Privacy-preserving Algorithm for Outsourcing Matrix Multiplication Based on Strassen's Algorithm

by

Moeedreza Mostafavi Toroqi

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS, INFORMATION TECHNOLOGY
M.A.Sc.

MONTREAL, "FEBRUARY 4TH, 2020"

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

M. Jean-Marc Robert, Memorandum Supervisor
Département de génie logiciel et des TI, ÉTS

M. Patrick Cardinal, President of the Board of Examiners
Département de génie logiciel et des TI, ÉTS

M. Chamseddine Talhi, External Examiner
Département de génie logiciel et des TI, ÉTS

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON "JANUARY 10TH, 2020"

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisor, Prof. Jean-Marc Robert who means more than an university adviser for me during my master's. The door of his office was always open whenever I faced a problem in my thesis and his helpful and valuable supports conduct me to complete this thesis.

My parents deserve special thanks for their continued support and encouragement during all my life. I would especially like to thank my wife, Sadina has been extremely supportive of me throughout this entire process and has made countless sacrifices to help me get to this point. Also my lovely son, Liam for providing cheerful atmosphere at home.

Without such a team behind me, I doubt that I would be in this place today.

Algorithme distribué efficace pour la multiplication matricielle protégeant la confidentialité des données

Moeedreza Mostafavi Toroqi

RÉSUMÉ

La multiplication de matrices est un outil mathématique fondamental et un problème de base dans de nombreux domaines tels que l'ingénierie et l'informatique. Fréquemment, les clients dont les ressources sont limitées ne peuvent supporter le coût de ce type de calcul et doivent externaliser les informations et les calculs vers un système infonuagique puissant et efficace. Dans cet échange de données, la sécurité et la confidentialité deviennent un problème important en raison de la valeur de certaines données sensibles telles que des informations personnelles privées ou des données industrielles. Les serveurs infonuagiques non fiables représentent toujours un risque important pour les clients et augmentent le risque de fuite d'informations lors de l'externalisation des données. Il existe de nombreux algorithmes sécurisés avec des méthodes de chiffrement de données qui sont déjà utilisés dans l'industrie, mais en raison de nombreux modèles de système et de traitement et de l'importance du niveau de sécurité, de nouveaux algorithmes apparaissent toujours pour améliorer les schémas existants.

Dans ce mémoire, nous proposons quatre algorithmes préservant la confidentialité des données basés sur le schéma de Strassen afin de fournir le résultat souhaité aux clients qui ne sont pas en mesure d'effectuer ce type de calcul par manque de ressources.

Mots-clés: La Multiplication de Matrices, Le Schéma de Strassen

Privacy-preserving Algorithm for Outsourcing Matrix Multiplication Based on Strassen's Algorithm

Moeedreza Mostafavi Toroqi

ABSTRACT

Matrix Multiplication is a fundamental mathematical tool and basic problem with various applications in many domains such as data mining and data analyzing. Most of the time, limited-resource clients cannot afford the cost of heavy computations and needs to outsource the information and computations to a powerful and efficient cloud system. In this data exchanging, security and privacy concerning become a significant issue due to the value of some sensitive data such as private personal information or industrial data. The untrusted cloud servers beside malicious adversaries are always a kind of hazard for the clients and increase the risk of information leakage in data outsourcing. There exist many secure algorithms with creative data masking methods which are already usable in the industry, but due to numerous system and treat models, and the importance of the level of the security, new algorithms always emerge to improve the existing schemes.

In this thesis, we propose four privacy-preserving algorithms based on Strassen's scheme to provide the desirable result of matrix multiplication for the clients who are not able to process the regular matrix multiplication algorithms because of lake of powerful resources.

Keywords: Matrix Multiplication, Private/Secure, Outsource Computation, Strassen's Algorithm

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF ALGORITHMS

INTRODUCTION

Due to the availability of increasing number of high technology devices generating large amount of personal data, highly intensive computational requirements and storage become crucial. Then, data outsourcing emerged as a universal solution to decrease the computation cost on the client side and transfer it to external powerful cloud systems.

Matrix multiplication as an important tool for data analysis requires important resources which are too costly in terms of time and space for data owners. But data outsourcing, on the other hand, brings many security and privacy concerns for the clients. Usually such matrices contain sensitive information (i.e. private information of banks' clients, insurance holders or industrial manufacture, etc.) Thus, such processing should be protected from leaking on a public cloud side.

Many research has been done in the recent years to improve the privacy of the outsourcing matrix multiplication while the malicious adversaries are developing new methods and techniques as well. The clients as the data owners try to hide all the matrix elements using some encryption process before sending them to the cloud systems since the communication channel, or the cloud systems themselves are exposed to adversaries attacks. On the other hand, the cost of masking input data and deriving the desired result from receiving values should not be computationally high for the clients. It means that every proposed schemes for outsourcing matrix multiplication must be appropriate for specific goals. Some data owners prefer to stick to the highest security requirements for their sensitive information while others may trust their cloud systems in order to reduce their costs of encryption and decryption of their data.

In this thesis, we propose four privacy-preserving algorithms based on Strassen's scheme to provide the desirable result of matrix multiplication for the clients who are not able to process the regular matrix multiplication algorithms because of lake of powerful resources.

CHAPTER 1

PROBLEM STATEMENT

In this section, the family of problems tackled in this thesis is presented. In the following section, the family of problems is formally defined. In the second section, the actors involved in the problem-solving process and the security model used to evaluate the proposed solutions are described.

1.1 Problem definition

As mentioned in the introduction, the family of problems considered in this thesis consists in the secure computation of the multiplication of two matrices. In this context, the secure property refers to the protection of the privacy of the information, when possible.

Consider the following two compatible matrices:

- the matrix $A = [a_{i,j}]$, for $1 \leq i \leq r$ and $1 \leq j \leq s$;
- the matrix $B = [b_{i,j}]$, for $1 \leq i \leq s$ and $1 \leq j \leq t$.

The objective is to obtain the resulting matrix $C = [c_{i,j}]$, for $1 \leq i \leq r$ and $1 \leq j \leq t$ defined as follows

$$c_{i,j} = \sum_{k=1}^{s} a_{i,k} \cdot b_{k,j}. \tag{1.1}$$

In the degenerate case, the problem can simply be the dot product of two vectors $A = (a_1, \cdots, a_s)$ and $B^t = (b_1, \cdots, b_s)$.

Obviously, this family of problems should be defined in distributed setup to be interesting. In the first step, the elements of the matrices may be distributed among different actors. For

example, the matrices A and B can be owned by two different parties $\mathscr{A}$ and $\mathscr{B}$, respectively. Even worst, the matrix B can be shared by $s \times t$ parties. Ultimately, the resulting matrix C should only be obtained by the identified party. In the second step, the computation may be done by some unreliable third parties.

In this thesis, the goal is to propose a solution for a powerful outsourcing entity to do a matrix multiplication, using an efficient divide-and-conquer algorithm (Strassen's algorithm) when the matrices are distributed among different independent data owners. The goal of this solution is calculating the product of these matrices while the privacy of the data owners is protected. For example, when a financial institute ask his clients to contribute in a survey, the most significant issue for the clients is the protection of their information. On the other hand, the financial institute needs the final result in a protected form as well.

1.2 Security Model

The security model defines the behaviour of the actors involved in the matrix multiplication process. It is used to demonstrate the security level and possible attacks on the proposed solutions. According to the actors' behaviours and contributions to the matrix multiplication process, different states are assigned to the actors. First of all, we define three actors in the process:

- ***Data Owners***: Data owners are the participants who have one or more elements of matrices. They just know their own data, without any information about other elements even in the same matrix. Moreover, they know the public key of the third parties and the result owner, if such keys are needed in a proposed solution.
- ***Result Owner***: This single owner wants to obtain the resulting information of the matrix multiplication process. However, he shall not obtain any information on the primary matrices, even if he knows the algorithms and the protocols used by the different parties.

- ***Third Parties***: The third parties do not own any element of matrices, neither the result of the computation. They are intermediate nodes who receive some masked data, encrypted or randomized, process, and send them to the targeted actors. They should not get any detail about the product matrix or the primary matrices. Depending on the protocols, there may be one or more third parties.

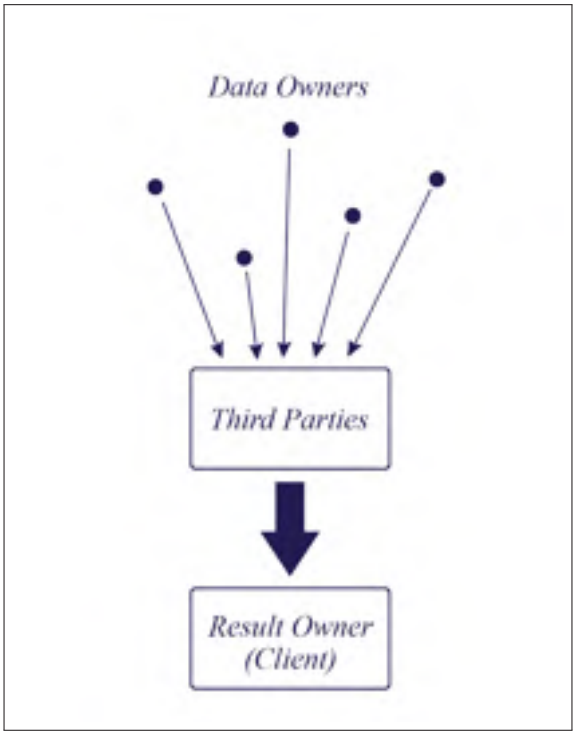


Figure 1.1 Communication Flow Between Parties

Ideally, the communication flow between these parties is one way. The data owners send their masked values to the third parties which send them to the result owner after processing. There is no interactive communication between parties.

Depending on the behaviour of the participants, they are categorized in different groups: *honest*, *semi-honest* or *malicious* Hazay & Lindell (2010)

- ***An honest participant:*** Such a participant follows trustfully a protocol with no extra activity to find any other information. This behaviour guaranties the correctness of the result.

- ***A semi-honest participant:*** Such a participant follows a protocol and performs his assigned duties, but may also tries to find out some extra information that he would not know otherwise. These semi-honest parties do alter the result. So the result should be correct and valid, but some private information may leak in the process. Sometimes, such a participant is referred to as *honest-but-curious*.

- ***A malicious participant:*** Such a participant does not necessarily follows the protocol, may send bogus information to other participants, or may try to impersonate another participant. Consequently, the result of process is not reliable and some private information may leak in the process.

The following figure characterizes the different possibilities for the secure matrix multiplication protocols presented in this thesis. The result owner does not contribute to the computations and the protocol exchanges. This actor just receives the end result of the process. Therefore, he should not have to be characterized.

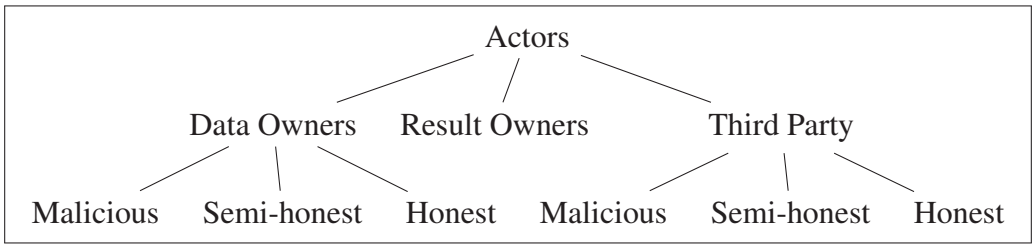


Figure 1.2 Different Characterization for the Participants behavior

According to the parties cooperating in the execution of an algorithm, they may collude to get private information or work alone. In a collusion, two or more participants may work together, reveal their private information to each other, or even to the other participants. Their goal is to get some extra private information, which they could not obtain otherwise. They may send

bogus values to victims, or process the receiving data on their own to find out some private information. The collusion may occur among semi-honest participants or malicious parties.

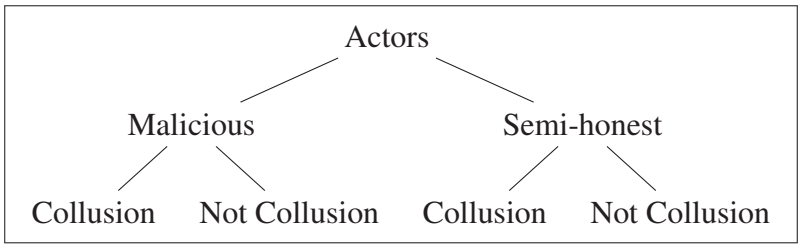


Figure 1.3 Collusion Probability in Different Participants

CHAPTER 2

LITERATURE REVIEW

2.1 Cryptographic Tools

In order to transfer over a public channel data securely and privately, some cryptographic systems have to be used. In this section, the basic cryptographic primitives are presented. The objective is to ensure that the description of the protocols in this thesis are self-contained. This is particularly important for the homomorphic properties of the cryptographic systems on which these protocols rely upon.

In a public/private key cryptographic system, each party has a unique public key *pk* used by the other parties to encrypt messages intended for this party. On the other hand, this party owns the corresponding private key *sk* used to decrypt receiving messages. Let m be a plain-text message and c be the corresponding ciphertext message. The public encryption algorithm E_{pk} and the corresponding private decryption algorithm D_{sk} are such that, for any message m, :

- $E_{pk}(m) = c$
- $D_{sk}(c) = m$

An homomorphic encryption is a form of encryption that allows computations on ciphertexts to correspond to some desired computations on plaintexts. For example, the multiplication of two ciphertexts may give the sum of the corresponding plaintexts once decrypted.

Generally speaking, two different operations are usually required: (i) the additions of two plaintexts and (ii) the multiplications of two plaintext. If a cryptographic system provides both unrestricted operations, it is said to be fully homomorphic. There are many variants of fully homomorphic cryptographic systems but they are very costly and are not appropriate to solve the problems considered in this thesis Gentry (2009), Kaosar *et al.* (2012).

To obtain better performances, some cryptographic systems support unrestrictedly only one of the two classical operations (e. g., the addition of the plaintexts). For the other operation, they support only some limited cases. For example, the multiplication of an encrypted plaintext with an unencrypted one. Or, only few multiplications of the encrypted plaintexts.

Homomorphic Properties

For any proper messages m_1 and m_2,

1. $E_{pk}(m_1) \times E_{pk}(m_2) \equiv E_{pk}(m_1 + m_2) \mod n^2$
2. $E_{pk}(m_1)^{m_2} \equiv E_{pk}(m_1 \times m_2)$

There exist several semi-homomorphic cryptographic systems. Here we propose to use the specific encryption scheme proposed by Paillier (1999) to illustrate the main properties of these cryptographic systems.

Paillier Crypto-system

The Paillier crypto-system is based on the problem of computing n^{th} residue classes, which is believed to be computationally difficult. Here we recall this scheme in details:

Key generation

1. Choose two large random prime numbers p and q, independently, such that the greatest common divisor $gcd(pq,(p-1)(q-1)) = 1$. If the primes have the same length, this property is always fulfilled.
2. Compute $n = pq$ and $\lambda = lcm(p-1,q-1)$, knowing that $lcm(\cdot)$ is the least common multiple function.
3. Select a random integer g where $g \in Z^*_{n^2}$.
4. Compute $\mu = (L(g^\lambda \ mod \ n^2))^{-1}$, knowing that $L(x) = \frac{x-1}{n}$

The public encryption key is (n, g) and the corresponding private decryption key is (λ, μ).

