Secure Distributed MapReduce Protocols : How to have privacy-preserving cloud applications?

Matthieu Giraud

Université Clermont Auvergne
Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes (LIMOS)
CNRS, UMR 6158, LIMOS, 63 173 Aubière, France

Secure Distributed MapReduce Protocols

How to have privacy-preserving cloud applications?

Thèse

pour obtenir le

Doctorat de l'Université Clermont Auvergne
Spécialité informatique
École doctorale des Sciences pour l'Ingénieur

présentée et soutenue par

Matthieu Giraud

le 24 septembre 2019

devant le jury composé de :

M. Benjamin Nguyen	Professeur des universités LIFO, INSA Centre Val de Loire	Rapporteur
M^{me} Melek Önen	Maître de conférences Eurecom, Sophia Antipolis	Rapporteur
M^{me} Céline Chevalier	Maître de conférences Université Paris 2 Panthéon-Assas	Examinateur
M. Jean-Guillaume Dumas	Professeur des universités LJK, Université Grenobles Alpes	Examinateur
M. Sébastien Salva	Professeur des universités LIMOS, Université Clermont Auvergne	Examinateur
M. Pascal Lafourcade	Maître de conférences LIMOS, Université Clermont Auvergne	Directeur de thèse

UNIVERSITÉ Clermont Auvergne

LIMOS
LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Remerciements

En premier lieu, je tiens à remercier Pascal Lafourcade qui m'aura accompagné tout du long de cette thèse et envers qui je suis très reconnaissant. *Einmal ist keinmal*, une fois ne compte pas, une fois c'est jamais, mais je suis persuadé que Pascal est un directeur de thèse exceptionnel. Pascal, merci pour ton enthousiasme, tes conseils et ta disponibilité, tous sans faille. Merci aussi pour ces moments passés dans ton bureau à discuter et à bidouiller Emacs, Mutt, GnuPG, etc. Ton encadrement m'a beaucoup appris, non seulement scientifiquement, mais aussi humainement. Trois années, non des plus reposantes, mais si riches d'expériences et certainement inoubliables.

Je tiens également à remercier Radu Ciucanu avec qui j'ai travaillé durant une bonne partie de ma thèse et grâce à qui je me suis intéressé au traitement des *big data* à l'aide du paradigme MapReduce. De cette rencontre, en découle aussi cette thèse. Merci pour tes précieux conseils techniques, d'organisation et de rédaction.

Je remercie vivement les rapporteurs de cette thèse Melek Önen et Benjamin Nguyen d'avoir accepté la tâche ardue de relire ce manuscrit et pour le temps qu'ils y ont consacré. Je remercie également Céline Chevalier, Jean-Guillaume Dumas et Sébastien Salva pour avoir accepté d'être membres de mon jury. Leur présenter mes travaux fut un plaisir.

I would like to extend my deepest thanks to Rei Safavi-Naini for hosting me, during my PhD, for 4 months in her Institute for Security, Privacy and Information Assurance at the University of Calgary. This rich scientific collaboration was also humanly by his kind and fraternal meetings. Thanks to Alireza, Mamunur, Morshedul, Sabyasachi, Sepideh, Setareh, Shuai, Simpy, and Zain for their very warm welcome. This stay has certainly changed my life.

Mes remerciements vont également à William Guyot-Lénat qui par sa bonne humeur, son enthousiasme et sa curiosité perpétuels m'a grandement aidé à la réalisation des calculs effectués à l'aide de l'environnement Apache Hadoop®.

Ces trois années de thèse au laboratoire LIMOS, ce sont aussi des échanges plus que variés au quotidien avec des collègues doctorants. Merci Antoine et Alexis pour ces captivantes et uniques discussions, ainsi que pour ces promenades inatendues dans le campus. Merci David pour ces échanges tant appréciés et ces conférences partagées ; Šibenik, Zurich et Prague sont, notamment grâce à toi, de très bons souvenirs. Merci Xavier d'avoir partagé généreusement et toujours avec gentillesse tes connaissances en cryptographie. Merci Kergann et Alexandre pour avoir fait du bureau F-203 un mythe devenu réalité. Merci Marius pour ces moments plus que magiques. Merci Giacomo pour ce portrait accroché face à mon bureau qui m'a tant inspiré.

I also had the chance to work several times with Manik Lal Das and Hardik Gajera thanks to the Franco-Indian collaboration supported by the Indo-French Centre for the

Promotion of Advanced Research (IFCPAR) and the Center Franco-Indien Pour La Promotion De La Recherche Avancée (CEFIPRA). I want to thank especially Hardik for its warm welcome within Dhirubhai Ambani Institute of Information and Communication Technology and for the beautiful and succulent Indian cultural discoveries he brought me during my two stays in India. Such meetings are not forgotten.

Plus personnellement, je remercie de tout cœur mes parents pour m'avoir accompagné depuis toujours quelle que soit la direction prise. Leur écoute et leur aide sont inestimables. Qui savait ce que l'ordinateur (équipé d'un Pentium II cadencé à 233 MHz, de 32 Mo de mémoire vive et de 16 Go de disque dur) reçu pour Noël 1998 impliquerait... Grâce à vous, j'ai pu découvrir et littéralement expérimenter en toute liberté l'informatique, au gré bien entendu de quelques formatages. Je remercie aussi les intrépides et aventureux Chevreuils pour leur soutien et la fraîcheur qu'ils ont toujours su m'apporter. Je tiens également à remercier Marie-Christine et Frédéric pour leur prévenance, leurs précieux conseils et les moments régénérateurs.

Enfin, ces remerciements ne pourraient se finir sans te remercier profondément Pauline. Merci pour ton soutien inconditionnel, ton aide précieuse au quotidien et ta patience remarquable durant ces soirées et nombreux week-ends studieux. Merci d'avoir été aussi attentionnée et compréhensive. Cette thèse n'aurait jamais vu le jour sans toi.

Contents

1 Introduction **1**
- 1.1 A Brief History of Cryptography . . . 1
- 1.2 Cloud Computing Challenges . . . 2
- 1.3 The MapReduce Paradigm . . . 3
  - 1.3.1 Vocabulary and Notations . . . 3
  - 1.3.2 MapReduce . . . 4
- 1.4 Algorithms Using MapReduce . . . 5
  - 1.4.1 Matrix Multiplication . . . 5
  - 1.4.2 Relational-Algebra Operations . . . 6
- 1.5 Contributions . . . 6
- 1.6 Related Work . . . 7
  - 1.6.1 Matrix Multiplication . . . 7
  - 1.6.2 Relational-Algebra Operations . . . 8
- 1.7 Publications . . . 10
  - 1.7.1 Presented in this Manuscript . . . 10
  - 1.7.2 Other Publications . . . 10
- 1.8 Outline . . . 12

2 Technical Introduction **13**
- 2.1 Notations . . . 14
- 2.2 Mathematical Background . . . 14
- 2.3 Cryptographic Background . . . 15
  - 2.3.1 Cryptographic Tools . . . 15
  - 2.3.2 Hardness Assumptions . . . 17
- 2.4 Cryptographic Primitives . . . 18
  - 2.4.1 Hash Function . . . 18
  - 2.4.2 Pseudo-Random Function . . . 19
  - 2.4.3 Symmetric Encryption . . . 19
  - 2.4.4 Asymmetric Encryption . . . 21
- 2.5 Conclusion . . . 24

