in [\![\ell, 1]\!]$ do
    foreach $j \in [\![1, 7^{k-1}]\!]$ do
        $(R_j^{(k-1)}, \mathsf{view}') := \mathsf{SW\text{-}Combination}(R_{7\cdot j-6}^{(k)}, \ldots, R_{7\cdot j}^{(k)})$
        $\mathsf{view} := \mathsf{view} \cup \mathsf{view}'$
return view
```

Fig. 23. Simulator $\mathcal{S}_{\mathcal{C}}^{\mathrm{S2M3}}$ for the proof of Lemma 2.

```
Adversary: $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$

$U := \emptyset$
view $:= \emptyset$
foreach $i \in [\![1, d]\!]$ do
    foreach $j \in [\![1, d]\!]$ do
        $(m_{i,j}, n_{i,j}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
        $(\alpha_{i,j}, \beta_{i,j}) \xleftarrow{\$} (\mathbb{Z}_n)^2$

$M^* := \big(\mathcal{E}(pk, \mathsf{LoR}_b(m_{i,j}, \alpha_{i,j}))\big)_{i,j\in[\![1,d]\!]}$
$N^* := \big(\mathcal{E}(pk, \mathsf{LoR}_b(n_{i,j}, \beta_{i,j}))\big)_{i,j\in[\![1,d]\!]}$

SW-Deconstruction$(M^*, N^*, U, \mathsf{view}, pk)$

foreach $i \in [\![1, 7^\ell]\!]$ do
    $(r_i, s_i, t_i) \xleftarrow{\$} (\mathbb{Z}_n)^3$
    $x_i := \mathcal{E}(pk, r_i)$
    $y_i := \mathcal{E}(pk, s_i)$
    $R_i^{(\ell)} := \mathcal{E}(pk, \mathsf{LoR}_b(U[i-1][0] \cdot U[i-1][1], t_i))$
    view $:=$ view $\cup \{R_i^{(\ell)}\}$

foreach $k \in [\![\ell, 1]\!]$ do
    foreach $j \in [\![1, 7^{k-1}]\!]$ do
        $(R_j^{(k-1)}, \mathsf{view}') :=$ SW-Combination$(R_{7\cdot j-6}^{(k)}, \ldots, R_{7\cdot j}^{(k)})$
        view $:=$ view $\cup$ view$'$
return view
```

Fig. 24. Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 2.

```
Simulator: $\mathcal{S}_{\mathcal{P}}^{\mathrm{S2M3}}(1^\lambda, (pk, sk), P)$

foreach $i \in [\![1, 7^\ell]\!]$ do
    $(r_i, s_i) \xleftarrow{\$} (\mathbb{Z}_n)^2$
    $x_i := \mathcal{E}(pk, r_i)$
    $y_i := \mathcal{E}(pk, s_i)$
    $z_i := \mathcal{E}(pk, r_i \cdot s_i)$
$P^* := (\mathcal{E}(pk, p_{i,j}))_{i,j\in[\![1,d]\!]}$
view $:= \big(\{(x_i, y_i), z_i\}_{i\in[\![1,7^\ell]\!]}, P^*\big)$
return view
```

Fig. 25. Simulator $\mathcal{S}_{\mathcal{P}}^{\mathrm{S2M3}}$ for the proof of Lemma 3.