Encryption

To encrypt a plaintext message m, we should pick a random number r such that $0 < r < n$ and $r \in Z^*_{n^2}$, and compute the ciphertext message as follows: $c = g^m \times r^n \ mod \ n^2$.

Decryption:

To decrypt a ciphertext message c, we should retrieve the corresponding plaintext massage as follows: $m = L(c^\lambda \ mod \ n^2) \times \mu \ mod \ n$.

The most interesting property of Paillier scheme is the fact that the encryption algorithm is *additive homomorphic* (or semi-homomorphic). This means that knowing the encryption of two messages $E_{pk}(m_1)$ and $E_{pk}(m_2)$, anyone can compute the encryption of the sum of the two messages $E_{pk}(m_1 + m_2)$, without any further information on the cryptographic system properties or the underlying messages. This can be simply achieved by multiplying both ciphertexts. Furthermore, the multiplication of a message $E_{pk}(m_1)$ with a cleartext message m_2 can also be done efficiently. In this specific case, the ciphertext $E_{pk}(m_1 \cdot m_2)$ can simply be obtained by exponentiating the value of $E_{pk}(m_1)$ by m_2.

The other semi-homomorphic system which is frequently used in secure computation algorithms is Naccache-stern cryptosystemNaccache & Stern (1998).

Negative numbers

Let us assume that the messages represent integers. It is then possible to manipulate negative integers. By convention, let us assume that if $m \in Z^*_{n^2}$ is smaller than $\frac{n^2}{2}$, m is positive integer. On the other hand, if $m \in Z^*_{n^2}$ is greater than $\frac{n^2}{2}$, m represents the negative number $m - n^2$.

With such a convention, if $a, r \in Z^*_{n^2}$ and $a, r \leq \frac{n^2}{2}$ then:

$$\frac{E_{pk}(a+r)}{E_{pk}(r)} = E_{pk}(a+r) \times E_{pk}(-r) = E_{pk}(a).$$

Other semi-homomorphic cryptographic schemes have been presented: Benaloh (1994), Naccache & Stern (1998) and Okamoto & Uchiyama (1998).All these systems can be used interchangeably in the different protocols presented in this thesis.

2.2 Vector Multiplication

In this classical problem, there are two vectors $\vec{U} = [u_1, u_2, ..., u_t]$ and $\vec{V} = [v_1, v_2, ..., v_t]$ of the same length. The goal is to compute the dot product of these vectors. No matter how the elements are distributed among the different parties, the corresponding elements of two vectors should be multiplied. The result is the sum of these products:

$$\vec{U} \cdot \vec{V} = \sum_{i=1}^{t} u_i v_i.$$

This can be seen as a basic tool for the classical matrix multiplication.

Secure/private Vector Multiplication

This problem is discussed in the different system models and element distributions. The elements of initial vectors may be distributed in different parties, or one party has a complete vector. Moreover, the presence or absence or a third party affects the algorithm. All possible set-ups are illustrated below.

2.2.1 Two Data Owners

First of all, assume there are only two parties. Each one of them owns a complete vector of length t, and their goal is to compute the dot product of these vectors securely, without the assistance of third party. One of these two parties, say $\mathscr{A}$, is the result owner as well. Therefore,

- the party $\mathscr{A}$ owns the vector $\vec{A} = [a_1, a_2, ..., a_t]$;
- the party $\mathscr{B}$ owns the vector $\vec{B} = [b_1, b_2, ..., b_t]$.

In the first step, $\mathscr{A}$ encrypts all the elements individually with his public key and send them to $\mathscr{B}$. Any semi-homomorphic encryption scheme can be used in this algorithm. Then, $\mathscr{B}$ raises the received data to the power of the corresponding elements, multiplies the intermediate results, and sends the result back to $\mathscr{A}$. Here, $\mathscr{A}$ decrypts the received value with his private key and gets the end result corresponding to the dot product of the two original vectors:

1. $\mathscr{A} : E_{\mathscr{A}}(a_1), E_{\mathscr{A}}(a_2), ..., E_{\mathscr{A}}(a_t) \Longrightarrow \mathscr{B}$
2. $\mathscr{B} : E_{\mathscr{A}}(a_1)^{b_1} \times E_{\mathscr{A}}(a_2)^{b_2} \times \ldots \times E_{\mathscr{A}}(a_t)^{b_t} = E_{\mathscr{A}}(a_1b_1 + a_2b_2 + \ldots + a_tb_t) \Longrightarrow \mathscr{A}$
3. $\mathscr{A} : D_{\mathscr{A}}[E_{\mathscr{A}}(a_1b_1 + a_2b_2 + \ldots + a_tb_t)] = \sum_{i=1}^{t} a_ib_i$

The computation cost of this algorithm is $t(C_{enc} + C_{exp}) + (t-1)C_{mul} + C_{dec}$. There are just two parties, who are also data owners in this approach. Here, no malicious collusion is possible. This algorithm is secure against semi-honest adversaries. Nevertheless, if $\mathscr{A}$ sends n different vectors to $\mathscr{B}$ and gets the corresponding results, he should obtain t linear independent equations, and would be able to find the vector $\vec{B}$. Technically, $\mathscr{B}$ should refuse to answer to more than one dot product request nevertheless. Moreover, if $\mathscr{A}$ send specific vectors to $\mathscr{B}$, he may get some information but not all:

- if $\mathscr{A}$ sends $\vec{A} = [0, 0, ..., a_i = 1, ..., 0]$, he would learn b_i;
- if $\mathscr{A}$ sends $\vec{A} = [1, 1, ..., 1]$, he would learn $\sum_{i=1}^{t} b_i$.

The other approach is in presence of a third party, which would also be the result owner. Here, there are two parties $\mathscr{A}$ and $\mathscr{B}$, as before, and a result owner $\mathscr{P}$ who wants to obtain the dot product of the two vectors. The goal is to compute the dot product, without revealing any information to the data owners and even to the result owner.

First of all, $\mathscr{A}$ picks some random numbers r_i, adds them to his elements, encrypts the results individually with $\mathscr{P}$ public key, and sends them to $\mathscr{B}$. Then, he encrypts the random numbers r_n with $\mathscr{B}$ public key and sends them to $\mathscr{B}$. Thus, $\mathscr{A}$ computes the following values for $\mathscr{B}$:

1. $E_{\mathscr{P}}(a_1+r_1), E_{\mathscr{P}}(a_2+r_2), ..., E_{\mathscr{P}}(a_t+r_t)$

2. $E_{\mathscr{B}}(r_1), E_B(r_2), ..., E_{\mathscr{B}}(r_t)$

Here, $\mathscr{B}$ decrypts the random numbers, reencrypts them with $\mathscr{P}$ public key, and computes the following equations and sends the result to the result owner $\mathscr{P}$:

1. $E_{\mathscr{P}}(r_1), E_{\mathscr{P}}(r_2), \ldots, E_{\mathscr{P}}(r_n)$

2. $E_{\mathscr{P}}(-r_1), E_{\mathscr{P}}P(-r_2), \ldots, E_{\mathscr{P}}(-r_n)$

3. $\{E_{\mathscr{P}}(a_1+r_1) \times E_{\mathscr{P}}(-r_1)\}^{b_1} \times \ldots \times \{E_{\mathscr{P}}(a_t+r_t) \times E_{\mathscr{P}}(-r_t)\}^{b_t} = E_{\mathscr{P}}\left(\sum_{i=1}^{t} a_i b_i\right)$

The result owner $\mathscr{P}$, decrypts the received value to get the dot product. This algorithm is secure against semi-honest attackers but in case of collusion between $\mathscr{B}$ and $\mathscr{P}$, they get the elements of vector $\vec{A}$.

Next we discuss an algorithm in which both participants are semi-honest Ioannidis *et al.* (2002). Assume that one of the parties, say $\mathscr{A}$, is interested in the dot product $\vec{A} \cdot \vec{B}$. The other party is assumed to be cooperative as long as he is assured that $\mathscr{A}$ would not get any other information while following the protocol - even if he is only semi-honest. Unfortunately, no third party can be used in this case. Nevertheless, let assume that a public $s \times s$ random matrix is publicly known to both participants.

In the first step, the participant $\mathscr{B}$ computes (or defines) the following values:

- a $s \times s$ random matrix Q - for some security parameter $s \geq 2$
- a $s \times (t+1)$ random matrix X such that a random row $1 \leq r \leq s$ of the matrix is composed of the vector $\vec{B}$ - note that the extra component $t+1$ of this row is simply set to 1
- a value $b = \sum_{i=1}^{s} Q_{i,r}$
- a vector of dimension $t+1$ $c = \sum_{i=1, i \neq r}^{s} Xi, \cdot \times \sum_{j=1}^{s} Q_{j,i}$

- a random vector of dimension $t+1$ f
- three random values R_1, R_2, and R_3

Once these values have been computed, $\mathscr{B}$ sends to $\mathscr{A}$ the following values:

1. the $s \times (t+1)$ matrix QX - encoding the vector $\vec{B}$ in the row r of X
2. the $(t+1)$-dimensional vector $c' = c + (R_1 \times R_2) \cdot f$
3. the $(t+1)$-dimensional vector $g = f + (R_1 \times R_3) \cdot f$

Upon the reception of all these values, $\mathscr{A}$ first computes the following values:

- the s-dimensional vector $y = QXv$ - where v is the extended vector $\vec{A}$ where the extra component $t+1$ of this vector is simply set to the random value α
- the value $z = sum_{i=1}^{s} y_i$

Once these values have been computed, $\mathscr{A}$ sends to $\mathscr{B}$ the following values:

1. the value $a = z - c' \cdot v$
2. the value $h = g \cdot v$

Finally, $\mathscr{B}$ computes $\beta = \frac{a + h\frac{R_1}{R_2}}{b}$ and returns this value to $\mathscr{A}$. From this value, the dot product $\vec{A} \cdot \vec{B}$ is simply equal to $\beta - \alpha$.

The communication flows between both participants are illustrated in Figure 2.1.

2.2.2 Multi Data Owners

Let us consider a different problem. One party $\mathscr{A}$ has an entire vector $\vec{A}$, and the second vector is distributed among t different parties $\mathscr{B}_i$, in the worst case. Each of these parties would own

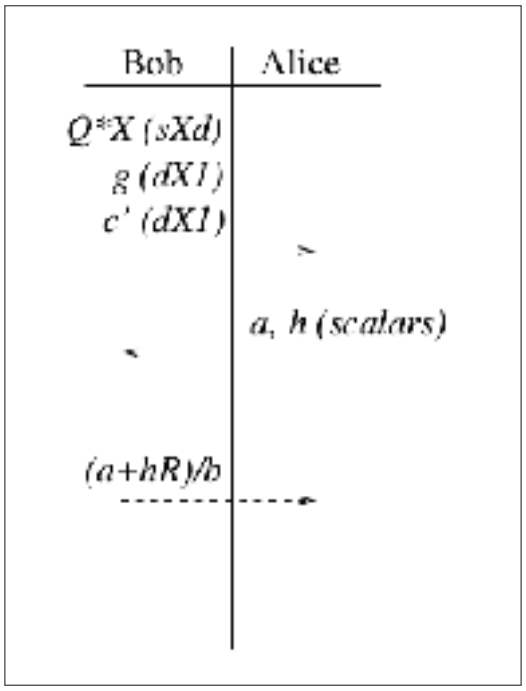


Figure 2.1 Communications Between two participants

a unique element b_i. These elements would form the vector $\vec{B}$. The goal is to compute the dot product without revealing any value. In the same way, it will be discussed in the presence of a third party or the absence of it. For this set-up, four different algorithms are presented in details. These solutions have been proposed by Dumas *et al.* (2017). In all these solutions, Paillier cryptosystem is used, however any other semi-homomorphic cryptosystem can be used, but this may lead to different overall costs.

In the first algorithm, called **DSDP**, the participant $\mathscr{A}$ and $\mathscr{B}_i$ parties follow the steps as follow. All the parties, except $\mathscr{A}$, encrypt their element with their own public key, and send it to $\mathscr{A}$. Thus, the party $\mathscr{B}_i$ would send to $\mathscr{A}$, the encrypted message $E_{\mathscr{B}_i}(b_i)$, for $1 \leq i \leq t$. Then, $\mathscr{A}$ would multiply the receiving values by the corresponding elements of the vector $\vec{A}$ homomorphically, and masks these values with some random numbers, r_i. Thus, $\mathscr{A}$ would compute the following values

- $E_{\mathscr{B}_i}(b_i)^{a_i} \times E_{\mathscr{B}_i}(r_i) = E_{\mathscr{B}_i}(b_i a_i + r_i)$, for $1 \leq i \leq t$

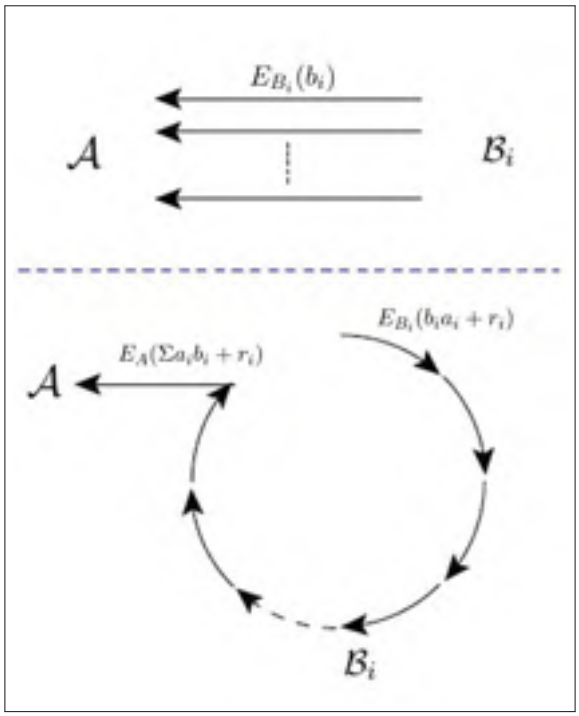


Figure 2.2 DSDP Algorithm Schematic

and send all these values to the first participant $\mathscr{B}_1$, as shown in Figure 2.2. This party would decrypt the first value using his own private key, re-encrypts it with the next party $\mathscr{B}_2$ public key, and add it to the second equation homomorphically. Thus, $\mathscr{B}_1$ would compute

1. $D_{\mathscr{B}_1}[E_{\mathscr{B}_1}(b_1a_1 + r_1)] = b_1a_1 + r_1$
2. $E_{\mathscr{B}_2}(b_1a_1 + r_1) \times E_{B_2}(b_2a_2 + r_2) = E_{B_2}(b_1a_1 + r_1 + b_2a_2 + r_2)$

sends the last result to the next participant $\mathscr{B}_2$.

This process continues till the last participant $\mathscr{B}_t$, who would send the result to the result owner $\mathscr{A}$, encrypted with $\mathscr{A}$ public key. The party $\mathscr{A}$ knows the random numbers, so he is able to eliminate these values, and get the dot product of the two vectors.

It is secure against semi-honest attackers but not against colluding ones. Obviously, if $\mathscr{A}$, $\mathscr{B}_i$ and $\mathscr{B}_{i+2}$ collude together, they can get the information of $\mathscr{B}_{i+1}$. Therefore, the best mitigation of this attack is to use a random ring order.