I Matrix Multiplication **25**

3 Secure Matrix Multiplication with MapReduce **27**
- 3.1 Introduction . . . 29

3.1.1 Problem Statement . . . 29
3.1.2 Contributions . . . 30
3.1.3 Outline . . . 30
3.2 Matrix Multiplication with MapReduce . . . 30
3.2.1 Matrix Multiplication with Two MapReduce Rounds . . . 31
3.2.2 Matrix Multiplication with One MapReduce Round . . . 32
3.3 Secure Matrix Multiplication with MapReduce . . . 32
3.3.1 Preprocessing for Secure Matrix Multiplication . . . 33
3.3.2 SP Matrix Multiplication with MapReduce . . . 33
3.3.3 CRSP Matrix Multiplication with MapReduce . . . 35
3.3.4 Complexity Comparison . . . 38
3.4 Experimental Results . . . 38
3.4.1 Dataset and Settings . . . 38
3.4.2 Results . . . 38
3.5 Security Proofs . . . 40
3.5.1 Security Proof for CRSP-2R and CRSP-1R Protocols . . . 40
3.5.2 Security Proof for the SP-2R Protocol . . . 43
3.5.3 Security Proof for the SP-1R Protocol . . . 51
3.6 Conclusion . . . 54

4 Secure Strassen-Winograd Matrix Multiplication with MapReduce 55
4.1 Introduction . . . 57
4.1.1 Problem Statement . . . 57
4.1.2 Contributions . . . 57
4.1.3 Outline . . . 58
4.2 Strassen-Winograd Algorithm . . . 58
4.2.1 Strassen-Winograd Algorithm for 2-Power Size Matrices . . . 58
4.2.2 Padding and Peeling: On a Quest for All Dimensions . . . 59
4.3 Strassen-Winograd Matrix Multiplication . . . 60
4.3.1 Strassen-Winograd MapReduce Protocol . . . 60
4.3.2 Strassen-Winograd MapReduce Protocol with the Dynamic Padding Method . . . 63
4.3.3 Strassen-Winograd MapReduce Protocol with the Dynamic Peeling Method . . . 65
4.4 Secure Strassen-Winograd Matrix Multiplication . . . 67
4.4.1 Preprocessing for Secure Strassen-Winograd Matrix Multiplication . . . 67
4.4.2 Secure Approach . . . 69
4.4.3 Secure Strassen-Winograd Matrix Multiplication Protocol . . . 69
4.4.4 Secure Strassen-Winograd Matrix Multiplication with the Dynamic Padding Method . . . 70
4.4.5 Secure Strassen-Winograd Matrix Multiplication with the Dynamic Peeling Method . . . 71
4.5 Experimental Results . . . 73
4.5.1 Dataset and Settings . . . 73
4.5.2 Results . . . 76
4.6 Security Proofs . . . 76
4.6.1 Security Proof for S2M3 Protocol . . . 77
4.6.2 Security Proof for S2M3-Pad Protocol . . . 81
4.6.3 Security Proof for S2M3-Peel Protocol . . . 82
4.7 Conclusion . . . 82

II Relational-Algebra Operations 85

5 Secure Intersection with MapReduce 87
5.1 Introduction . . . 88
5.1.1 Intersection with MapReduce . . . 88
5.1.2 Problem statement . . . 89
5.1.3 Contributions . . . 90
5.1.4 Outline . . . 90
5.2 Standard and Secure Intersection with MapReduce . . . 90
5.2.1 Standard Intersection with MapReduce . . . 91
5.2.2 Secure Intersection with MapReduce . . . 91
5.2.3 Proof of Correctness . . . 93
5.2.4 Complexity of Original and Secure Protocols . . . 95
5.3 Experimental Results . . . 95
5.3.1 Settings . . . 95
5.3.2 Varying the Number of Tuples . . . 96
5.3.3 Varying the Number of Intersected Relations . . . 96
5.4 Security Proof . . . 97
5.4.1 Modeling of our SI Protocol . . . 97
5.4.2 Proof . . . 98
5.5 Conclusion . . . 103

6 Secure Grouping and Aggregation with MapReduce 105
6.1 Introduction . . . 107
6.1.1 Grouping and Aggregation with MapReduce . . . 107
6.1.2 Problem Statement . . . 108
6.1.3 Contributions . . . 109
6.1.4 Outline . . . 109
6.2 Grouping and Aggregation with MapReduce . . . 109
6.2.1 MapReduce Grouping and Aggregation with COUNT Operation . . . 109
6.2.2 MapReduce Grouping and Aggregation with SUM Operation . . . 110
6.2.3 MapReduce Grouping and Aggregation with AVG Operation . . . 110
6.2.4 MapReduce Grouping and Aggregation with MIN Operation . . . 111
6.3 Secure Grouping and Aggregation with MapReduce . . . 112
6.3.1 Preprocessing for SGA Protocols . . . 112
6.3.2 Secure MapReduce Grouping and Aggregation with COUNT Operation . . . 113
6.3.3 Secure MapReduce Grouping and Aggregation with SUM Operation 114
6.3.4 Secure MapReduce Grouping and Aggregation with AVG Operation 114
6.3.5 Secure MapReduce Grouping and Aggregation with MIN Operation 116
6.3.6 Proof of Correctness . . . 116
6.3.7 Complexity of Originals and SGA Protocols . . . 117
6.4 Experimental Results . . . 118
6.4.1 Dataset and Settings . . . 118
6.4.2 Results . . . 118
6.5 Security Proof . . . 119
6.5.1 Modeling of SGA_{SUM} Protocol . . . 119
6.5.2 Security Proof of SGA_{SUM} Protocol . . . 120
6.5.3 Modeling of SGA_{MIN} Protocol . . . 124
6.5.4 Security Proof of SGA_{MIN} Protocol . . . 124
6.6 Conclusion . . . 129

7 Secure Joins with MapReduce **131**
7.1 Introduction . . . 132
7.1.1 Joins with MapReduce . . . 133
7.1.2 Problem statement . . . 135
7.1.3 Contributions . . . 135
7.1.4 Outline . . . 136
7.2 n-ary Joins with MapReduce . . . 136
7.2.1 Cascade Protocol . . . 136
7.2.2 Hypercube Protocol . . . 136
7.3 Secure n-ary Joins with MapReduce . . . 137
7.3.1 Preprocessing for Secure Protocols . . . 137
7.3.2 Secure n-ary Joins with MapReduce and Cascade Protocol . . . 138
7.3.3 Secure n-ary Joins with MapReduce and Hypercube Protocol . . . 139
7.4 Experimental Results . . . 140
7.4.1 Dataset and Settings . . . 140
7.4.2 Results . . . 141
7.5 Security Proofs . . . 142
7.5.1 Security Proof for SCAS Protocol . . . 142
7.5.2 Security Proof for SHYP Protocol . . . 147
7.6 Conclusion . . . 151

8 Conclusion **153**

CHAPTER 1

Introduction

In this thesis, we study distributed computations in the cloud following the MapReduce paradigm. We start this first chapter with a brief history of cryptography, and present cloud cryptography's challenges. Then, we introduce the concept of distributed computations following the MapReduce paradigm. Finally, we list the contributions of this thesis.

1.1 A Brief History of Cryptography

Cryptography, from Ancient Greek κρυπτός (hidden, secret) and γράπηειν (to write) or λογία (study), etymologically refers to the *science of secret*. Historically, its aim is to guarantee the confidentiality of sensitive data, e.g., for military use. For instance, the Spartan army used an encryption technique based on the *scytale* to deliver secret messages during military campaigns. A scytale is a tool used to perform a transposition cipher, consisting of a cylinder with a strip of parchment wound around it on which is written a message. The recipient uses a rod of the same diameter on which the parchment is wrapped to read the message. An other example, is the use of substitution cipher in the 16^{th} century by Mary, Queen of Scots, for the Babington Plot in order to assassinate Queen Elizabeth I[†].

Century after century, techniques have been improved. However, it was not until the World War II, with the Enigma machine used by Nazi Germany, for the appearance of automated encryption techniques allowing to have a faster encryption without human-errors. However, the Enigma machine was not perfect. The Allies, with the help of prodigious mathematicians and linguists (whose Alan Turing), succeeded in the crack of the machine. The consequences on the end of the war were decisive [Koz84].

In the late of 1940s, the mathematician Claude Shannon introduced the theoretical foundations of cryptography by formally defining *secrecy* and *authentication* [Sha49], two fundamental principles of cryptography. This has given rise to the modern cryptography. Few years later, in 1975, the first symmetric encryption standard called *Data Encryption Standard* (DES) has been officially published [SSW77]. A symmetric encryption scheme requires the same secret key to encrypt and decrypt a message. Hence, a sender and a receiver must agree on this key, which can be delicate in real situations. The DES standard has been developed by IBM and used by governments and companies during decades, until it became obsolete in 2000 and was replaced by the new standard called *Advanced*

[†]The history is fond of cryptography, for the curious reader, the author recommends the reading of *The Code Book* by Simon Singh.

Encryption Standard (AES), proposed by Daemen and Rijmen [DR98]. In parallel, Diffie and Hellman invented in 1976, a technique to share a secret between two users with no initial shared knowledge; *asymmetric* cryptography was born [DH76]. Two years later, Rivest, Shamir, and Adelman proposed the first asymmetric encryption scheme allowing two users to safely exchange encrypted messages [RSA78]. They also introduce the concept of homomorphic encryption allowing arbitrary computations on encrypted data. Before the first construction of such a *fully homomorphic encryption* (FHE) scheme proposed by Gentry [Gen09] thirty years later, several partially homomorphic schemes have been designed. A partially homomorphic encryption scheme allows to perform a specific operation from ciphertexts. Such a scheme is the Paillier's cryptosystem [Pai99] that is additive-homomorphic, i.e., it allows to compute the encryption of the sum of two plaintexts from their respective encryption.

Naturally, FHE schemes are a powerful tool in cloud computing. However, currently known FHE schemes are yet impractical for real applications [CGGI16]. In recent years, *somewhat homomorphic encryption* (SHE) schemes, initially used as a building block for FHE construction have attracted a lot of attention from the community [BGV12, FV12a, BLLN13, LTV12]. Although SHE schemes can support only a limited number of additions and multiplications, they are much faster and more compact than FHE schemes. Therefore SHE schemes can give a practical solution in wide applications, and it is coming to attention to research on applications with SHE schemes [NLV11].

If cryptography was mainly used by governments and companies before the 20^{th} century, the emergence of computers and of the Internet in our everyday life allows us to be actors (or at less users) of it. In fact, the cryptography allows Internet users to seclude themselves, or information about themselves, and thereby express themselves selectively: what is called privacy[‡]. One of the first solution is *Pretty Good Privacy* [CDF+07], an encryption program developed by Paul Zimmermann in 1991 that provides privacy and authentication for data communication. It is used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications. Its free-software replacement named GnuPG [Pro19] can be used easily by everyone to encrypt and sign e-mails via the Thunderbird's extension Enigmail [SB19]. Moreover, since Edward Snowden's revelations in 2013[§] and various data scandals including the one with Facebook and Cambridge Analytica in early 2018[¶], the general public realized that it was important to take control of its digital life with the help of cryptographic tools. In addition to the encryption of e-mails, the Tor web browser [TTP19] solution is more and more adopted by journalists or regular internauts to browse privately.

All this proves the importance of the cryptography in history and societal and political problems of today's world, as well as the various applications of this *science of secret*.

1.2 Cloud Computing Challenges

As seen above, cryptography can be used for different goals: confidentiality, authentication, anonymity, etc. This thesis focuses on the privacy-preserving cloud applications.

During the last decade, a plethora of cloud computing service providers appeared, e.g., Amazon Web Services, Microsoft Azure, Google Cloud Platform, or IBM Cloud. They propose to manage data centers to the users in order they have only to deal with the

[‡]What Is Privacy? https://www.privacyinternational.org/video/1625/video-what-privacy

[§]For further details, the author recommends the documentary film *Citizenfour* directed by Laura Poitras, and the non-fiction book *No Place to Hide: Edward Snowden, the NSA, and the U.S. Surveillance State* by Glenn Greenwald.

[¶]https://www.theguardian.com/news/series/cambridge-analytica-files

application-side. On the positive side, using such a cloud computing provider makes data processing accessible to users who cannot afford to build their own clusters. A recent real-world example is the use of Microsoft Azure by the European search engine Qwant[‖] to compute a part of the web indexation[*]. However, outsourcing data and computations to a cloud computing provider involves inherent security and privacy concerns. Indeed, such a computation delegation invokes two threats.

1. A cloud computing service provider can receive data to process in a plain form. Hence, it is able to learn information (possibly sensitive) on its clients.

2. Cloud users must trust the infrastructure's security for the safety of their data. Indeed, if some security breaches exist, an adversary can exploit them and steal the plain data.

Furthermore, more and more Internet applications deal with a huge amount of data, these applications are called data-mining applications or "big-data" analysis. In practice, the processing of the data must be fast. In many of these applications, the structure of the data is regular allowing to use parallelism in order to manage the data quickly. Classic computation uses only one processor, with its main memory, cache, and local disk. In the past, parallel processing, such as large scientific computations, was performed on very expensive and specific machine with many processors and specialized hardware. Due to the prevalence of large-scale Web services and the cost of such machine, computations are more and more performed on thousands of compute nodes connected by Ethernet cables and switches. These new computing facilities have given rise to a new generation of programming systems. These systems take advantage of the power of parallelism and at the same time avoid the reliability problems that arise when the computing hardware consists of thousands of independent components, any of which could fail at any time.

In summary, new cloud applications have to deal, on the one hand, with the software side by considering the privacy of data user, and on the other hand, with the hardware side by considering the architecture of the data centers.

1.3 The MapReduce Paradigm

This thesis focus on the MapReduce paradigm to deal with the cloud service provider's architecture composed of thousands of compute nodes. To deal with applications on such architectures, a new software stack has evolved. The software stack begins with a new form of file system, called a *distributed file system* which features much larger units than the disk blocks in a conventional operating system. Distributed file systems also provide replication of data or redundancy to protect against media failures that occur when data is distributed over thousands of low-cost compute nodes. Before explaining how computations are performed using the MapReduce paradigm, we recall some elements of databases domain.

1.3.1 Vocabulary and Notations

MapReduce deals with *relations*. A relation is a table with column headers called *attributes*. Rows of the relation are called *tuples*. The set of attributes of a relation is called its *schema*. We represent by $R(A_1, \ldots, A_n)$ a relation whose name is R and having $A_1, \ldots, A_n$ as attributes. We give such an example in Figure 1.1.

[‖] https://www.qwant.com/

[*] https://standblog.org/blog/post/2019/05/19/Qwant-Microsoft-et-Vivatech

From	*To*
url11	*url12*
url21	*url22*
url31	*url32*
...	...

Figure 1.1: Relation *Links*(*From*, *To*) consists of the set of pairs of URLs, such that the first has one or more links to the second.

1.3.2 MapReduce

MapReduce is a style of computing. It has been developed by Google in 2004 in order to efficiently compute the ranking of Web pages [DG04]. Moreover, it has been implemented in different systems as, Google's internal implementation (simply called MapReduce) and the open-source implementation Apache Hadoop® [Fou19b], used in this thesis. Apache Hadoop® is widely used by companies such as Facebook with a 1,100-machine cluster for analytics and machine learning or Spotify with a 1,650-machine cluster for data aggregation[**]. These implementations allow to manage many large-scale computations that are tolerant to hardware faults. Only two functions, called *Map* and *Reduce*, must be implemented while the system manages the parallel execution, coordination of tasks that execute Map or Reduce, and also deals with the possibility that one of these tasks fails to execute. To sum up, MapReduce computations execute as follows:

1. Map tasks are given one or more chunks. The Map tasks transform the chunk into a sequence of *key-value* pairs depending on the code written in the Map function.