The total computation cost of this algorithm is $3t \times C_{enc} + t \times C_{dec} + 2t \times C_{mul} + t \times C_{exp}$. The participant $\mathscr{A}$ has to do $t \times C_{enc} + t \times C_{mul} + t \times C_{exp} + 1 \times C_{dec}$. On the other hand, each participant $\mathscr{B}_i$ has to do $2 \times C_{enc} + 1 \times C_{mul} + 1 \times C_{dec}$.

The second algorithm in the same set-up is called **YTP** (Dumas *et al.*, 2017). The first participant $\mathscr{A}$ has as entire vector $\vec{A}$ and some parties $\mathscr{B}_i$ own elements of the second vector b_i. The algorithm goal is finding the dot product of these vectors in a secure way.

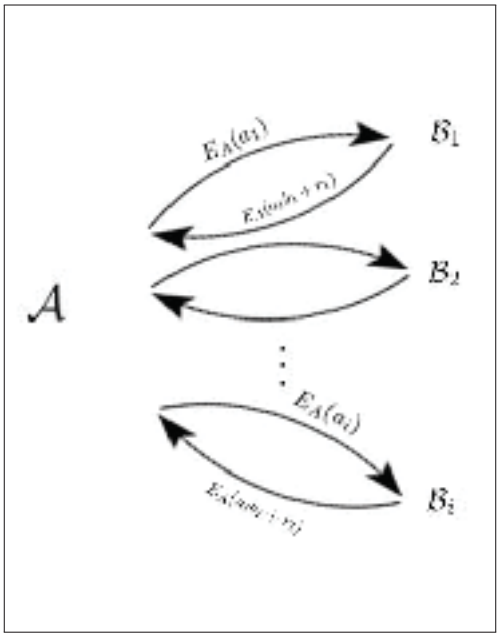


Figure 2.3 YTP Algorithm Schematic

First, $\mathscr{A}$ encrypts all the elements of the first vector individually by his public key and send them to corresponding parties.

- $\mathscr{A} : E_{\mathscr{A}}(a_1) \Longrightarrow \mathscr{B}_1, E_{\mathscr{A}}(a_2) \Longrightarrow \mathscr{B}_2, \cdots, E_{\mathscr{A}}(a_t) \Longrightarrow \mathscr{B}_t$

Then, the other participants picks a random number to mask their data and calculate the following equation homomorphically and sent it back to $\mathscr{A}$. Thus,

- $\mathscr{B}_i : E_A(a_i)^{b_i} \times E_A(r_i) = E_A(a_i b_i + r_i) \Longrightarrow \mathscr{A}$.

After this first phase, the participant $\mathscr{A}$ has the value of $\sum a_i b_i + \sum_i r_i$. The latter summation should be removed in order to obtain the desired result.. So far, the computation cost of this algorithm is $2t \times C_{enc} + t \times C_{mul}$. T All the parties contribute in a classical *salary summation* to calculate the random number securely and send it to $\mathscr{A}$. This algorithm uses also Paillier's crypto-system and it is secure against semi-honest attackers, but can be improved to be secure against malicious adversaries. The original salary summation goes as follows:

1. The first participant $\mathscr{B}_1$ selects a random value s and send to the send participant the encrypted value of $E_{\mathscr{B}_2}(r_1 + s)$.

2. The second participant $\mathscr{B}_2$ decrypts the value of $r_1 + s$, adds its own random value r_2, and re-encrypts the resulting value with the key of the next participants $E_{\mathscr{B}_2}(r_1 + r_2 + s)$.

3. The last participant does the same operations but send it to the initial participant $\mathscr{B}_1$. Since this participant knows the original value of s, he can retrieve the sum of $\sum_i r_i$ to send to the participant $\mathscr{A}$.

Thus, all these participants add $t \times C_{enc} + t \times C_{dec}$.

Naturally, this classical protocol does not resist to colluding adversaries. One elegant solution has been proposed by Dumas *et al.* (2017). They proposed to repeat the dot product for k iterations - k depending on the number of potential colluding adversaries. Hence, the participants determine a random permutation in which the classical salary protocol would be done, for a given iteration. This permutation should be determined with the help of a cryptographically robust hashing function. Hence, for a given participant $\mathscr{B}_i$, this participant would partition his random value r_i into k parts such that $r_i = \sum_j r_{i,j}$. In the worst case, in a given iteration, a pair

of colluding adversaries would only get one element of the partition, which is useless without the other parts. Dumas *et al.* (2017) present the implementation and the optimization of such a resistant and secure protocol.

The third algorithm proposed here is called **MPWP** Dolev *et al.* (2010). First, the participant $\mathscr{A}$ encrypts all the elements of his t-dimensional vector $\vec{A}$ individually and forms a new vector $\vec{T}$. The master also creates a $(t \times t)$- matrix Z whose all elements are initialized to 1 and a variable $A = 1$. All these values are sent to the first participant $\mathscr{B}_1$. Once the participant $\mathscr{B}_i$ has received the values (Z, T, A), he would update these values as follows before to send them to the participant $\mathscr{B}_{i+1}$:

1. $\mathscr{B}_i$ picks a random value r_i and update $A = A \times E_{\mathscr{A}}(a_i)^{b_i} \times E_{\mathscr{A}}(r_i)$
2. $\mathscr{B}_i$ would partition his value r_i into t elements $r_{i,j}$ such that $r_i = \sum_j r_{i,j}$.
3. The column vector $(r_{i,1}, \cdots, r_{i,t})$ would be encrypted in replace the column i of Z. Hence, the element $r_{i,j}$ is encrypted with the public key of the participant $\mathscr{B}_j$ - important: the encryption does not have to be homomorphic.

At the end, the last participant $\mathscr{B}_t$ sends all the values to $\mathscr{A}$ who can retrieve the masked result $\sum_i a_i b_i + r_i$.

In the second phase of this algorithm, the matrix Z is sent to all the participants. Hence, the participant $\mathscr{B}_i$ can retrieve all the partition values encrypted with his own key. The values are the row i of the matrix Z. By decrypting these values, $\mathscr{B}_i$ can sum these elements and return this value back to $\mathscr{A}$ - encrypted with $\mathscr{A}$ public key. When $\mathscr{A}$ has received all the encrypted values of $\sum_j r_{i,j}$, he would be able to unmask the dot product of $\vec{A} \cdot \vec{B}$.

Dumas *et al.* (2017) proposed a slight improvement on the last algorithm. Instead of transferring back and forth the matrix Z, a given participant $\mathscr{B}_i$ would simply sends his part $E_{\mathscr{B}_j} r_{i,j}$ encrypted directly to the participant $\mathscr{B}_j$. During this protocol, all the participants would collect the received values. At the end, they would simply send back to the participant $\mathscr{A}$ the

encrypted sum of the received value. This optimization simply reduces the communication flows between the participants. Otherwise, the protocol is the same.

2.3 Matrix Multiplication

Among the most common tools in computer science and engineering, matrices stand out. In many time-sensitive engineering applications, matrix multiplications represent a major burden. To respond to this problem, numerous solutions have been proposed to reduce the time complexity of the matrix multiplication.

Naive Algorithm

This approach is used to compute the multiplication of two matrices of compatible sizes (the number of columns of the first matrix should be the equal to the number of rows of the second matrix). This algorithm is the easiest but the most expensive one. Its computation cost is $\mathcal{O}(n^3)$, for $n \times n$ matrices. It is mainly used for simple small matrices in non time-sensitive applications.

Strassen's algorithm

The first divide-and-conquer algorithm has been proposed by Strassen (1969). It is much faster algorithm than the naive algorithm. It decreases the complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^{\log_2 7})$, which is considerable for $n > 100$ (Deng & Ramanan, 2017).

Strassen's algorithm works on square matrices with equal sizes. Then, both matrices are divided into four equal size quadrants as shown in the next figure. Assume there are two square matrices $[A]_{2^k \times 2^k}$ and $[B]_{2^k \times 2^k}$, finding the product of these two matrices $[C]_{2^k \times 2^k} = [A] \times [B]$ is the goal. The details are given in Algorithm 2.1.

Many more algorithms were introduced for matrix multiplication since the seminal work of Strassen with less computation costs over time such as Bini & Lotti (1980), Schönhage (1981),

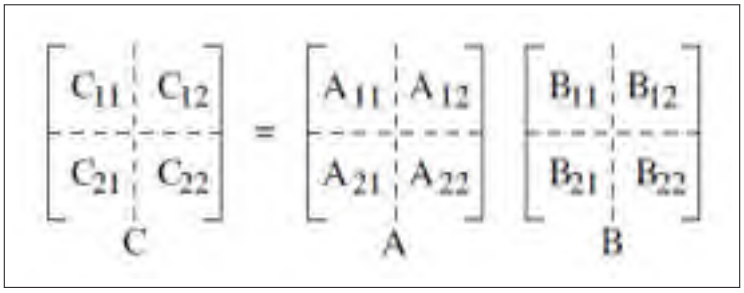


Figure 2.4 Strassen's Algorithm Initializing

Coppersmith & Winograd (1990) and finally the least costly algorithm introduced by Stothers (2010).

2.3.1 Secure/Private Matrix Multiplication

In parallel with the development of the classical matrix multiplication algorithms, researchers have been proposed solutions to distribute such algorithms in cloud environments. As mentioned in the introduction, their objectives are mainly to take advantage of massive computational resources. However, this represents a major threat against the privacy of the data.

In this section, many private/secure matrix multiplication algorithms are presented. Our choices have been done based on the relatedness with our solution presented in the next chapter and their practicality.

Fully distributed set-ups: Bultel *et al.* (2017) solutions

First we describe algorithms which were introduced by Bultel *et al.* (2017). The different parties and communication rounds are illustrated in Figure 2.6.

In the worst case scenario for the matrix multiplication problem, all the elements of initial matrices are distributed in different nodes. Thus, there would be $2n^2$ data owners - in the remaining of this chapter, we would only consider square $n \times n$ matrices. More over, two intermediate trusted nodes, $\mathscr{R}$ and Q, are introduced. They do not have any data but just have enough resources to help in the algorithm process. The last actor of this algorithm is the result

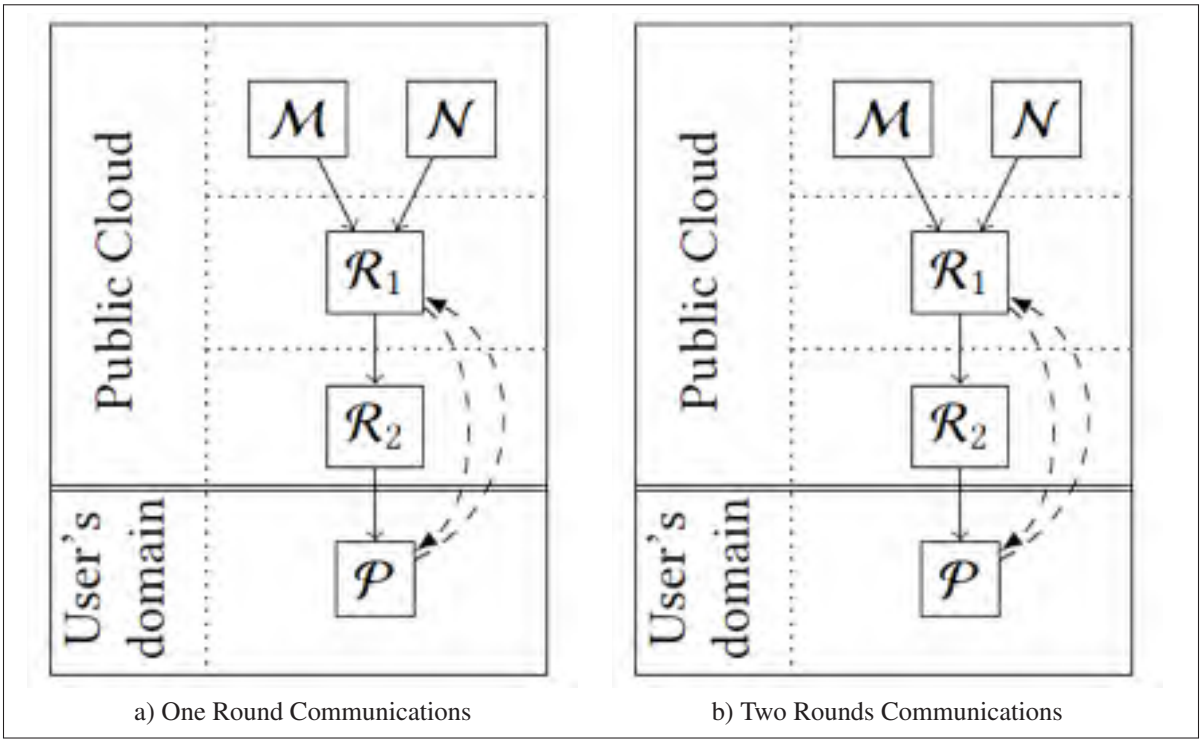
Data: Two $2^t \times 2^t$ matrices: $[A], [B]$.
Result: The product $[C] = [A] \times [B]$.
if $t \geq 2$ **then**
  Divide both matrices into four submatrices, for $1 \leq \alpha, \beta \leq 2$:
    $A_{\alpha,\beta} = [a_{(\alpha-1)2^{t-1}+i,(\beta-1)2^{t-1}+j}]$,
    $B_{\alpha,\beta} = [b_{(\alpha-1)2^{t-1}+i,(\beta-1)2^{t-1}+j}]$, for $1 \leq i, j \leq 2^{t-1}$
  Compute the following submatrices:
    $S_1 \leftarrow A_{2,1} + A_{2,2}$ $S_2 \leftarrow S_1 - A_{1,1}$
    $S_3 \leftarrow A_{1,1} - A_{2,1}$ $S_4 \leftarrow A_{1,2} - S_2$
    $T_1 \leftarrow B_{1,2} - B_{1,1}$ $T_2 \leftarrow B_{2,2} - T_1$
    $T_3 \leftarrow B_{2,2} - B_{1,2}$ $T_4 \leftarrow T_2 - B_{2,1}$
  Call this procedure recursively:
    $\mathsf{Strassen}(A_{1,1}), B_{1,1}, (R_1), t-1)$ $\mathsf{Strassen}((A_{1,2}), B_{2,1}, (R_2), t-1)$
    $\mathsf{Strassen}((S_4),\ B_{2,2}, (R_3), t-1)$ $\mathsf{Strassen}((A_{2,2}), T_4,\ (R_4), t-1)$
    $\mathsf{Strassen}((S_1),\ T_1,\ (R_5), t-1)$ $\mathsf{Strassen}((S_2),\ T_2,\ (R_6), t-1)$
    $\mathsf{Strassen}((S_3),\ T_3,\ (R_7), t-1)$
  Compute the following submatrices:
    $(U_1) = (R_1) + (R_2)$ $(U_2) = (R_1) + (R_6)$
    $(U_3) = (U_2) + (R_7)$ $(U_4) = (U_2) + (R_5)$
    $(U_5) = (U_4) + (R_3)$ $(U_6) = (U_3) - (R_4)$
    $(U_7) = (U_3) + (R_5)$
  return $(AB) = [(U_1)||(U_5), (U_6)||(U_7)]$
else
  $(r_{1,1}) = (a_{1,1})(b_{1,1}) + (a_{1,2})(b_{2,1})$
  $(r_{1,2}) = (a_{1,1})(b_{1,2}) + (a_{1,2})(b_{2,2})$
  $(r_{2,1}) = (a_{2,1})(b_{1,1}) + (a_{2,2})(b_{2,1})$
  $(r_{2,2}) = (a_{2,1})(b_{1,1}) + (a_{2,2})(b_{2,2})$
  return $(AB) = (R)$

Figure 2.5 $\mathsf{Strassen}([A], [B], [AB], t)$

owner P who needs the product. The goal of the participants is to compute the product of two matrices securely, without leaking information to third parties.