2. The key-value pairs emitted by each Map task are aggregated by a *master controller* and sorted by key. Key-value pairs sharing the same key are then sent to the same Reduce task.

3. The Reduce tasks work on one key at a time and combine all the values associated with that key depending on the code written in the Reduce function.

We illustrate this mechanism in Figure 1.2.

The Map Tasks. Map tasks take as input *elements* which can be a tuple or a document. A chunk is a collection of elements. Inputs and outputs of Map and Reduce functions are of the key-value pair form. The Map function written by the user takes as input element as its argument and produces zero or more key-value pairs. The types of keys and values are each arbitrary. We stress that keys do not have to be unique, in fact Map task can produce several key-value pairs with the same key, even from the same element.

Grouping by Key. Once the Map tasks have finished successfully, all generated key-value pairs are grouped by key. For a given key, all associated values produce a list of values. Regardless of what the Map and the Reduce tasks do, the grouping is produced by the system. Depending on the number of Reduce tasks there will be, the MapReduce system picks a hash function that applies to keys and produces the corresponding number of buckets. Hence, each key outputted by the Map tasks is hashed and its key-value pair is sent to the Reduce tasks according to the hash value. We stress that the used hash function does not have to be a cryptographic hash function. To perform the grouping

[**] https://wiki.apache.org/hadoop/PoweredBy

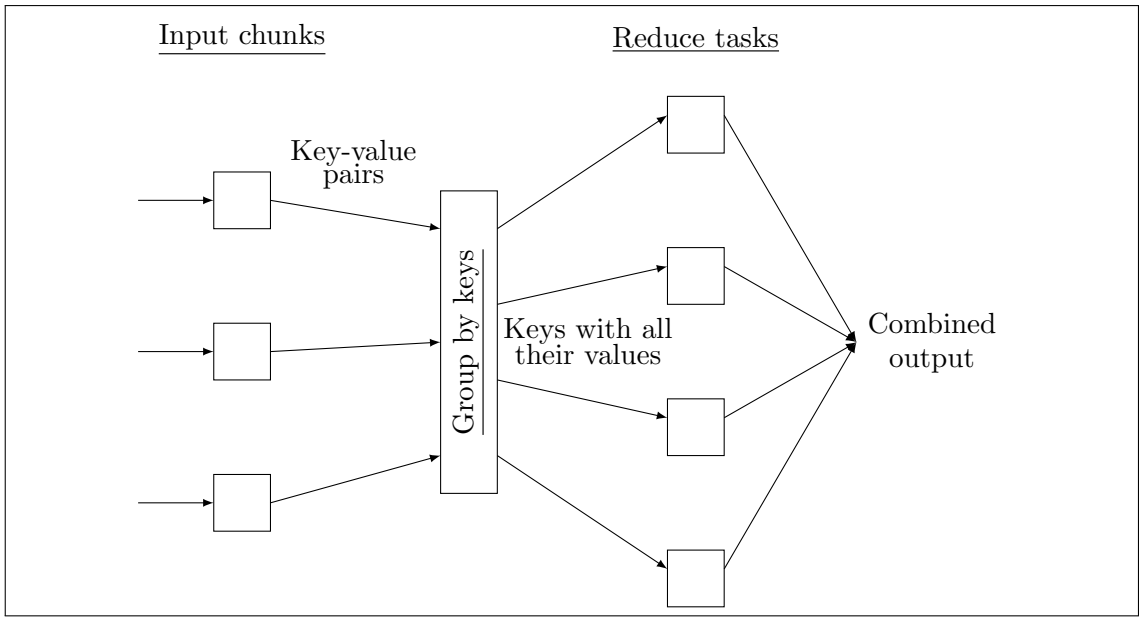


Figure 1.2: Schematic of a MapReduce computation.

by key and distribution to the Reduce tasks, the master controller merges the files from each Map task that are destined for a particular Reduce task and feeds the merged file to that process as a sequence of key-list-of-values pairs. In other words, for each key k, the input to the Reduce task that handles key k is a pair of the form $(k, [v_1, \ldots, v_n])$, where $(k, v_1), \ldots, (k, v_n)$ are the key-values pairs coming from the Map tasks.

The Reduce Tasks. The Reduce function's input is a pair constituted of a key and a list of associated values. The output of the Reduce function is a sequence of zero or more key-values pair. We refer to the application of the Reduce function to a single key and its associated list of values as a *reducer*. A Reduce task receives one or more keys and their associated value lists. In other terms, a Reduce task can execute several reducers. When the Reduce tasks have finished, all the outputs are merged into a single file and stored on the file system of the cloud computing service provider.

1.4 Algorithms Using MapReduce

In this thesis, we focus on two categories of computation: the first one concerns the matrix multiplication, and the second one relates to the relational-algebra operations.

1.4.1 Matrix Multiplication

If M is a matrix with element $m_{i,j}$ in row i and column j, and N is a matrix with element $n_{j,k}$ in row j and column k, then the product MN is the matrix P with element $p_{i,k}$ in row i and column k, where

$$p_{i,k} := \sum_j m_{i,j} \cdot n_{j,k} \ .$$

Matrix multiplication is a mathematical tool of various problems spanning over a plethora of domains, e.g., statistical analysis, medicine, image processing, machine learning or web ranking. Indeed, Markov chains applications on genetics and sociology [Cha12], or applications such that computation of shortest paths [SZ99, Zwi98], convolutional neural network [KSH17] deal with data processed as matrix multiplication. In such applications,

the size of the matrices to be multiplied is often very large. The matrix multiplication is also the original purpose for which the Google implementation of MapReduce was created. Indeed, such multiplications are needed by Google in the computation of the PageRank algorithm [DG04].

1.4.2 Relational-Algebra Operations

This thesis focuses on the three following relational-algebra operations.

1. *Intersection.* This well-known set operation applies to the sets of tuples in two or more relations that have the same schema.

2. *Grouping and Aggregation.* Tuples of a relation are partitioned according to their values in one set of attributes, called the *grouping attributes*. For each group, the values in certain other attributes, called the *aggregation attributes*, are aggregated. The normally permitted aggregation operations are SUM, COUNT, AVG, MIN, and MAX, with the natural meanings. Note that SUM and AVG require that the type allows arithmetic operations, while MIN and MAX operations require that the aggregated values have a type that can be compared. The results of this aggregation operation is one tuple for each group. That tuple has a component for each of the grouping attributes, with the value common to tuples of that group. It also has a component for each aggregation attribute, with the aggregated value for that group.

3. *Natural Join.* Given two relations, this operation compares each pair of tuples, one from each relation, on all the attributes that are common to the two schemas. If values of these attributes agree, then it produces a tuple that has components for each of the attribute in either schema and agrees with the two tuples on each attribute.

These relational-algebra operations have obvious real-world applications.

1.5 Contributions

This thesis addresses the problem of how to perform secure distributed matrix multiplication and relational-algebra operations in a public cloud using the MapReduce paradigm. We propose the following contributions.

- In the first part of this manuscript, we consider matrix multiplication. We propose two secure approaches [BCGL17] enhancing the standard MapReduce matrix multiplication [LRU14] with privacy preservation (cf. Chapter 3). Based on this work, we propose a protocol to compute the Strassen-Winograd matrix multiplication, one of the most efficient matrix multiplication algorithm, using the MapReduce paradigm. Moreover, we design a secure approach allowing to compute the Strassen-Winograd matrix multiplication in a privacy-preserving way using the MapReduce paradigm [CGLY19b] (cf. Chapter 4). Our new protocols rely on the Paillier's cryptosystem [Pai99].

- In the second part, we deal with relational-algebra operations, in particular intersection, grouping and aggregation, and natural join operations. For each of these three operations, we design a privacy-preserving approach [CGLY19a, CGLY18, BCG+18] of the corresponding standard MapReduce protocol [LRU14]. We respectively present these three secure improvements in Chapter 5, 6, and 7.

All protocols assume a *semi-honest* cloud service provider, i.e., it dutifully executes the considered protocol but tries to learn as much information as possible. Moreover, we also assume that all communications are performed over secure channels, i.e., transferred data is resistant to eavesdropping.

Finally, we give experimental results for each protocol using the open-source framework Apache Hadoop® 3.2.0 [Fou19b] implementing the MapReduce paradigm. All computations have been done on a computer having an Intel® Core™ i7-4790 CPU cadenced at 3.60 GHz, 16 GB of RAM, and running on the GNU/Linux distribution Ubuntu 16.04. We use the Hadoop streaming interface and implement the Map and Reduce functions in Go 1.6.2 [GPT10].

1.6 Related Work

We present the related work on both parties of the manuscript, namely matrix multiplication and relational-algebra operations.

1.6.1 Matrix Multiplication

Distributed matrix multiplication has been thoroughly investigated in the secure multi-party computation model (MPC) [DA01, DLOP16, DLOP17, DLF+19, AE07, WSZ+09], whose goal is to allow different nodes to jointly compute a function over their private inputs without revealing them. The aforementioned works on secure distributed matrix multiplication have different assumptions compared to the MapReduce paradigm: (i) they assume that nodes contain entire vectors, whereas the division of the initial matrices in chunks as done in MapReduce does not have such assumptions, and (ii) in MapReduce, the functions specified by the user [DG04] are limited to Map (process a key-value pair to generate a set of intermediate key-value pairs) and Reduce (merge all intermediate values associated with the same intermediate key), and the matrix multiplication is done in one or two communication rounds [LRU14]. On the other hand, the works in the MPC model assume arbitrary numbers of communication rounds, relying on more complex functions than Map and Reduce. Moreover, generic MPC protocols [MD08, CDN01] allow several nodes to securely evaluate any function. Such protocols could be used to secure MapReduce. However, due to their generic nature, they are inefficient and require a lot of interactions between parties. Our goal is to design optimized protocols to secure the standard and the Strassen-Winograd matrix multiplication using the MapReduce paradigm, and the additive-homomorphic Paillier's cryptosystem [Pai99].

In a different way, homomorphic encryption allows to perform operations on encrypted data. The concept of homomorphic encryption has been introduced by Rivest et al. in 1978 [RSA78] while the first scheme of fully homomorphic encryption (FHE) that supports arbitrary operations on encrypted data was proposed by Gentry in 2009 [Gen09]. Since this pioneer work, a lot of effort has been done to propose and improve FHE schemes in both theory and practice [ACC+18]. At present, some ring-LWE-based FHE schemes such as BGV [BGV12] or FV [FV12a] and its variant BFV are efficient and useful for matrix multiplication. However, when several matrix multiplications are performed between $k \geq 2$ square matrices $M^{(i)}$ (with $i \in [\![1, k]\!]$) of order $n \in \mathbb{N}^*$, i.e., the computation of

$$M_{j_0,j_k} := \sum_{j_{k-1}=1}^{n} \left(\cdots \left(\sum_{j_2=1}^{n} \left(\sum_{j_1=1}^{n} M^{(1)}_{j_0,j_1} \cdot M^{(2)}_{j_1,j_2} \right) \cdot M^{(3)}_{j_2,j_3} \right) \cdots \right) \cdot M^{(k)}_{j_{k-1},j_k} ,$$

where M_{j_0,j_k} is the matrix element of the result of coordinates (j_0, j_k) with $j_0, j_k \in [\![1, n]\!]$, the performance might be considerably slower since it implies the use of huge parameters or techniques such as relinearization and modulus switching [BV11]. Our methods use

Paillier's cryptosystem [Pai99] that is more practical for multiple matrix multiplications. In Figure 1.3, we show the running time for multiple matrix multiplications using the Paillier's cryptosystem and the BFV scheme. We use a 1024-bit RSA modulus for the Paillier's cryptosystem while parameters for the BFV scheme are presented in Table 1.1.

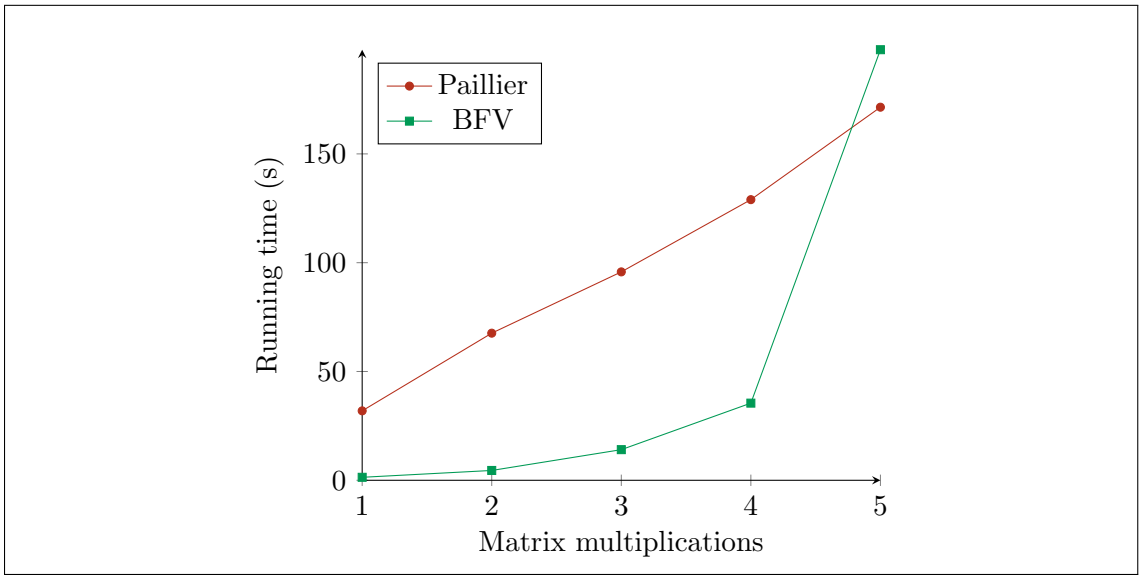


Figure 1.3: Running time vs the number of matrix multiplications using Paillier's cryptosystem [Pai99] and the BFV [FV12a] scheme.

Table 1.1: Parameters for the BFV scheme to compute multiple matrix multiplications.

# matrix mul.	Degree of polynomial	Coefficient mod.	Plaintext mod.
1	2048	2048	8
2	2048	4096	8
3	4096	4096	12
4	4096	8192	16
5	8192	16384	19

On the other hand, several optimization for matrix multiplication over ring-based homomorphic encryption have been realized. Based on the idea of Yasuda et al. [YSK+15a, YSK+15b], Duong et al. [DMY16] proposed a new method consisting in packing an entire matrix into a single polynomial and then encrypt the polynomial over the homomorphic scheme. Following this work, Mishra et al. [MRDY18] generalized the method of Duong et al. to multiple matrix multiplications. If these methods reduce both the ciphertext size and the computation cost, they remain difficult to apply with big-data matrices which require large parameters.

1.6.2 Relational-Algebra Operations

Since the seminal MapReduce paper [DG04], different protocols have been proposed to perform relational-algebra operations in a privacy-preserving manner [DDGS16] such as search [BPMÖ12, MBC13], count [VBN15] or joins [DLS16]. We present related work on the intersection, grouping and aggregation, and natural join relational-algebra operations on which we focus in this thesis.