SP-2Rounds

The first step of this algorithm is to mask the elements of the matrices. All the elements of the first matrix A are encrypted with the public key of P. The elements of the second matrix B are



a) One Round Communications b) Two Rounds Communications

Figure 2.6 Communications Between Included Parties

masked with random numbers $r_{i,j}$ picked by the data owners. These values are encrypted with the public key of the second third party Q and send to him. In summary:

- First matrix element owners: $E_P(a_{ij}) \longrightarrow R, Q$
- Second matrix element owners: pick r_{jk} , $b_{jk} + r_{jk} \longrightarrow R$, $E_Q(r_{jk}) \longrightarrow Q$

Then, R computes the multiplication of corresponding elements homomorphically and send the result to next party, Q.

- $R : E_P(a_{ij})^{(b_{jk}+r_{jk})} = E_P(a_{ij}b_{jk}) + E_P(a_{ij}r_{jk}) \longrightarrow Q$

At this point, Q decrypts the random numbers and deletes them from the result. Then, he computes the summations homomorphically and forms the product to send it the user P encrypted.

- $Q : \frac{E_P(a_{ij})^{(b_{jk}+r_{jk})}}{E_P(a_{ij})^{r_{jk}}} = E_P(a_{ij})^{b_{jk}} = E_P(a_{ij}b_{jk})$ and the encrypted resulting element is given by $E_P(\Sigma_j(a_{ij}b_{jk})) = \Pi_j E_P(a_{ij}b_{jk})$.

This algorithm uses Paillier's cryptoystem and it is secure against semi-honest attackers but not secure against malicious attackers so it is not collision resistant. The algorithm complexity is $(C_{sum} + 2C_E)n^2 + (2C_{mul} + 2Cexp + C_D)n^3$.

The complexity of this solution can be easily reduced if only one of the matrix has to be kept secret. In such a case, only one trusted party is necessary. Thus, R can receive the encrypted elements of A and the elements of B and computes each element $E_P(\Sigma_j(a_{ij}b_{jk})) = \Pi_j E_P(a_{ij})^{b_{jk}} \longrightarrow P$.

CRSP-One Round

If the previous algorithm was secure against semi-honest participants, it could not resist to malicious colluding participants. Hence, if data owners of the second matrix and the second trusted party R collude, R would be able to obtain too much information. In the next solution, the algorithm has been designed to thwart this problem.

First, all the elements of *both* matrices are encrypted with the public key of P and sent to the third party R.

- Data owners: $E_P(a_{ij}) \longrightarrow R$, $E_P(b_{jk}) \longrightarrow R$

Then, R computes the multiplication of corresponding elements using the interactive protocol presented in Figure 2.7. Hence, R would be able to obtain $E_P(a_{ij}b_{jk})$ with the help of P without giving any information to P. This can be achieved by increasing the communication complexity of the overall protocol.

Knowing the result of elements' multiplication $E_P(a_{ij}b_{jk})$, R can easily compute the summations homomorphically and send these results to the user P

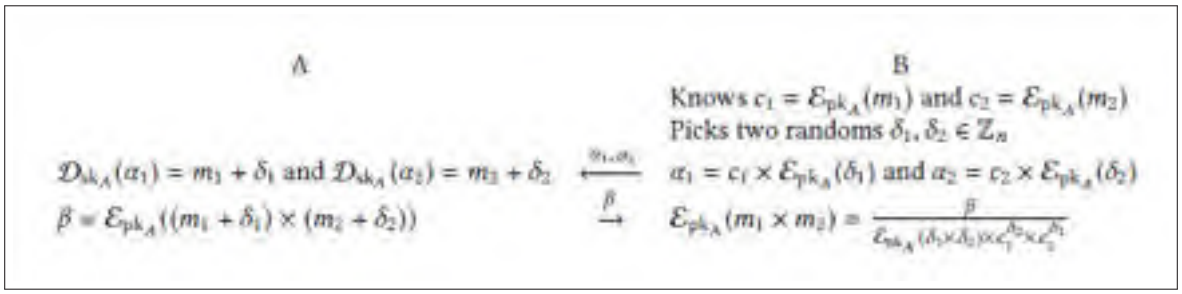


Figure 2.7 Interactive Protocol

- $R : E_P(\Sigma_j(a_{ij}b_{jk})) = \Pi_j E_P(a_{ij})^{b_{jk}} \longrightarrow P$.

This algorithm uses Paillier's cryptosystem and it is secure against semi-honest attackers and maliciously colluding third party R.

Outsourcing Large Matrix Computation to a Malicious Cloud (Lei *et al.*, 2014)

In this context, only one resource-limited data owner has both matrices. This entity wants to obtain from a potentially malicious cloud service the product of his two matrices. The computation of the product of two input matrices A and B by the following algorithm is based on five functions: KeyGen, MMCEnc, MMCSolve, MMCDec, Result Verify, which are described in details below. For this algorithm, the dimensions of the matrices should be compatible but otherwise unrestricted. Thus, A has dimension $n \times m$ and B has dimension $m \times s$.

KeyGen: The data/result owner selects three sets of non-zero random numbers from the same domain as the matrix elements: $\{\alpha_1, \alpha_2, ..., \alpha_m\}$, $\{\beta_1, \beta_2, ..., \beta_n\}$ and $\{\gamma_1, \gamma_2, ..., \gamma_s\}$ Then, he selects three random permutations $\pi_1 \leftarrow RandP(1, ..., m)$, $\pi_2 \leftarrow RandP(1, ..., n)$ and $\pi_3 \leftarrow RandP(1, ..., s)$. These informations can be seen as the secret keys of the data owner.

MMCEnc: In this process the owner generates three invertible permuted diagonal matrices P_1, P_2 and P_3, where $P_1(i, j) = \alpha_i \delta_{\pi_1(i),j}$, $P_2(i, j) = \beta_i \delta_{\pi_2(i),j}$ and $P_3(i, j) = \gamma_i \delta_{\pi_3(i),j}$ - where $\delta(i, j) = 1$, if and only if $i = j$ (Kronecker delta function). Then, the owner computes $A' =$

$P_1AP_2^{-1}$ and $B' = P_2BP_3^{-1}$ and sends them to the cloud system. The inverse matrices for P_1, P_2, P_3 can be easily calculated as follows:

- $P_1^{-1}(i,j) = (\alpha_j)^{-1}\delta_{\pi_1^{-1}(i),j}$
- $P_2^{-1}(i,j) = (\beta_j)^{-1}\delta_{\pi_2^{-1}(i),j}$
- $P_3^{-1}(i,j) = (\gamma_j)^{-1}\delta_{\pi_3^{-1}(i),j}$

MMCSolve: The cloud computes $C' = A'B' = P_1ABP_3^{-1}$ using any internal efficient matrix multiplication algorithm and sends back C' to thes owner.

MMCDec: The owner simply computes $C = P_1^{-1}C'P_3 = AB$ to retrieve the desired value.

Result Verify:The owner can verify the correctness of the returned result by computing $P = A \times (B \times r) - C \times r$, where r is a random vector. The correct result should lead to $P = (0, ..., 0)^T$.

This algorithm masks the input values from the cloud system (without any encryption). However, the number of zero elements are revealed but not necessarily their positions.

The Privacy Preserving Scheme for Outsourcing MMC (Fu *et al.*, 2017)

As just mentioned for the previous algorithm, the number of zero values in the resulting matrix is revealed to the cloud system. This may represent a major disadvantage in some cases. The following protocol addresses this problem.

A *Privacy-preserving* process is added to the aforementioned five functions to hide the input values. This transformation helps the data owner to mask his information from the cloud system. Let assume that all the elements of a $r \times c$-dimensional input matrices X are within the range $[-K, K]$, for some $K = 2^l (l > 0)$. The data owner can hide the privacy of these values by adding a random matrix to his input value $X' = X + R$. This matrix R can be defined as $R = UV$, where U is a $(m \times k)$-matrix with random entries in $[-2^p, \cdots, 2^p]$ and V is a $(k \times n)$-matrix with random entries in $[-2^l, \cdots, 2^{l+q}]$, for some $2 \leq k \ll r, c$. This matrix can be computed in time

$\mathcal{O}(nmk)$, hence the importance of selecting $k \ll r, c$. Otherwise, the all process would be useless since the data owner would have to compute an expansive matrix multiplication himself.

The remaining of the algorithm is somehow similar to the previous solution.

KeyGen: The data owner specifies a positive integer k $(2 \leq k \ll min(m,n,s))$ and builds two random matrices $R_1 = U_1V_1$ and $R_2 = U_2V_2$ as described in the Privacy-preserving process. Thus, $A' = A + R_1$ and $B' = B + R_2$.

MMCSolve: The cloud system must calculate the product of two received randomized matrices and send back the result $C = A'B' = (A + R_1)(B + R_2)$ to the data owner.

MMCDec: The result is $AB = C - S$ where $S = AR_2 + R_1B + R_1R_2 = (A + R_1)R_2 + R_1B$. Fortunately, the two matrix multiplications can be done more efficiently due to the construction of these random matrices. Thus, $(A + R_1)R_2 = (A + R_1)U_2V_2$, which can be computed in $\mathcal{O}(nmk + nsk)$ with the naive matrix multiplication. Similarly, R_1B can be computed in $\mathcal{O}(msk + nsk)$. Assuming that $k \ll n, m, s$ is therefore crucial.

The cost of this algorithm is very important compared to the previous one. The data owner has to do at least two expensive matrix multiplications – not with the full size but nevertheless expensive. There is therefore a trade-off between this solution and the previous one leaking the number of zero entries.

Private and Cheat-free Outsourcing of Algebraic Computations (Benjamin & Atallah, 2008)

Assume that a data owner $\mathcal{A}$ has two $n \times n$ input matrices A and B and because of his limited computation resources cannot execute more than $\mathcal{O}(n^2)$ computations nor the expensive public key encryptions of all input values. The goal is obtaining the product matrix AB using two untrusted third parties, let say cloud system S_1 and S_2, with enough computational resources. As usual, there should not be any information leakage about the input matrices or their product to

the cloud systems. Moreover, any malicious behavior of any cloud system should be detectable with high probability by $\mathscr{A}$.

The first step of this algorithm is hiding the input matrices. So the owner $\mathscr{A}$ generates two random matrices R_1 and R_2 and computes $A' = A - R_1$ and $B' = B - R_2$. He also generates two other random matrices U and V and computes their Schur product $R = U \odot V$ - this corresponds to the multiplication of the corresponding pairs of elements (i.e., $u_{i,j} \times v_{i,j}$). Then, $\mathscr{A}$ sends R_1, R_2 and U to the cloud S_1, and A', B' and V to the cloud S_2. He also sends some temporary public and *private* keys of a semi-homomorphic cryptographic system to S_1.

Now, S_1 computes R_1R_2, $Enc_{\mathscr{A}}(R_1)$, $Enc_{\mathscr{A}}(R_2)$ and $Enc_{\mathscr{A}}(U)$ and sends them back to $\mathscr{A}$. On his side, S_2 computes $A'B'$ and sends it back to $\mathscr{A}$. Then, $\mathscr{A}$ sends the values just received from S_1 i.e., R_1R_2, $Enc_{\mathscr{A}}(R_1)$, $Enc_{\mathscr{A}}(R_2)$ and $Enc_{\mathscr{A}}(R')$ to S_2.

Here, S_2 computes $Enc_{\mathscr{A}}(R_1B')$, $Enc(A'R_2)$ and $Enc_{\mathscr{A}}(U \odot V) = Enc_{\mathscr{A}}(R)$. This can be done due to the homomorphic property of the cryptographic system and the fact that S_2 has always the second element in cleartext. Finally, S_2 computes and sends back to $\mathscr{A}$ the following matrix: $Enc_{\mathscr{A}}(R_1B') \odot Enc_{\mathscr{A}}(A'R_2) \odot Enc_{\mathscr{A}}(R) = Enc_{\mathscr{A}}(R_1B' + A'R_2 + R)$. The owner $\mathscr{A}$ sends the received matrix to S_1 who decrypts it and send it back in plaintext. $\mathscr{A}$ subtracts R and gets $R_1B' + A'R_2$. Finally, he adds this value with the value of $A'B' - R_1R_2$, which has been received previously, to obtain the desired result AB.

The above protocol is vulnerable to any malicious behavior of the cloud systems S_1 and S_2. They can easily return wrong results. By modifying this scheme, $\mathscr{A}$ can catch any incorrect result by high probability. He simply needs to generates a random $n \times 1$ vector matrix V and computes $(AB)V$ and $A(BV)$, if the equations are equal, the cloud systems are honest with high probability otherwise they did not follow the instructions.

Efficient Secure Outsourcing Computation of Matrix Multiplication in Cloud Computing Zhang *et al.* (2016)

The last solution presented in the chapter is a bit different than the other ones. It has a participant who plays a new role in the solution. It is also based on a different cryptographic paradigm.

Three parties are included in this scheme: the data owner $\mathscr{A}$ (also referred to as the client of the system), the cloud system and the verification party. It is based on five functions: Key generation, Preprocessing, Requesting, Computation, Verification, Decryption. The system model is shown in Figure 2.8.

The new actor has an important role. His task is to verify the correctness of the encrypted computations done by the cloud system.

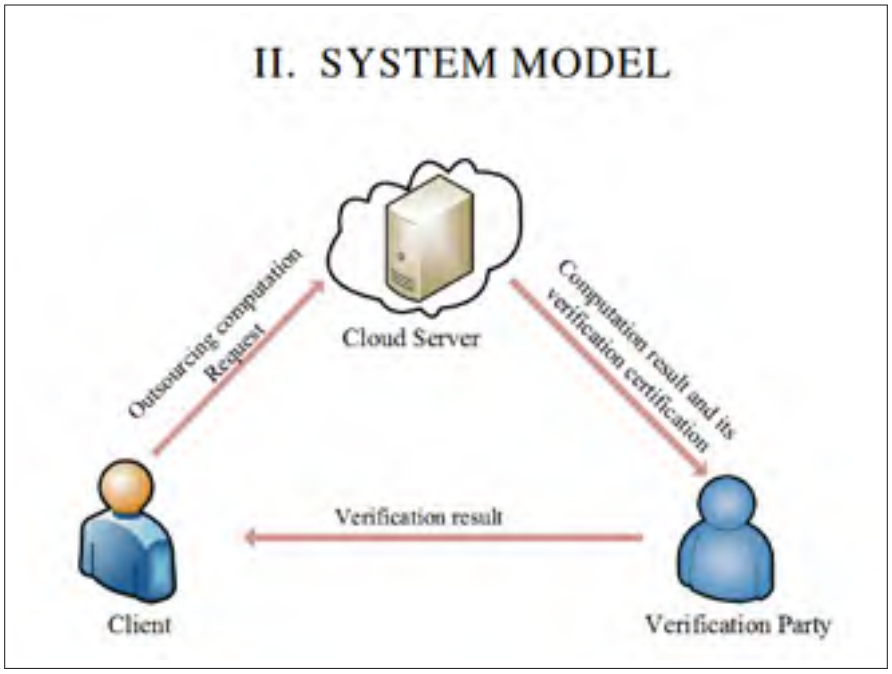


Figure 2.8 The System Model of the Proposed Algorithm

KeyGen: The cloud generates the parameters of bi-linear groups: $param = (q, g_1, g_2, g_t, G_1, G_2, G_t, e)$. The cloud publishes *param* as well as the hash of his secret key SK.