Intersection. Private Set Intersection (PSI) is a well-known cryptographic primitive where

two parties compute the intersection of their respective sets, such that minimal information is revealed in the process. It was introduced by Freedman et al. [FNP04]. The aim of this primitive is to allow the two parties to learn the elements common to both sets and nothing else. Such primitives where neither party has any advantage over the other and where all parties know the intersection are called *mutual PSI* [CKT10]. On the contrary, primitives where only one party learns the intersection of the two sets while the other learns nothing are called *one-way PSI* [CKT10]. A natural PSI extension is called *PSI with Data Transfer* (PSI-DT) [JL10]. In PSI-DT, one or both parties have data associated with each element of their respective sets.

Many secure MPC protocols computing the set intersection have been proposed [KS05, HN10, HEK12, DCW13] in the literature. However, each protocol follows either the mutual or the one-way model, and do not consider a model where an external user may receive the intersection, which our protocol proposes. Also, using the MapReduce paradigm, our proposal only requires the two functions Map and Reduce.

Grouping and Aggregation. Bonawitz et al. [BIK+17] provides a technique to compute secure aggregation, while relying on Shamir's secret sharing [Sha79] to compute the sum of values coming from different sources. Similarly, Alghamdi et al. [AWK17] provides a technique to compute secure aggregation for wireless sensor networks. Contrary to us, these two approaches do not consider the MapReduce paradigm and they cannot be easily adapted for MapReduce because values of shared attributes are encrypted in a non-deterministic way. This is not a suitable choice for MapReduce keys that need to be equal in order to aggregate the key-value pairs on the same reducer.

Moreover, Dolev et al. [DLS16] proposed a technique for executing MapReduce on the count aggregation in a public cloud while preserving data owner privacy. They use the Shamir's secret sharing and accumulating automata [DGL15]. The count computation is done on secret-shares in the public cloud, and at the end, the user performs the interpolation on the outputs. On the other hand, in our setting, the user has only to decrypt the final query result, contrary to the need of doing interpolations in the paper of Dolev et al. [DGL15].

In a different way, several papers use trusted hardware-based systems to securely compute aggregation [TNP16, DSC+15, SCF+15]. This thesis does not rely on such system but studies security on the application-side.

Natural Join. Emekçi et al. [EAEG06] proposed protocols to perform joins in a privacy-preserving manner using Shamir's secret sharing [Sha79]. Contrary to us, they do not consider the MapReduce paradigm and their approach cannot be trivially adopted in MapReduce because values of shared attributes are encrypted in a non-deterministic way. Indeed, the MapReduce paradigm requires that data satisfying a certain property must be mapped to the same reducer which is not possible with a probabilistic encryption scheme. In the same way, Laur et al. [LTW13] also proposed a protocol to compute joins using secret sharing, in addition to only consider two relations, their approach cannot be easily used with the MapReduce paradigm. Chow et al. [CLS09] introduced a generic model that uses two non-colluding servers to perform join computation between an arbitrary number relations in a privacy-preserving manner but did not consider the MapReduce paradigm. Moreover, we assume a more general setting where the public cloud servers collude.

Finally, Popa et al. [PRZB11] proposed a system, called CryptDB, that is closely related to our work. CryptDB provides practical and provable confidentiality for applications backed by SQL databases. It works by executing SQL queries over encrypted data, such as natural join, aggregation, and search. Contrary to us, they do not consider the intersection operations and do not follow MapReduce programming model.

1.7 Publications

We recall the papers about the works presented in this manuscript, and we list the papers about other works that were conducted throughout this thesis.

1.7.1 Presented in this Manuscript

The five following papers summarize the works that are presented in this manuscript.

1. *Secure Matrix Multiplication with MapReduce*
   By Xavier Bultel, Radu Ciucanu, Matthieu Giraud, and Pascal Lafourcade. Published in proceedings of the 12th International Conference on Availability, Reliability and Security (ARES 2017) [BCGL17].

2. *Secure Strassen-Winograd Matrix Multiplication with MapReduce*
   By Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, and Lihua Ye. Published in proceedings of the 16th International Joint Conference on e-Business and Telecommunications (SECRYPT 2019) [CGLY19b].

3. *Secure Intersection with MapReduce*
   By Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, and Lihua Ye. Published in proceedings of the 16th International Joint Conference on e-Business and Telecommunications (SECRYPT 2019) [CGLY19a].

4. *Secure Grouping and Aggregation with MapReduce*
   By Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, and Lihua Ye. Published in proceedings of the 15th International Joint Conference on e-Business and Telecommunications (SECRYPT 2018) [CGLY18].

5. *Secure Joins with MapReduce*
   By Xavier Bultel, Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, and Lihua Ye. Published in proceedings of the 11th International Symposium on Foundations and Practice of Security (FPS 2018) [BCG+18].

1.7.2 Other Publications

We list exhaustively the other papers published during the thesis, and we give a short abstract for each of them.

1. *Formal Analyze of a Private Access Control Protocol to a Cloud Storage*
   By Mouhebeddine Berrima, Matthieu Giraud, Pascal Lafourcade, and Narjes Ben Rajeb. Published in proceedings of the 14th International Joint Conference on e-Business and Telecommunications (SECRYPT 2017) [BLGR17].

   Abstract: Storing data in the cloud makes challenging data's security and users' privacy. To address these problems cryptographic protocols are usually designed. Cryptographic primitives have to guarantee some security properties such as data and user privacy or authentication. *Attribute-Based Signature* (ABS) and *Attribute-Based Encryption* (ABE) are very suitable for storing data on an untrusted remote entity. In this work, we formally analyze the Ruj et al. [RSN12] protocol of cloud storage based on ABS and ABE schemes. We model the protocol and its security properties with ProVerif, an automatic tool for the verification of cryptographic protocols. We discover an unknown attack against user privacy. We propose a correction, and automatically prove the security of the corrected protocol with ProVerif.

2. *Practical Passive Leakage-Abuse Attacks Against Symmetric Searchable Encryption*
By Alexandre Anzala-Yamajako, Olivier Bernard, Matthieu Giraud, and Pascal Lafourcade. Published in proceedings of the 14th International Joint Conference on e-Business and Telecommunications (SECRYPT 2017) [GABL17].

   Abstract: *Symmetric Searchable Encryption* (SSE) schemes solve efficiently the problem of securely outsourcing client data with search functionality. These schemes are provably secure with respect to an explicit leakage profile; however, determining how much information can be inferred in practice from this leakage remains difficult. First, we recall the leakage hierarchy introduced in 2015 by Cash et al. [CGPR15]. Second, we present complete practical attacks on SSE schemes of L4, L3, and L2 leakage profiles which are deployed in commercial cloud solutions. Our attacks are passive and only assume the knowledge of a small sample of plaintexts. Moreover, we show their devastating effect on real-world data sets since, regardless of the leakage profile, an adversary knowing a mere 1% of the document set is able to retrieve 90% of documents whose content is revealed over 70%. Then, we further extend the analysis of existing attacks to highlight the gap of security that exists between L2- and L1-SSE and give some simple countermeasures to prevent our attacks.

3. *Verifiable Private Polynomial Evaluation*
By Xavier Bultel, Manik Lal Das, Hardik Gajera, David Gérault, Matthieu Giraud, and Pascal Lafourcade. Published in proceedings of the 11th International Conference on Provable Security (ProvSec 2017) [BDG+17].

   Abstract: Delegating the computation of a polynomial to a server in a verifiable way is challenging. An even more challenging problem is ensuring that this polynomial remains hidden to clients who are able to query such a server. In this work, to tackle this problem, we formally define the notion of *Private Polynomial Evaluation* (PPE) and present rigorous security models, along with relations between the different security properties. Particularly, we define the "indistinguishability against chosen function attack" (IND-CFA) where the attacker tries to guess which polynomial is used among two polynomials of his choice. Then we show that the existing schemes of the literature are not IND-CFA secure. We are able to break two of them by learning the secret polynomial used by the server. Finally, we design PIPE, the first IND-CFA secure PPE scheme and prove its IND-CFA security under the decisional Diffie-Hellman assumption in the random oracle model.

4. *No Such Thing as a Small Leak: Leakage-Abuse Attacks Against Symmetric Searchable Encryption*
By Alexandre Anzala-Yamajako, Olivier Bernard, Matthieu Giraud, and Pascal Lafourcade. Published in volume 990 of the Communications in Computer and Information Science journal (CCIS 2019) [ABGL17].

   Abstract: *Symmetric Searchable Encryption* (SSE) schemes enable clients to securely outsource their data while maintaining the ability to perform keywords search over it. The security of these schemes is based on an explicit leakage profile, has initiated the investigation into how much information could be deduced in practice from this leakage. In this work, after recalling the leakage hierarchy introduced in 2015 by Cash et al. [CGPR15] and the passive attacks of Giraud et al. [GABL17] on SSE schemes, we demonstrate the effectiveness of these attacks on a wider set of real-world datasets than previously shown. On the other hand, we show that the attacks are inefficient against some types of datasets. Finally, we use what we learn from the unsuccessful datasets to give insight into future countermeasures.

5. *Security Analysis and Psychological Study of Authentication Methods with Personal*

Identifier Number Codes
By Xavier Bultel, Jannik Dreier, Matthieu Giraud, Marie Izaute, Timothée Kheyrkhah, Pascal Lafourcade, Dounia Lakhzoum, Vincent Marlin, and Ladislav Moták. Published in proceedings of the 12th International Conference on Research Challenges in Information Science (RCIS 2018) [BDG+18].

Abstract: Touch screens have become ubiquitous in the past few years, for instance in smartphones and tablets. These devices are often the entry door to numerous information systems, hence having a secure and practical authentication mechanism is crucial. In this work, we examine the complexity of different authentication methods specifically designed for such devices. We study the widely spread technology to authenticate a user using a Personal Identifier Number (PIN) code. Entering the code is a critical moment where there are several possibilities for an attacker to discover the secret. We consider the three attack models: a *Bruteforce Attack* model, a *Smudge Attack* model, and an *Observation Attack* model where the attacker sees the user logging in on his device. The aim of the intruder is to learn the secret code. Our goal is to propose alternative methods to enter a PIN code. We compare such different methods in terms of security. Some methods require more intentional resources than other, this is why we performed a psychological study on the different methods to evaluate the users' perception of the different methods and their usage.

1.8 Outline

In Chapter 2 we introduce the notations and mathematical notions we use through the thesis, as well as the cryptographic primitives that we use and their security properties. In Chapter 3, we present our secure protocols to compute standard matrix multiplication using the MapReduce paradigm. The research problem that these protocols aim to solve is how to combine practical distributed computation of matrix multiplication and privacy. In Chapter 4, we pursue studies on distributed computation of secure matrix multiplication and focus on the Strassen-Winograd algorithm, one of the most efficient algorithms to compute matrix multiplication. This gives rise to two MapReduce protocols. In Chapter 5, we relational-algebra operations by considering the intersection between relations. We propose a privacy-preserving MapReduce protocol that computes intersection of an arbitrary number of relations. Chapter 6 presents privacy-preserving protocols for grouping and aggregations operations using the MapReduce paradigm. In particular, we focus on the COUNT, SUM, AVG, MIN, and MAX aggregations. Finally, in Chapter 7, we present a secure MapReduce approach, for both state-of-the-art algorithms computing the natural join between relations, namely the cascade and the hypercube algorithms.

CHAPTER 2

Technical Introduction

We start by presenting the notations that we use throughout this thesis and recall some required mathematical background. Then, we present cryptographic tools and notions. Finally, based on these tools and notations, we define cryptographic primitives along with their corresponding security properties that will be used in the first and second part of this thesis.

Contents

2.1 Notations **14**
2.2 Mathematical Background **14**
2.3 Cryptographic Background **15**
2.3.1 Cryptographic Tools 15
2.3.2 Hardness Assumptions 17
2.4 Cryptographic Primitives **18**
2.4.1 Hash Function 18
2.4.2 Pseudo-Random Function 19
2.4.3 Symmetric Encryption 19
2.4.4 Asymmetric Encryption 21
2.5 Conclusion **24**

2.1 Notations

We define the notations that we use through the thesis.

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

2.2 Mathematical Background

We recall the definition of a prime number playing a fundamental role in cryptography.

Definition 1 (Prime number [HW08])**.** *Let $p \in \mathbb{N}^*$, p is a prime number if $p > 1$ and if p cannot be divided by any other number except itself and the number 1.*

We now recall some fundamental algebra objects playing an important role in cryptography, namely groups and cyclic groups.

Definition 2 (Group [Gri15])**.** *A* group *is a couple $(\mathbb{G}, \cdot)$ where $\mathbb{G}$ is a set of elements and $\cdot$ is a binary operation of $\mathbb{G}^2 \to \mathbb{G}$ that combines any $(a, b) \in \mathbb{G}^2$ to an element of $\mathbb{G}$ denoted $a \cdot b$ such that the three following properties hold:*

1. Associativity. *For all $(a, b, c) \in \mathbb{G}^3$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*
2. Identity element. *There exists a unique element of $\mathbb{G}$ denoted $\mathbb{1}$, and called the* identity element*, such that for all $a \in \mathbb{G}$, $\mathbb{1} \cdot a = a \cdot \mathbb{1} = a$.*
3. Inverse element. *For all $a \in \mathbb{G}$, there exists a unique element $b \in \mathbb{G}$ denoted a^{-1} such that $a \cdot a^{-1} = a^{-1} \cdot a = \mathbb{1}$.*

Moreover, we way that a group is commutative *(or* abelian*) when for any $(a, b) \in \mathbb{G}^2$, it holds that $a \cdot b = b \cdot a$. A group is said to be* finite *when $\mathbb{G}$ has a finite number of elements. The number of elements of $\mathbb{G}$ is called the* order *of the group and is denoted by $|\mathbb{G}|$. When it is clear in the context, we denote by $\mathbb{G}$ the group $(\mathbb{G}, \cdot)$.*

Definition 3 (Subgroup [Gri15])**.** *Let $(\mathbb{G}, \cdot)$ be a group. Let $\mathbb{H}$ be a subset of $\mathbb{G}$, $\mathbb{H}$ is called a* subgroup *of $\mathbb{G}$ if $\mathbb{H}$ also forms a group under the operation $\cdot$.*

Some finite groups have the particularity they can be described using only one element of the group. Such a group is called *cyclic* while the element describing the group is called the *generator*. We give the definition of such a cyclic group below.

Definition 4 (Cyclic group and generator [Gri15])**.** *Let $(\mathbb{G}, \cdot)$ be a finite group. Without the loss of generality, we denote by g^x the x applications of $\cdot$ over g where $g \in \mathbb{G}$ and $x \in \mathbb{Z}$ such that*

$$g^x = \underbrace{g \cdot g \cdot \cdots \cdot g}_{x \text{ times}} .$$

We say that g is a generator of $(\mathbb{G}, \cdot)$ when $\mathbb{G} := \{g^x : x \in [\![1, |\mathbb{G}|]\!]\}$. A group having a generator is called a cyclic group*.*

In this thesis, we consider multiplicative groups, i.e., the binary operator $\cdot$ corresponds to the multiplication. Note that we indistinctly use the symbol $\cdot$ for multiplication between elements of a multiplicative group and for the scalar multiplication while we omit for the multiplication between matrices.

2.3 Cryptographic Background

In this section, we recall the cryptographic tools, the cryptographic assumptions, and the cryptographic primitives used throughout this thesis.