This solution is the first one that is not using a semi-homomorphic cryptosystem. It is based on bilinear pairings, which is a powerful concept. It has been used extensively in the past. In this

context, only one example is given. Fiore & Gennaro (2012) proposed to use bilinear-pairings to develop a solution for the matrix multiplication allowing an external verifier to check the computations done by a third party.

Preprocessing: This is the masking phase done by the data owner. The input matrices A and B, which have dimension $n_1 \times n_2$ and $n_2 \times n_3$, respectively, would have two different preprocessings. Using the secret key and random vectors, $\mathscr{A}$ would compute D_A and D_B as shown in Figure 2.9 and Figure 2.10, respectively. Once these matrices would be masked they would be send to the cloud system.

Algorithm 1 Processing on the class $\mathcal{A}_c$

Input: $A \in \mathcal{A}_c$, ID_A

Output: α_a, β_a, D_A

1: Choose the n_1-dimensional vector α_a and n_2-dimensional vector β_a randomly.
2: $\tilde{A} = A + \alpha_a \beta_a^T$
3: **for** $i = 1$ to n_1 **do**
4: $\quad r_i = h(ID_A \| i, SK)$
5: $\quad (w_{i,j})_{n_1 \times n_2} = (g_1^{r_i \tilde{a}_{i,j}})_{n_1 \times n_2}$
6: **end for**
7: $D_A = (\tilde{A}, W, ID_A)$
8: **return** α_a, β_a, D_A

Figure 2.9 Preprocessing on $\mathscr{A}_c$

In both cases, the input matrices are randomized with a random matrix, which is produced by multiplying two random vectors. Furthermore, for the first matrix A, a matrix W is generated (on a line per line basis). This matrix is the verification key that would have to be used later on in this solution. For the second matrix B, a public key $PK2$ is generated (also on a line per line basis).

Requesting: Here, the data owner $\mathscr{A}$ requests the result of multiplication of the two matrices A and B by sending the IDs of matrices. He also computes and publishes PKT. This is the

Algorithm 2 Processing on the class $\mathcal{B}_c$

Input: $Y \in \mathcal{B}_c$, ID_B

Output: α_b, β_b, D_B, $PK2$

1: Choose the n_2-dimensional vector α_a and n_3-dimensional vector β_a randomly.
2: $\tilde{B} = B + \alpha_b \beta_b^T$
3: $D_B = (\tilde{B}, ID_B)$
4: **for** $i = 1$ to n_3 **do**
5: $\quad c_i = h(ID_B \| i, SK)$
6: $\quad pk2_i = g_2^{c_i}$
7: **end for**
8: **return** α_b, β_b, D_B, $PK2$

Figure 2.10 Preprocessing on $\mathscr{B}_c$

second part of the public key associated to the two matrices to be multiplied. The details are described in Figure 2.11.

Algorithm 3 Computing the public key VKT

Input: ID_A, ID_B, SK

Output: PKT

1: **for** $i = 1$ to n_1 **do**
2: $\quad$ **for** $j = 1$ to n_3 **do**
3: $\quad\quad r_i = h(ID_A \| i, SK)$
4: $\quad\quad c_j = h(ID_B \| j, SK)$
5: $\quad\quad pkt_{i,j} = g_2^{r_i c_j}$
6: $\quad$ **end for**
7: **end for**
8: **return** PKT

Figure 2.11 Computing the public key

Computation: The cloud computes the result of two received matrices according to the IDs and the verification key as well as shown below:

$$\tilde{Res} = \tilde{A}.\tilde{B} = (A + R_A)(B + R_B)$$

$$VK = (\pi_{i,j})_{n_1 \times n_3} = (\prod_{k=1}^{n_2} (\tilde{w}_{i,k})^{\tilde{b}_{k,j}})_{n_1 \times n_3}$$

The value of $\tilde{w}_{i,k}$ is the verification key entry of the corresponding randomized element of the matrix A (see Figure 2.9). The value of $\tilde{b}_{k,j}$ is simply the corresponding randomized element of the matrix B.

The result and verification key will be sent to the verification party. This step is crucial if the cloud system can not be fully trusted.

Verification: The verification party judges the correctness of $\tilde{Res}$ by computing the equation below:

$$e(\pi_{i,j}, pk2_j) = (pkt_{i,j})^{\tilde{Res}_{i,j}}$$

If all the conditions are true, then he send the results to the client.

Decryption: The client recovers the final result be subtracting R from verified $\tilde{Res}$ where R is defined as below:

$$R = (A\alpha_b)\beta_b^T + \alpha_a(\beta_a^T B) + \alpha_a(\beta_a^T \alpha_b)\beta_b^T$$

CHAPTER 3

PRIVATELY SECURE MATRIX MULTIPLICATION

In this chapter we discuss about four privately secure algorithms for matrix multiplication based on Strassen's algorithm. In the next section, we propose the basic encrypted Strassen's algorithm, called EStrassen, which is applied on one input matrix encrypted under public key semi-homomorphic cryptosystem and the other one may be simply masked or not with random values. Then, in the following sections, a privately secure matrix multiplication is described with two trusted party and, finally, improved versions of the algorithm using only one trusted third party are presented.

3.1 Encrypted Strassen's Algorithm: EStrassen

To develop this privately secure algorithm ,we need to encrypt the first input matrix A with an additive semi-homomorphic public-key cryptosystem. This means that the addition of the clairtexts can be done effectively by doing some computations on their cyphertext. This process encrypts each element of the matrix independently. The second input matrix should be in plain text but could be randomized, permuted or rotated. The final result is therefore encrypted. Algorithm 3.1 shows the details of EStrassen when the second matrix is not masked. Since the second input matrix B is not masked in this process, the trusted party will learn all the elements of B, while the information of matrix A remain secure since its entries are encrypted. Practically we do not use this algorithm as an independent algorithm for matrix one, but use it as a subroutine to provide privately secure algorithms in the following sections.

Remember the context of the problem considered in this thesis:

- Different data owners are involved in this algorithm since the input matrix are distributed among different participants;

- One or two trusted third parties are needed to do the computationally intensive multiplications. These participants are semi-honest. They will follow the protocols but if they have the opportunity to discover the data they will exploit this security breach;
- The result owner (or the client) for who the computation is done. His public key should be available to any participant - in a trusted manner.

Proposing an algorithm involving two trusted parties is straightforward. Unfortunately, it does not seem to have a simple solution involving only trusted party. Some compromises have to be done. In the two trusted party solution, the risk of collusion is considerable and should be paid attention while having one trusted party decreases the possible attacks.

3.2 Private Secure Algorithm Involving Two Trusted Parties

System Model

The main problem discussed in this section is to securely multiply two square $2^t \times 2^t$ matrices, say A and B, while protecting the privacy of their elements. Data owner parties may know one or more elements, row, column or even a complete input matrix, but in the most general set-up, we consider that all the elements are distributed among independent nodes. Hence, the elements of matrix A are distributed in different 2^{2t} nodes (called $\mathscr{A}_{ij}$) and the elements of matrix B are distributed in different 2^{2t} nodes (called $\mathscr{B}_{ij}$) , so there are 2^{2t+1} data owners. Also there are two intermediate trusted parties, $\mathscr{T}_1$ and $\mathscr{T}_2$ (third parties) who perform the computations but do not have any information about the input matrices. The client $\mathscr{O}$, stands out of the system, needs the product of two matrices, $C = A \times B$. The described system model is illustrated in Figure 3.1 including communication channels.

The algorithm should meet the following properties:

- The user $\mathscr{O}$, cannot learn any information about input matrices, A and B
- None of the nodes of $\mathscr{A}$ can learn any information about B and C.

Data: Two $2^t \times 2^t$ matrices: $HE_O(A), B$ and the public keys of O.
Result: The encrypted product $HE_O(AB)$.
if $t \geq 2$ **then**
Divide both matrices into four submatrices, for $1 \leq \alpha, \beta \leq 2$:
$HE_O(A_{\alpha,\beta}) = [HE_O(a_{(\alpha-1)2^{t-1}+i,(\beta-1)2^{t-1}+j})]$,
$B_{\alpha,\beta} = [b_{(\alpha-1)2^{t-1}+i,(\beta-1)2^{t-1}+j}]$, for $1 \leq i, j \leq 2^{t-1}$
Compute the following submatrices:
$HE_O(S_1) \leftarrow HE_O(A_{2,1}) \odot HE_O(A_{2,2})$
$HE_O(S_2) \leftarrow HE_O(S_1) \odot HE_O(A_{1,1})^{-1}$
$HE_O(S_3) \leftarrow HE_O(A_{1,1}) \odot HE_O(A_{2,1})^{-1}$
$HE_O(S_4) \leftarrow HE_O(A_{1,2}) \odot HE_O(S_2)^{-1}$
$T_1 \leftarrow B_{1,2} - B_{1,1}$
$T_2 \leftarrow B_{2,2} - T_1$
$T_3 \leftarrow B_{2,2} - B_{1,2}$
$T_4 \leftarrow T_2 - B_{2,1}$
Call this procedure recursively:
$\mathsf{Strassen}(HE_O(A_{1,1}), B_{1,1}, HE_O(R_1), t-1)$
$\mathsf{Strassen}(HE_O(A_{1,2}), B_{2,1}, HE_O(R_2), t-1)$
$\mathsf{Strassen}(HE_O(S_4), B_{2,2}, HE_O(R_3), t-1)$
$\mathsf{Strassen}(HE_O(A_{2,2}), T_4, HE_O(R_4), t-1)$
$\mathsf{Strassen}(HE_O(S_1), T_1, HE_O(R_5), t-1)$
$\mathsf{Strassen}(HE_O(S_2), T_2, HE_O(R_6), t-1)$
$\mathsf{Strassen}(HE_O(S_3), T_3, HE_O(R_7), t-1)$
Compute the following submatrices:
$HE_O(U_1) = HE_O(R_1) \odot HE_O(R_2)$
$HE_O(U_2) = HE_O(R_1) \odot HE_O(R_6)$
$HE_O(U_3) = HE_O(U_2) \odot HE_O(R_7)$
$HE_O(U_4) = HE_O(U_2) \odot HE_O(R_5)$
$HE_O(U_5) = HE_O(U_4) \odot HE_O(R_3)$
$HE_O(U_6) = HE_O(U_3) \odot HE_O(R_4)^{-1}$
$HE_O(U_7) = HE_O(U_3) \odot HE_O(R_5)$
return $HE_O(AB) = [HE_O(U_1)||HE_O(U_5), HE_O(U_6)||HE_O(U_7)]$
else
$HE_O(r_{1,1}) = HE_O(a_{1,1})^{b_{1,1}} \odot HE_O(a_{1,2})^{b_{2,1}}$
$HE_O(r_{1,2}) = HE_O(a_{1,1})^{b_{1,2}} \odot HE_O(a_{1,2})^{b_{2,2}}$
$HE_O(r_{2,1}) = HE_O(a_{2,1})^{b_{1,1}} \odot HE_O(a_{2,2})^{b_{2,1}}$
$HE_O(r_{2,2}) = HE_O(a_{2,1})^{b_{1,1}} \odot HE_O(a_{2,2})^{b_{2,2}}$
return $HE_O(AB) = HE_O(R)$

Procedure 1: $\mathsf{EStrassen}(HE_O(A), B, HE_O(AB), t)$

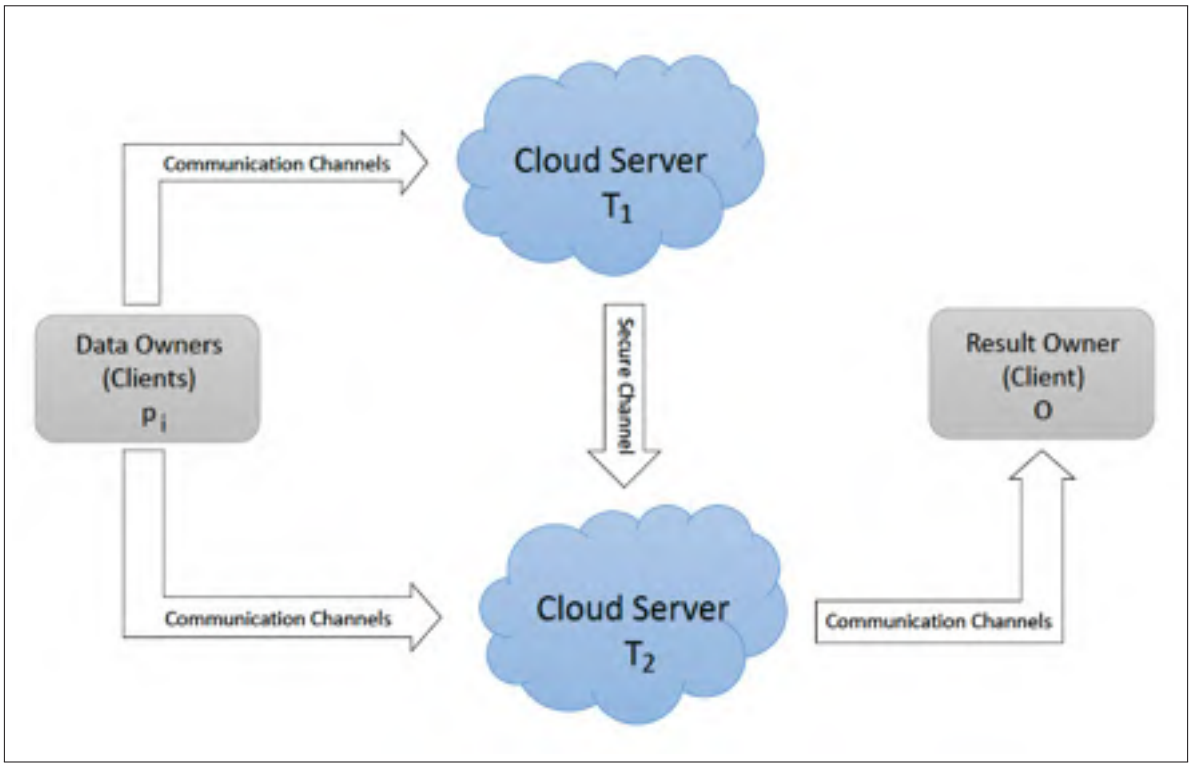


Figure 3.1 The System Model of the secure algorithm including two trusted parties.

- None of the nodes of $\mathscr{B}$ can learn any information about A and C.
- None of the intermediate nodes, $\mathscr{T}_1$ and $\mathscr{T}_2$, can learn any information about A, B and C.

In the algorithm presented here, first of all, each individual entry of the matrix A is encrypted with an additive semi-homomorphic cryptographic system (e.g., Paillier's cryptosystem presented in Chapter 2), and the matrix B is masked by random $2^t \times 2^t$ matrix R (this can be seen as a one-time pad encryption process) The encryption of the matrix A used the result owner public key, which has been distributed to all participants. Both encrypted matrix are sent to the first trusted party $\mathscr{T}_1$. On the other hand, the encrypted random matrix $HE_{T_2}(R)$ is sent to $\mathscr{T}_2$ as well.Remember the goal is to send encrypted result of the matrix multiplication to the result

owner $\mathscr{O}$. The participant can be seen as the client of this process. He will decrypt the received matrix with his private key and obtain the product.

Data: Two $2^t \times 2^t$ matrices: $HE_O(A)$, $B+R$ and $HE_{T_2}(R)$.

Result: The encrypted product $HE_O(AB)$.

$\mathscr{T}_1$ receives $HE_O(A)$ and $B+R$ from the data owners

$\mathscr{T}_1$ calls $\mathsf{EStrassen}(HE_O(A), B+R, HE_O(A(B+R)), t)$ and sends the result to $\mathscr{T}_2$

$\mathscr{T}_2$ receives $HE_O(A)$ and $HE_{T_2}(R)$ from the data owners, and $HE_O(A(B+R))$ from $\mathscr{T}_1$

$\mathscr{T}_2$ computes $HE_O(AB) = \frac{HE_O(A(B+R))}{HE_O(A)^R}$

return $HE_O(AB)$ *to the result owner* $\mathscr{O}$

Procedure 2: Multiplying two matrices with two trusted third parties

Security Analysis

The aforementioned algorithm is secure against the semi-honest adversaries. This means that if all the parties follow the algorithm but anyone tries to obtain more information about the initial data or the result, there is no success since the input matrices are either encrypted or randomized using long enough random number or keys.

However, we have to consider that the communications with the two trusted parties take place over secure channels (in system communications). Otherwise, the result owner would be able to retrieve easily the matrix A, which is simply encrypted with his own public key. Once this first matrix is known, it can be inverted and the matrix B will be readily obtained from the resulting matrix AB - since $A^{-1}(AB) = B$).

Unfortunately, if there are some collusion among the participants, the different matrices are at risk. In the case of a collusion between $\mathscr{T}_1$ and $\mathscr{T}_2$, they would obtain easily the matrix B. $\mathscr{T}_2$ would need to decrypt the random matrix R and share it with $\mathscr{T}_1$. This latter participant would then retrieve easily the values of the matrix B. On the other hand, since all the entries of the matrix A are encrypted with the public key of the result owner $\mathscr{O}$, there will not be any information leakage of this matrix even in case of collusion between the third parties. But if

one of them colludes with $\mathscr{O}$, everything falls apart. Just too much information is shared among these malicious participants.

Finally, if the result owner $\mathscr{O}$ can pretend to be one of the data owners and send some input matrices to the trusted parties, again the all protocol falls apart. For example, if $\mathscr{O}$ pretends to own the matrix A and sends it $\mathscr{T}_1$, he would be able to retrieve easily the matrix B without any more effort. He would simply send the identity matrix I and would obtain back form $\mathscr{T}_2$ the resulting matrix $IB = B$.

3.3 Privately secure algorithms involving only one trusted party

In this section, we describe three different solutions based on ESreassen involving only one trusted party - reducing the needs of finding two independent but cooperating trusted parties and obviously reducing the opportunities of collusion. We begin to present all the common hypotheses which are considered for the next three proposed algorithms.

3.3.1 Preliminaries

Recalling Basic Matrix Properties:

In the following algorithms, some basic matrix concepts in linear algebra are used. They are recalled here:

1. Identity Matrix : An Identity or Unit matrix is a square matrix which all the values on the main diagonal are one and rest of the values are zero. This matrix is denoted I

2. Transposed Matrix: The transpose of a matrix is an operator which flips a matrix over its main diagonal, that is it switches the row and column indices of the matrix by producing another matrix, which is denoted as M^T. Hence, this row i of the matrix M becomes the column i of the corresponding matrix M^T.

3. Inverse Matrix: The right inverse of a matrix M, which is denoted M^{-1}, is such that $MM^{-1} = I$. Similarly, the left inverse of a matrix M is such that $M^{-1}M = I$. Remark that if M is a square matrix, its right inverse is also its left inverse. This means that $MM^{-1} = M^{-1}M = I$.

4. Orthogonal Matrix: An orthogonal matrix is a square matrix whose columns and rows are orthogonal unit vectors. Such a matrix Q must respect the following properties:

   - $Q^T Q = QQ^T = I$
   - $Q^T = Q^{-1}$

5. Permutation Matrix: It is a square binary matrix which has only one entry of 1 in each row and each column. Each such matrix can be obtained by permuting the rows (or equivalently the columns) of the identity matrix I. If P is such a matrix, the multiplication PA simply permutes the rows of the matrix A. Similarly, the multiplication AP simply permutes its columns.

System Model:

Assume there are some data owners with limited computational resources which need the product of their two matrices $A_{2^t \times 2^t}$ and $B_{2^t \times 2^t}$. Each independent participant $\mathscr{P}_i$ knows the i^{th} column of the matrix A and the corresponding i^{th} row of the matrix B. Also, there is a trusted third party (a cloud system) with powerful and fast resources who would follow any the proposed matrix multiplication algorithm but would try at the same time to obtain as much as information about the input matrices and the resulting one. Figure 3.2 shows the included parties and the communication flow between them.

Hypotheses:

1. An orthogonal permutation matrix P is defined and distributed secretly between the involved participants such that:

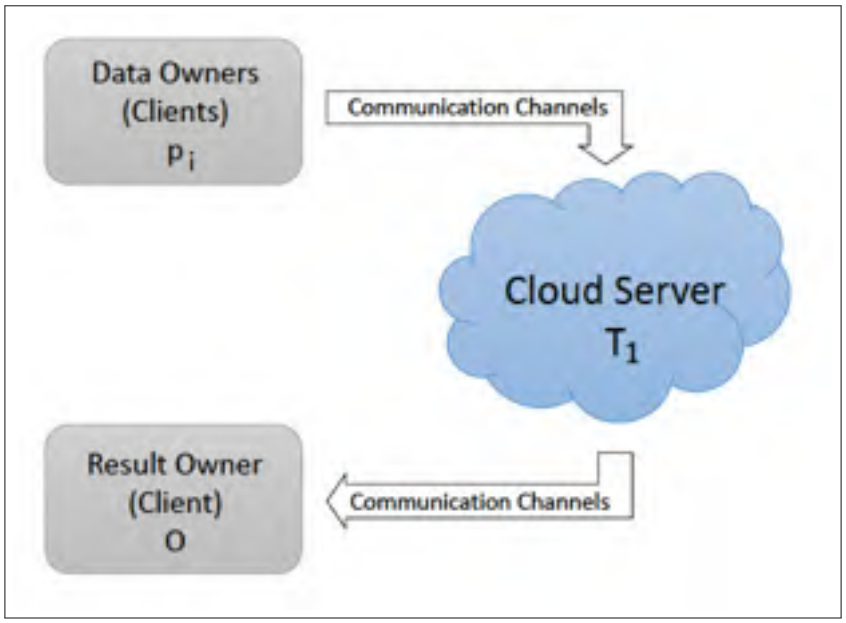


Figure 3.2 The System Model of the Proposed algorithms

- p_i only has to know his corresponding value π_i of the underlying permutation π used to define the permutation matrix P.
- The trusted party $\mathscr{T}$ does not have any information on the initial matrix distribution. This means that the participant p_i knowing the i^{th} column of matrix $A_{2^t \times 2^t}$ and the i^{th} row of matrix $B_{2^t \times 2^t}$ pretends to know the π_i^{th} column of matrix $[AP]_{2^t \times 2^t}$ and the π_i^{th} row of matrix $[P^T B]_{2^t \times 2^t}$ where $[P^T]$ is the transpose of matrix P.

2. The encrypted components of $HE_O(A)$ should be sent anonymously to $\mathscr{T}_1$ somehow to preserve the privacy of the owners. It means $\mathscr{T}_1$ should not be able to distinguish which data owner p_i knows which row or column of the input matrices. To perform this communication we propose two solutions:

   - There is an anonymized/secure channel between the data owners and the trusted party, which would prevent $\mathscr{T}_1$ to learn about the identity of the data owners. Each participant should send an anonymized matrix $HE_O(A)$ to $\mathscr{T}_1$ which is filled with zeros except for the appropriate row. Then, $\mathscr{T}_1$ would simply need to sum the received matrices to

derive the desired matrix. This follows from the semi-homomorphic property of the underlying cryptosystem.

- The first data owner p_1 makes a $2^t \times 2^t$ matrix which filled by zeros except for the row π_1 and sends it to P_2. This second participant replaces the zeros in the row π_2 with his encrypted data and send it to the next data owner. This process continues till the full matrix is formed by all the participants. The finalized matrix is sent to $\mathscr{T}_1$.

There is naturally some trade-offs between the two solutions. The first one is very costly from the computational point of view. The trusted third party $\mathscr{T}_1$ would have to do $2^t - 1$ matrix additions. Each of them necessitates a quadratic number of homomorphic operations. On the other hand, the second solution would require more coordination among the participant. The encrypted matrix would have to be exchanged among all of them. Each participant would receive a matrix and would send a matrix. This method doubles the among of information exchanged for any participant.

3.3.2 First Algorithm

The trusted party $\mathscr{T}_1$ receives $HE_O(AP)$ and P^TB anonymously and calls EStrassen to provide the result. Algorithm 3.3 illustrates the calculations and the communications between parties.

Data: Two $2^t \times 2^t$ matrices: $HE_O[AP]$ and P^TB.
Result: The encrypted product $HE_O(AB)$.
$\mathscr{T}_1$ receives $HE_O[AP]$ and P^TB anonymously from the participants $\mathscr{P}_i$
$\mathscr{T}_1$ calls $\mathsf{EStrassen}(HE_O(AP), P^TB, HE_O(AP \times P^TB), t)$
return $HE_O(AB) = HE_O(AP \times P^TB)$ *to the result owner* $\mathscr{O}$

Procedure 3: Multiplication with a permuted matrix P^TB.

Security Analyze

Since the first matrix A is encrypted with the result owner public key, the other participants and the trusted party will not get any information about it. But the second matrix B is just permuted. This means that the trusted party would obtain the permuted columns of the matrix

B, obtaining some information. For example, if the sum of the elements of a column represents any significant information, it would be considered as an information leakage. Thus, this first solution leads to an information leakage on the second matrix B.

3.3.3 Second Algorithm

To improve the privacy protection of the input matrix B in this algorithm, we split into two *random* matrices. Let assume that $B = B_1 + B_2$. For example, a totally random matrix R can be used to obtain these two matrices. $B_1 = B - R$ and $B_2 = R$. Naturally, each participant $\mathscr{P}_i$ would simply have to know the corresponding column i of the random matrix to transform his column of B. Let assume also there are two independent permutation matrices P_1 and P_2 distributed securely between the participants $\mathscr{P}_i$. The data are then sent to $\mathscr{T}_1$ in two steps by the participants. Obviously, the anonymisation process described earlier is very important. Otherwise, this splitting solution is totally useless. The trusted party $\mathscr{T}_1$ would know the relation between these two matrices B_1 and B_2. The following algorithm describes how $\mathscr{T}_1$ computes the result securely.

Data: Four $2^t \times 2^t$ matrices: $HE_O[AP_1]$, $[P_1^T B_1]$, $HE_O[AP_2]$ and $[P_2^T B_2]$.
Result: The encrypted product $HE_O(AB)$.
$\mathscr{T}_1$ receives $HE_O[AP_1]$ and $[P_1^T B_1]$ anonymously from the participants $\mathscr{P}_i$
$\mathscr{T}_1$ calls $\mathsf{EStrassen}(HE_O(AP_1), [P_1^T B_1], HE_O(AP_1 \times P_1^T B_1), t)$
$\mathscr{T}_1$ receives $HE_O[AP_2]$ and $[P_2^T B_2]$ anonymously from the participants $\mathscr{P}_i$
$\mathscr{T}_1$ calls $\mathsf{EStrassen}(HE_O(AP_2), [P_2^T B_2], HE_O(AP_2 \times P_2^T B_2), t)$
return $HE_O(AB_1) \odot HE_O(AB_2) = HE_O(AB_1 + AB_2) = HE_O(AB)$
Procedure 4: Multiplication with a matrix B split into two matrices with **one** random matrix and two permutation matrices.

Security Analyze

As mentioned before, the security of the first encrypted matrix is preserved. If the trusted party tries to obtain some information on the second matrix B, he should permute the columns of matrices B_1 and B_2 to retrieve $B = B_1 + B_2$, which needs $2 \cdot 2^t!$ tries. If there is no collusion

between participants and the trusted party, no values of the input and result matrices would leak.

Another interesting point, if we take a given column of $P_1^T B_1$ and add it to all the columns of $P_2^T B_2$, one of the resulting 2^t columns would be a column of the original matrix B. Some information is leaking in this process. This particular aspect would be considered in our last solution presented in the following section.

3.3.4 Third Algorithm

Now we improve upon the second algorithm presented in the previous section by randomizing further the second matrix B . Assume $B = B_1 + B_2$ and there are two permutation matrices P_1, P_2. Each data owner pick two random vectors R_1 and R_2 and apply it two B_1 and B_2 respectively. As mentioned earlier, these participants would have to know only their own columns of these random matrices. Then, the permutation matrices P_1^T and P_2^T would mask the randomized data before sending to $\mathscr{T}_1$. They also send $R_1 + R_2$ and $HE_O A$ to $\mathscr{T}_1$. The following Algorithm 3.5 shows how $\mathscr{T}_1$ calculate the result securely.

Data: Five $2^t \times 2^t$ matrices: $HE_O[AP_1]$, $[P_1^T(B_1 + R_1)]$, $HE_O[AP_2]$, $[P_2^T(B_2 + R_2)]$ and $R_1 + R_2$.
Result: The encrypted product $HE_O(AB)$.
$\mathscr{T}_1$ receives $HE_O[AP_1]$ and $[P_1^T(B_1 + R_1)]$ anonymously from the participants $\mathscr{P}_i$
$\mathscr{T}_1$ calls $\mathsf{EStrassen}(HE_O(AP_1), [P_1^T(B_1 + R_1)], HE_O(AP_1 \times P_1^T(B_1 + R_1)), t)$
$\mathscr{T}_1$ receives $HE_O[AP_2]$ and $[P_2^T(B_2 + R_2)]$ anonymously from the participants $\mathscr{P}_i$
$\mathscr{T}_1$ calls $\mathsf{EStrassen}(HE_O(AP_2), [P_2^T(B_2 + R_2)], HE_O(AP_2 \times P_2^T(B_2 + R_2)), t)$
$\mathscr{T}_1$ receives $HE_O A$ and $R_1 + R_2$ anonymously from the participants $\mathscr{P}_i$
$\mathscr{T}_1$ calls $\mathsf{EStrassen}(HE_O A, -(R_1 + R_2), HE_O[-A(R_1 + R_2), t]$
return $HE_O(AB) = HE_O[A(B_1 + R_1)] \odot HE_O[A(B_2 + R_2)] \odot HE_O[-A(R_1 + R_2)]$
Procedure 5: Multiplication with a matrix B split into two matrices with **two** random matrices and two permutation matrices

Security Analyze

Trying $2 \cdot 2^t!$ permutations, the trusted party $\mathscr{T}_1$ will get the $B + (R_1 + R_2)$. Since he knows $R_1 + R_2$ in plain text he gets the second input matrix. If we delegate the third and forth steps to the data owner to eliminate the random number, then the privacy of this algorithm is improved, however, it leads to some computation costs on the result owner. This can be considered somehow as going back to a two trusted party algorithm.

The approach can be more parametrized, Instead of splitting B into $B_1 + B2$, it can be split into three or more matrices, then more permutations are needed as well and more EStrassen multiplication has to be called. Depending on the priority of the preserving security or cost of the computations a Trade-off can be set for the numbers of the portions. However, this does not improved a lot the protection.

In the recent three proposed algorithms, the data owners only encrypt/decrypt the first matrix and the cost of operations on the second matrix is neglectable. All the other heavy computations are performed by the powerful cloud resources which makes the aforementioned algorithms more efficient than executing the matrix multiplication algorithm by the data owners.

CHAPTER 4

COMPARISON WITH THE RECENTLY INTRODUCED ALGORITHM

4.1 Dumas et al Algorithm Definition

In April 2019, a new secure approach for matrix multiplication based on strassen's algorithm was introduced by Dumas et al. In this approach, we assumed that there are two $t \times t$ square matrices, called $\mathscr{A}$ and $\mathscr{B}$ and the goal is calculating the product matrix $\mathscr{C}$ securely. In this system model there are t parties that each one has one row of the first matrix $\mathscr{A}$ and the corresponding row of the second matrix $\mathscr{B}$, at the end, each one will learn one row of the result $\mathscr{C}$. First of all, they agree on the location sequences $L = \{l_1, l_2, ..., l_t\}$ and the key sequence $K = \{k_1, k_2, ..., k_t\}$, then generate the secret key and publish the public keys. It means player $\mathscr{P}_{l_i}$ stores row i of matrix A, that was encrypted with the public key pk_{k_i} of player $\mathscr{P}_{k_i}$ for all $1 \leq i \leq t$. Since the Strassen's algorithm divides the matrices into four equal quadrants, the location and key sequence should be splitted into sub-sequences: for $X \in \{A, B, C\}, L_X = (L_{X_U}, L_{X_L})$ and $K_X = (K_{X_U}, K_{X_L})$ such that (L_{X_U}, K_{X_U}) are the location and key sequences for the upper half of X and (L_{X_L}, K_{X_L}) are the location and key sequences for the lower half of X. These notations are illustrated in figure 4.1

The semi-homomorphic Naccache-stern cryptosystem Naccache & Stern (1998) is used in this algorithm to encrypt and decrypt data.

Here we describe the algorithm in details:

Initialization Phase

As mentioned above, before the computations related to the algorithm, the involved parties should agree on the sequences L and K and share their public key and encrypted data as well. Figure 4.2 shows the data exchanging based on the sequences.

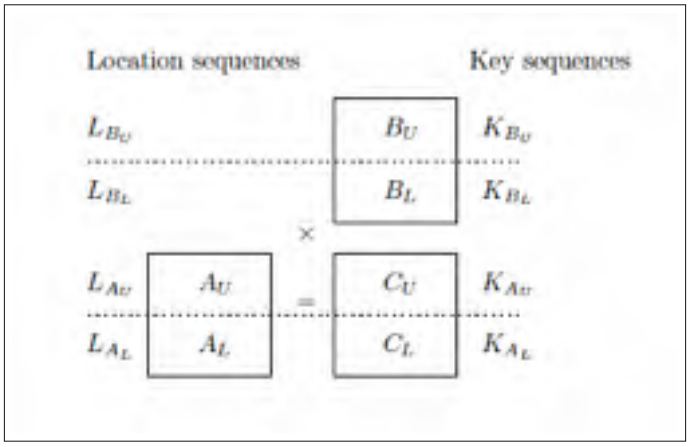


Figure 4.1 Recursive splitting of the location and key sequences of the input and output operands in Strassen-Winograd algorithm

Protocol 1 SW-Setup

Input: Two $N \times N$ matrices A and B over $\mathbb{Z}_q$, where $N = b2^\ell$, such that party P_i knows the i-th row of A and the i-th row B for all $i \in \{1..N\}$. A location and a key sequence $L \in \{1..N\}^N$ and $K \in \{1..N\}^N$ such that (L, K, L, K) form an ℓ-recursive data layout, following Definition 2. All parties know a security parameter λ.

Output: For all $i \in \{1..N\}$, party $P_{L[i]}$ learns vectors $\{a_{i,*}\}_{K[i]}$ and $\{b_{i,*}\}_{K[i]}$ and learns the public key of every other party.

Goal: Generate key pairs for each party, cipher and distribute input matrices according to their respective location and key sequences.

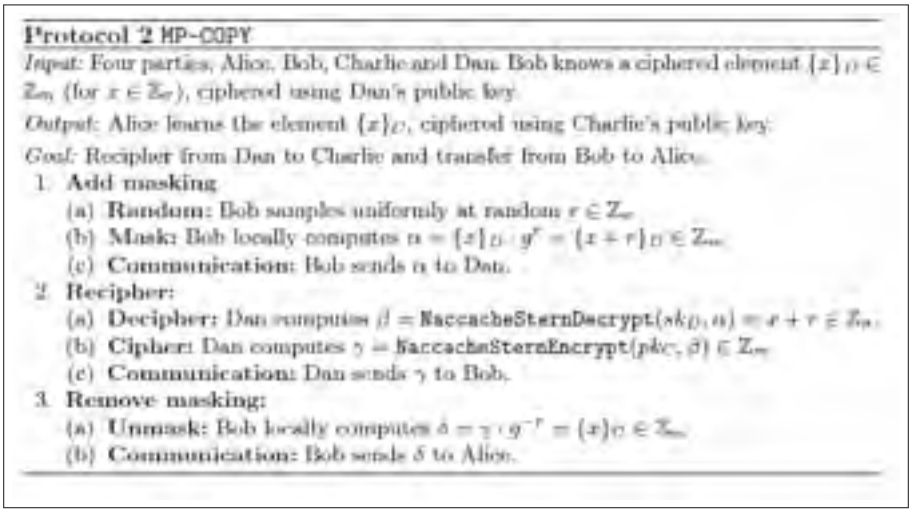
1. **Key generation:** for all $i \in \{1..N\}$, each party P_i locally executes $\texttt{HaccacheSternSetup}(1^\lambda)$ to generate a pair of keys (pk_i, sk_i).
2. **Broadcast keys:** for all $i \in \{1..N\}$, party P_i broadcasts its public key pk_i.
3. **Cipher inputs:** for all $i \in \{1..N\}$, for all $j \in [v]$, party P_i locally performs $\texttt{HaccacheSternEncrypt}(pk_{K[i]}, a_{i,j})$ and stores the result as a new vector $\{a_{i,*}\}_{K[i]}$. It does the exact same operation with $b_{i,*}$ to get $\{b_{i,*}\}_{K[i]}$.
4. **Distribute rows:**
   (a) **Rows of A:** for all $i \in \{1..N\}$, party P_i sends $\{a_{i,*}\}_{K[i]}$ to party $P_{L[i]}$.
   (b) **Rows of B:** for all $i \in \{1..N\}$, party P_i sends $\{b_{i,*}\}_{K[i]}$ to party $P_{L[i]}$.

Figure 4.2 Initialization Phase of Strassen's Algorithm

At the end, the encrypted data are sent to the party designated by the location sequence. If the initial matrices are $t \times t$, then $2t^2$ communication will be done in this phase.

Multiparty Copy

Somewhere in this algorithm we need to copy and recipher a vector from one party to another following location and key sequence. MP-COPY protocol is described in the figure 4.3, performing this very operation for a given ciphered element x hosted by Bob and encrypted for Dan, to its new location at Alice and encrypted for Charlie. Dan is the party how does the decryption and re-encrcyption. To avoid leaking information, Bob mask the data with the random value. Totally 3 communication is needed to perform this protocol.



Protocol 2 `MP-COPY`

Input: Four parties, Alice, Bob, Charlie and Dan. Bob knows a ciphered element $\{x\}_D \in \mathbb{Z}_m$ (for $x \in \mathbb{Z}_\sigma$), ciphered using Dan's public key.

Output: Alice learns the element $\{x\}_C$, ciphered using Charlie's public key.

Goal: Recipher from Dan to Charlie and transfer from Bob to Alice.

1. **Add masking**
   (a) **Random:** Bob samples uniformly at random $r \in \mathbb{Z}_\sigma$.
   (b) **Mask:** Bob locally computes $\alpha = \{x\}_D \cdot g^r = \{x + r\}_D \in \mathbb{Z}_m$.
   (c) **Communication:** Bob sends α to Dan.
2. **Recipher:**
   (a) **Decipher:** Dan computes $\beta = \texttt{NaccacheSternDecrypt}(sk_D, \alpha) = x + r \in \mathbb{Z}_\sigma$.
   (b) **Cipher:** Dan computes $\gamma = \texttt{NaccacheSternEncrypt}(pk_C, \beta) \in \mathbb{Z}_m$.
   (c) **Communication:** Dan sends γ to Bob.
3. **Remove masking:**
   (a) **Unmask:** Bob locally computes $\delta = \gamma \cdot g^{-r} = \{x\}_C \in \mathbb{Z}_m$.
   (b) **Communication:** Bob sends δ to Alice.

Figure 4.3 Multiparty Copy Protocol

Additions and Subtractions

When two values, x and y are encrypted by the same public key, c, addition and subtraction will be performed by homomorphic property of the cryptosystem:

$$E_c\{x+y\} = E_c\{x\} \times E_c\{x\}$$

$$E_c\{x-y\} = E_c\{x\} / E_c\{x\}$$

But if the values are encrypted by different parties public key, stored in the others party, then multiparty addition and multiparty subtraction known as MP-ADD and MP-SUB are computed by applying MP-COPY and homomorphic addition or subtraction.

Totally 15 matrix addition and 7 recursive calls are used in Strassen's algorithm. All the matrix additions are performed by component-wise homomorphic additions, HOM-MAT-ADD in which each player homomorphically adds the two rows of the two input operands that she owns. In the same way homomorphic subtraction is performed, HOM-MAT-SUB, however requires that the two operands share the same key and location sequences. To meet this, some matrices should be translated from one key-location sequence to the other one using MP-MAT-COPY which is archived by t^2 instance of MP-COPY protocol as shown in protocol 3.

Protocol 6 MP-MAT-COPY

Input: an $n \times n$ matrix $[A]_{K_A}$ distributed and ciphered according to a location and a key sequence $(L_A, K_A) \in (\{1..N\}^n)^2$ among parties $(P_1, \ldots, P_N)$ following Definition 2 and a location-key sequence (L', K').

Output: A copy $[A]_{K'}$ is distributed and ciphered among parties $(P_1, \ldots, P_N)$ according to the location and key sequences (L', K').

For all $i, j \in \{1..n\}^2$
  Parties $P_{L(i)}, P_{L(j)}, P_{K(i)}$ and $P_{K'(i)}$ run MP-COPY to copy $[a_{i,j}]_{K(i)}$ to $[a_{i,j}]_{K'(i)}$

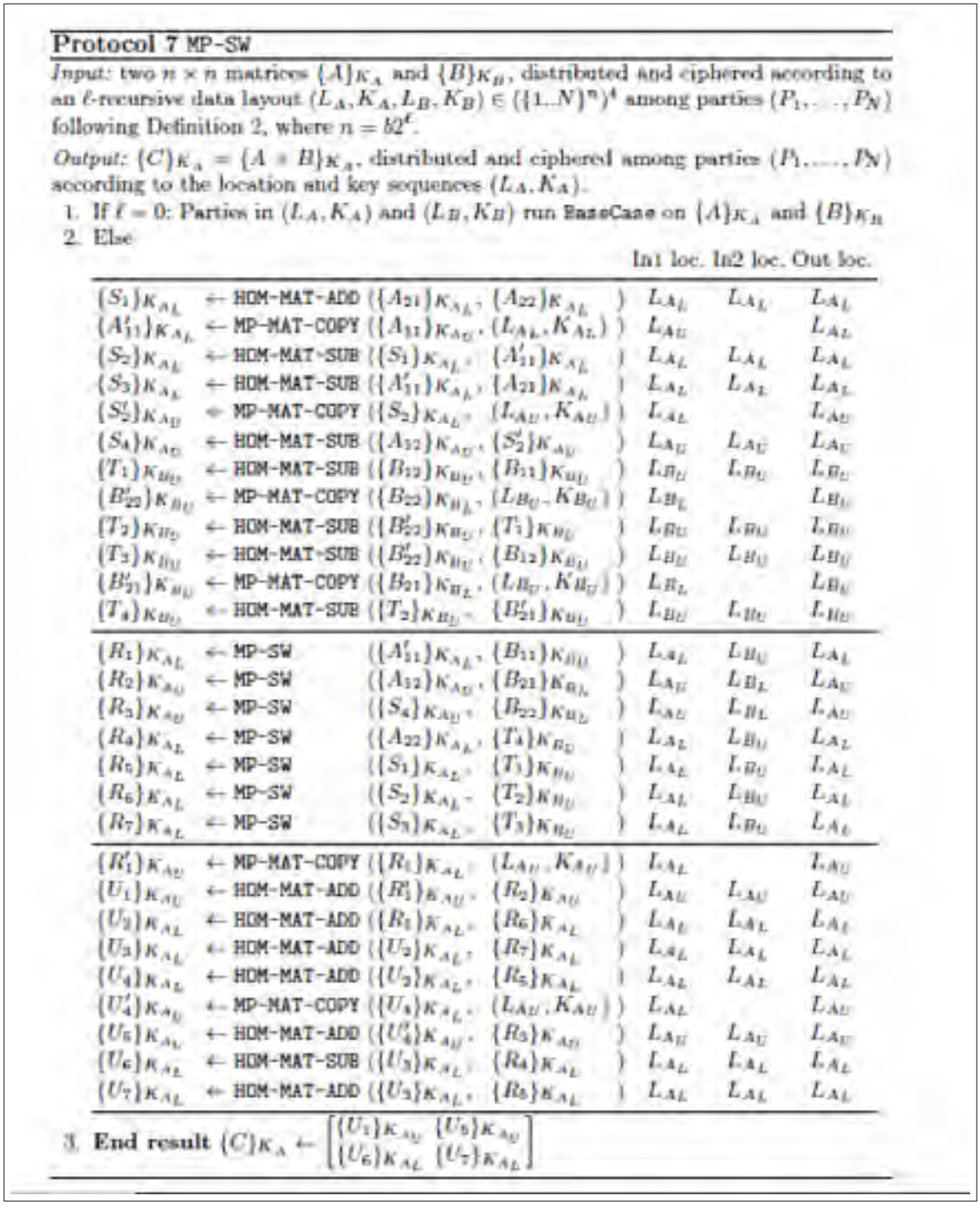
Figure 4.4 Multiparty MAT-Copy Protocol

Then in protocol 4, the scheduling the operands and data exchanging for the secure matrix multiplication based on Strassen's algorithm is defined.

Finalization Step

Finally the involved parties should decrypt and distribute each row of the product matrix to the corresponding nodes. This step is described in figure 4.6 using t^2 communications.

Based on the security analysis, this algorithm is secure against semi-honest parties, but in case of malicious attackers or any collusion between participant according to the location and key sequences, some data may leak.



Protocol 7 MP-SW

Input: two $n \times n$ matrices $\{A\}_{K_A}$ and $\{B\}_{K_B}$, distributed and ciphered according to an ℓ-recursive data layout $(L_A, K_A, L_B, K_B) \in (\{1..N\}^n)^4$ among parties $(P_1, \ldots, P_N)$ following Definition 2, where $n = b2^\ell$.

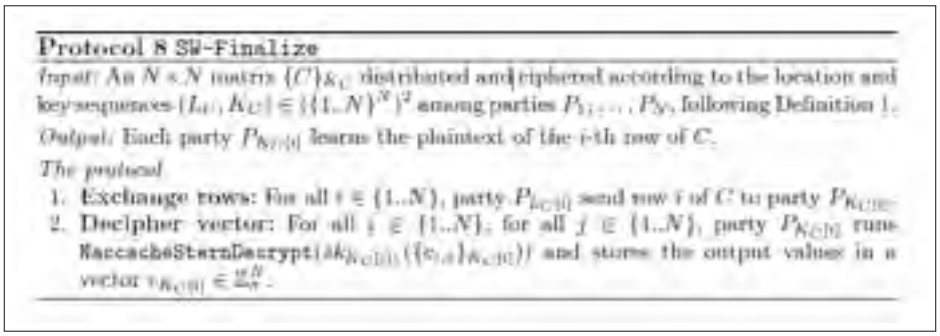
Output: $\{C\}_{K_A} = \{A * B\}_{K_A}$, distributed and ciphered among parties $(P_1, \ldots, P_N)$ according to the location and key sequences (L_A, K_A).

1. If $\ell = 0$: Parties in (L_A, K_A) and (L_B, K_B) run BaseCase on $\{A\}_{K_A}$ and $\{B\}_{K_B}$
2. Else

			In1 loc.	In2 loc.	Out loc.
$\{S_1\}_{K_{A_L}}$	← HOM-MAT-ADD	$(\{A_{21}\}_{K_{A_L}}, \{A_{22}\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{A'_{11}\}_{K_{A_U}}$	← MP-MAT-COPY	$(\{A_{11}\}_{K_{A_U}}, (L_{A_U}, K_{A_U}))$	L_{A_U}		L_{A_U}
$\{S_2\}_{K_{A_L}}$	← HOM-MAT-SUB	$(\{S_1\}_{K_{A_L}}, \{A'_{11}\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{S_3\}_{K_{A_L}}$	← HOM-MAT-SUB	$(\{A'_{11}\}_{K_{A_L}}, \{A_{21}\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{S'_2\}_{K_{A_U}}$	← MP-MAT-COPY	$(\{S_2\}_{K_{A_L}}, (L_{A_U}, K_{A_U}))$	L_{A_L}		L_{A_U}
$\{S_4\}_{K_{A_U}}$	← HOM-MAT-SUB	$(\{A_{12}\}_{K_{A_U}}, \{S'_2\}_{K_{A_U}})$	L_{A_U}	L_{A_U}	L_{A_U}
$\{T_1\}_{K_{B_U}}$	← HOM-MAT-SUB	$(\{B_{12}\}_{K_{B_U}}, \{B_{11}\}_{K_{B_U}})$	L_{B_U}	L_{B_U}	L_{B_U}
$\{B'_{22}\}_{K_{B_U}}$	← MP-MAT-COPY	$(\{B_{22}\}_{K_{B_L}}, (L_{B_U}, K_{B_U}))$	L_{B_L}		L_{B_U}
$\{T_2\}_{K_{B_U}}$	← HOM-MAT-SUB	$(\{B'_{22}\}_{K_{B_U}}, \{T_1\}_{K_{B_U}})$	L_{B_U}	L_{B_U}	L_{B_U}
$\{T_3\}_{K_{B_U}}$	← HOM-MAT-SUB	$(\{B'_{22}\}_{K_{B_U}}, \{B_{12}\}_{K_{B_U}})$	L_{B_U}	L_{B_U}	L_{B_U}
$\{B'_{21}\}_{K_{B_U}}$	← MP-MAT-COPY	$(\{B_{21}\}_{K_{B_L}}, (L_{B_U}, K_{B_U}))$	L_{B_L}		L_{B_U}
$\{T_4\}_{K_{B_U}}$	← HOM-MAT-SUB	$(\{T_2\}_{K_{B_U}}, \{B'_{21}\}_{K_{B_U}})$	L_{B_U}	L_{B_U}	L_{B_U}
$\{R_1\}_{K_{A_L}}$	← MP-SW	$(\{A'_{11}\}_{K_{A_L}}, \{B_{11}\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R_2\}_{K_{A_U}}$	← MP-SW	$(\{A_{12}\}_{K_{A_U}}, \{B_{21}\}_{K_{B_L}})$	L_{A_U}	L_{B_L}	L_{A_U}
$\{R_3\}_{K_{A_U}}$	← MP-SW	$(\{S_4\}_{K_{A_U}}, \{B_{22}\}_{K_{B_L}})$	L_{A_U}	L_{B_L}	L_{A_U}
$\{R_4\}_{K_{A_L}}$	← MP-SW	$(\{A_{22}\}_{K_{A_L}}, \{T_4\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R_5\}_{K_{A_L}}$	← MP-SW	$(\{S_1\}_{K_{A_L}}, \{T_1\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R_6\}_{K_{A_L}}$	← MP-SW	$(\{S_2\}_{K_{A_L}}, \{T_2\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R_7\}_{K_{A_L}}$	← MP-SW	$(\{S_3\}_{K_{A_L}}, \{T_3\}_{K_{B_U}})$	L_{A_L}	L_{B_U}	L_{A_L}
$\{R'_1\}_{K_{A_U}}$	← MP-MAT-COPY	$(\{R_1\}_{K_{A_L}}, (L_{A_U}, K_{A_U}))$	L_{A_L}		L_{A_U}
$\{U_1\}_{K_{A_U}}$	← HOM-MAT-ADD	$(\{R'_1\}_{K_{A_U}}, \{R_2\}_{K_{A_U}})$	L_{A_U}	L_{A_U}	L_{A_U}
$\{U_2\}_{K_{A_L}}$	← HOM-MAT-ADD	$(\{R_4\}_{K_{A_L}}, \{R_6\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{U_3\}_{K_{A_L}}$	← HOM-MAT-ADD	$(\{U_2\}_{K_{A_L}}, \{R_7\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{U_4\}_{K_{A_L}}$	← HOM-MAT-ADD	$(\{U_2\}_{K_{A_L}}, \{R_5\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{U'_4\}_{K_{A_U}}$	← MP-MAT-COPY	$(\{U_4\}_{K_{A_L}}, (L_{A_U}, K_{A_U}))$	L_{A_L}		L_{A_U}
$\{U_5\}_{K_{A_U}}$	← HOM-MAT-ADD	$(\{U'_4\}_{K_{A_U}}, \{R_3\}_{K_{A_U}})$	L_{A_U}	L_{A_U}	L_{A_U}
$\{U_6\}_{K_{A_L}}$	← HOM-MAT-SUB	$(\{U_3\}_{K_{A_L}}, \{R_4\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}
$\{U_7\}_{K_{A_L}}$	← HOM-MAT-ADD	$(\{U_3\}_{K_{A_L}}, \{R_5\}_{K_{A_L}})$	L_{A_L}	L_{A_L}	L_{A_L}

3. **End result** $\{C\}_{K_A} \leftarrow \begin{bmatrix} \{U_1\}_{K_{A_U}} & \{U_5\}_{K_{A_U}} \\ \{U_6\}_{K_{A_L}} & \{U_7\}_{K_{A_L}} \end{bmatrix}$

Figure 4.5 Scheduling for Strassen's Algorithm



Protocol 8 SW-Finalize

Input: An $N \times N$ matrix $\{C\}_{K_C}$ distributed and ciphered according to the location and key sequences $(L_C, K_C) \in (\{1..N\}^N)^2$ among parties $P_1, \ldots, P_N$, following Definition 1.

Output: Each party $P_{K_C[i]}$ learns the plaintext of the i-th row of C.

The protocol

1. **Exchange rows:** For all $i \in \{1..N\}$, party $P_{L_C[i]}$ send row i of C to party $P_{K_C[i]}$.
2. **Decipher vector:** For all $i \in \{1..N\}$, for all $j \in \{1..N\}$, party $P_{K_C[i]}$ runs `KaccacheSternDecrypt`$(sk_{K_C[i]}, (\{c_{i,j}\}_{K_C[i]}))$ and stores the output values in a vector $v_{K_C[i]} \in \mathbb{Z}_n^N$.

Figure 4.6 Finalization Step

A brief comparison:

As mentioned before, the system model which is used in this approach is completely different of the model we used in this thesis. Moreover, in our algorithm, the data owner will not learn any information about the final product but in Dumas et al approach every data owner get one row of the dot product. The other dissimilarity is the number of cryptographic keys. here, in our algorithm, we considered couple of public keys just for intermediate parties plus the user public key while in the other approach every data owner should store other parties public key to encrypt data in case of calling which leads to occupy a bigger storage size.

CONCLUSION AND RECOMMENDATIONS

In the real world, designing a privacy-preserving algorithm for mathematical computation is often very complex with the presence of different sorts of adversaries and attackers. In this thesis, we have presented some algorithms for different system models, which satisfy the definitions of a secure algorithm. Each one preserves (partially or totally) the privacy of the information. The applications of the proposed algorithms are big-data mining and analysis. In such cases, the importance of assuring the privacy of the results is crucial. Since the adversaries are usually malicious and have more and more resources, keeping the private information of the clients secure is more and more costly in terms of time and money. However, there should always be a trade-off between the costs and the level of privacy provided by the outsourcing algorithms. Although the proposed algorithms are not fully secure, they can be applied in most of matrix multiplication outsourcing set-ups because the information leakage is mathematically negligible. Moreover, in term of efficiency, we tried to decrease the computation costs on the client side and delegate the most expensive computations to the powerful cloud systems. The only significant computation which should be carry out by the client is the encryption of a matrix which is much cheaper than performing a regular matrix multiplication by himself. Additionally these algorithms avoid the data owners to obtain the product of the matrices which is an important consequence for outsourcing schemes.

Future Work

There are a couple of ways and directions to improve the privacy and efficiency of proposed algorithms. The following is a possible list of future related research direction that could be carried out to extend and improve this work:

- masking the both input matrices by public-key cryptosystem to increase the security of initial matrices which leads to interactive communications between the cloud server and the data owners.

- Masking the both input matrices by permutation or randomization to decrease the computation costs of the clients. Eliminating the random numbers in the product is the main challenge of this method.

- Using the other public-key cryptosystem and matrix multiplication methods to increase the efficiency.

- As mentioned before in the last algorithm, protecting the security of the algorithm needs delegating the last two steps to the result owner, which means more computation costs for him. Optimizing that algorithm and solving aforementioned problem could be considered as research for future.

BIBLIOGRAPHY

Benaloh, J. (1994). Dense probabilistic encryption. *Proceedings of the Workshop on Selected Areas of Cryptography*, pp. 120–128.

Benjamin, D. & Atallah, M. J. (2008). Private and cheating-free outsourcing of algebraic computations. *Proceedings of the Sixth Annual Conference on Privacy, Security and Trust*, pp. 240–245.

Bini, D. & Lotti, G. (1980). Stability of fast algorithms for matrix multiplication. *Numerische Mathematik*, 36(1), 63–72.

Björck, Å. (1994). Numerics of gram-schmidt orthogonalization. *Linear Algebra and Its Applications*, 197, 297–316.

Bultel, X., Ciucanu, R., Giraud, M. & Lafourcade, P. (2017). Secure Matrix Multiplication with MapReduce. *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pp. 11:1 – 11:10.

Coppersmith, D. & Winograd, S. (1990). Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3), 251–280.

Deng, M. & Ramanan, P. (2017). MapReduce Implementation of Strassen's Algorithm for Matrix Multiplication. *Proceedings of the 4th Algorithms and Systems on MapReduce and Beyond*, pp. 7:1 – 7:10.

Dolev, S., Gilboa, N. & Kopeetsky, M. (2010). Computing multi-party trust privately: in $O(n)$ time units sending one (possibly large) message at a time. *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 1460–1465.

Dumas, J.-G., Lafourcade, P., Orfila, J.-B. & Puys, M. (2017). Dual protocols for private multi-party matrix multiplication and trust computations. *Computers & Security*, 71, 51–70.

Dumas, J.-G., Lafourcade, P., Fenner, J., Lucas, D., Orfila, J.-B., Pernet, C. & Puys, M. (2019). Secure Multi-Party Matrix Multiplication Based on Strassen-Winograd Algorithm. *Proceedings of the 14th International Workshop on Security (IWSEC)*, pp. 67–88.

Fiore, D. & Gennaro, R. (2012). Publicly verifiable delegation of large polynomials and matrix computations, with applications. *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 501–512.

Fu, S., Yu, Y. & Xu, M. (2017). A Secure Algorithm for Outsourcing Matrix Multiplication Computation in the Cloud. *Proceedings of the Fifth ACM International Workshop on Security in Cloud Computing*, pp. 27–33.

Gentry, C. (2009). Fully Homomorphic Encryption Using Ideal Lattices. *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, pp. 169–178.

Hazay, C. & Lindell, Y. (2010). *Efficient secure two-party protocols: Techniques and constructions*. Springer Science & Business Media.

Ioannidis, I., Grama, A. & Atallah, M. (2002). A secure protocol for computing dot-products in clustered and distributed environments. *Proceedings of the International Conference on Parallel Processing*, pp. 379–384.

Kaosar, M. G., Paulet, R. & Yi, X. (2012). Fully homomorphic encryption based two-party association rule mining. *Data & Knowledge Engineering*, 76, 1–15.

Lei, X., Liao, X., Huang, T. & Heriniaina, F. (2014). Achieving security, robust cheating resistance, and high-efficiency for outsourcing large matrix multiplication computation to a malicious cloud. *Information sciences*, 280, 205–217.

Naccache, D. & Stern, J. (1998). A new public key cryptosystem based on higher residues. *Proceedings of the 5th ACM conference on Computer and Communications Cecurity*, pp. 59–66.

Okamoto, T. & Uchiyama, S. (1998). A new public-key cryptosystem as secure as factoring. *Proceedings of Advances in Cryptology - EUROCRYPT'98*, pp. 308–318.

Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. *Proceedings of Advances in Cryptology - EUROCRYPT'99*, pp. 223–238.

Schönhage, A. (1981). Partial and total matrix multiplication. *SIAM Journal on Computing*, 10(3), 434–455.

Stothers, A. J. (2010). *On the complexity of matrix multiplication*. (Ph.D. thesis, The University of Edinburgh).

Strassen, V. (1969). Gaussian elimination is not optimal. *Numerische mathematik*, 13(4), 354–356.

Zhang, S., Li, H., Jia, K., Dai, Y. & Zhao, L. (2016). Efficient secure outsourcing computation of matrix multiplication in cloud computing. *Proccedings of 2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6.