2.3.1 Cryptographic Tools

We start by recalling some cryptographic tools.

Negligible Function

We start by defining the notion of negligible function used throughout this thesis.

Definition 5 (Negligible function [KL14])**.** *A function $\mu : \mathbb{N} \to \mathbb{R}^+$ is called* negligible *if for every positive polynomial $p(\cdot)$ there exists $N_p \in \mathbb{N}$ such that for all integers $x > N_p$, we have $\mu(x) < 1/p(x)$.*

Security Parameter and Adversaries

In order to formalize security notions, we need to bound the computing power of an adversary. Indeed, an arbitrary adversary can always break cryptosystems using a large enough computer and spending an exponential amount of time. We first define a *polynomial-time* algorithm.

Definition 6 (Polynomial-Time Algorithm [KL14])**.** *Let $\lambda \in \mathbb{N}$. An algorithm $\mathcal{A}$ is said to run in* polynomial-time *if there exists a polynomial $p(\cdot)$ such that for every input $x \in \{0,1\}^\lambda$, the execution time of $\mathcal{A}(x)$ is bounded by $p(\lambda)$ steps.*

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

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

Experiments

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

In order to have concrete adversary model, the adversary may have access to black-box algorithms (sometimes with some restrictions), called *oracles*. An oracle allows the adversary to learn some information that she cannot obtain by herself in order to solve the experiment. For instance, an oracle can be an algorithm that decrypts some ciphertexts using a key that the adversary does not know. This oracle models the famous *lunch time attack* [CS98]. At the end of the experiment, the output of the adversary is defined as the output of the experiment. In this thesis, we denote by $\mathcal{A}^{\mathsf{Oracle}}$ to mean that the adversary $\mathcal{A}$ has access to the oracle denoted Oracle.

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

Computational Indistinguishability

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

Definition 7 (Computational indistinguishability [Lin17])**.** *Let $X_0 := \{X(I,\lambda)\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}$ and $Y_0 := \{Y(I,\lambda)\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}$ be two distribution ensembles. We say that X_0 and Y_0 are* computationally indistinguishable *if for every probabilistic polynomial algorithm D outputting a single bit, there exists a negligible function $\mu(\cdot)$ such that for every $I \in \mathcal{I}$ and every $\lambda \in \mathbb{N}$, we have*

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

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

Simulation-based Proofs

Many of the security proofs given in this thesis follow the ideal/real simulation paradigm introduced by Goldreich et al. [GMW87]. In other terms, proofs are based on the indistinguishability of two different distribution ensembles X^0 and X^1. However, in many cases it is infeasible to directly prove this indistinguishability. Instead, we use the *hybrid argument* consisting in the construction of *simulators* that generate a sequence of distributions ensembles, starting with X^0, and ending with X^1. Then, we prove that consecutive distribution ensembles are indistinguishable. The indistinguishability between X^0 and X^1 is therefore obtained by transitivity.

Secure Multiparty Computation

Some cryptographic protocols involve several participants, called *parties*, in order to jointly compute a function over their inputs while keeping those inputs private. This model is called *multiparty computation* and was formally introduced by Yao [Yao82]. Unlike traditional cryptosystems where the adversary is outside of the system and tries, for instance, to break the confidentiality or the integrity of communication, the adversary in this model controls one of the parties. In this thesis, we only consider *semi-honest* (or *honest-but-curious*) adversaries [Gol01]. Such an adversary controls one of the parties and follows the protocol specification exactly. However, it may try to learn more information than allowed by looking at the transcript of message that it received and its internal state. A protocol that is secure in the presence of semi-honest adversaries does guarantee that there is no *inadvertent leakage* of information.

Intuitively, a multiparty protocol is secure if whatever can be computed by a party participating in the protocol can be computed based on its input and output *only*. This idea is formalized according to the simulation paradigm by requiring the existence of a *simulator* who generates the *view* of a party, i.e., all values received, computed, and sent by this party during an execution of the protocol. More formally, the view is defined as follows:

Definition 8 (View [Lin17])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter, and π be a n-party protocol. The view of the party P_i, for all $i \in [\![1,n]\!]$, during an execution of π on $I = (I_i)_{i\in[\![1,n]\!]}$ is denoted $\mathsf{view}^{\pi}_{P_i}(I,\lambda)$ and equals (I_i, M_i, O_i), where I_i is the input of P_i, M_i represents messages sent by other parties and received by P_i, and O_i is the output of P_i computed from I_i and M_i during the protocol execution.*

We denote by $\mathsf{view}^{\pi}_{P_i,P_j}(I,\lambda) = (\mathsf{view}^{\pi}_{P_i}(I,\lambda), \mathsf{view}^{\pi}_{P_j}(I,\lambda))$*, with* $i,j \in [\![1,n]\!]$*, the joint view of a collusion between parties* P_i *and* P_j*.*

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

We now formally define the security of a multiparty protocol with respect to static semi-honest adversaries.

Definition 9 (Security with respect to static semi-honest adversary [Lin17])**.** *Let* π *be a* n*-party protocol that computes the function* $f = (f_i)_{i\in[\![1,n]\!]}$ *for parties* $(P_i)_{i\in[\![1,n]\!]}$ *using inputs* $I = (I_i)_{i\in[\![1,n]\!]} \in \mathcal{I}$ *and security parameter* $\lambda \in \mathbb{N}$*. We say that* π securely computes f in the presence of static semi-honest adversaries *if there exists, for each party* P_i *with* $i \in [\![1,n]\!]$*, a probabilistic polynomial-time simulator* $\mathcal{S}_i$ *such that*

$$\mathcal{S}_{P_i}(1^\lambda, I_i, f_i(I)) \overset{\mathrm{c}}{\equiv} \mathsf{view}^{\pi}_{P_i}(I,\lambda)\ .$$

We say that π *is* secure against collusions *between parties* P_i *and* P_j *with* $i,j \in [\![1,n]\!]$*, if there exists probabilistic polynomial-time simulators* $\mathcal{S}_{P_i,P_j}$ *such that*

$$\mathcal{S}_{P_i,P_j}((1^\lambda, I_i, f_i(I)), (1^\lambda, I_j, f_j(I))) \overset{\mathrm{c}}{\equiv} \mathsf{view}^{\pi}_{P_i,P_j}(I,\lambda)\ .$$

The Random Oracle Model

The *Random Oracle Model* (ROM), formally introduced by Bellare and Rogaway [BR93], is a computational model where all parties of a protocol have access to a public random oracle. Such an oracle outputs a random value chosen from uniform distribution according to the considered domain for every new input it is given. Proofs that use this model are said to be done in the random oracle model while those that do not use it are said to be done in the *standard model*.

In practice, random oracles cannot exist since they would require an infinite description. The random oracle model is widely accepted and widely used since there is no convincing evidence that protocols based on ROM have practical security weaknesses. However, from a theoretical point of view, a proof of security in the standard model is stronger (and more difficult to obtain) than a proof in the random oracle model that leads to much debate among cryptographers on the quality of ROM as an abstraction to analyze the security of cryptosystems [KM15].

2.3.2 Hardness Assumptions

Cryptographic primitives rely on the hardness of some mathematical problems. We describe the one that is used to prove the security of such primitives presented in this thesis.

Decisional Composite Residuosity Assumption

Before defining the *Decisional Composite Residuosity* (DCR) problem, we define the notion of N-residue modulo N, with $N \in \mathbb{N}^*$.

Definition 10 (N-residue modulo N)**.** *Let* q *be an integer in* $[\![0, N-1]\!]$*, we say that* q *is a* N-residue modulo N *if there exists a* $x \in \mathbb{N}$ *such that*

$$x^N \equiv q \mod N\ .$$

Solving the decisional composite residuosity problem for a given composite $N \in \mathbb{N}$ and $q \in \mathbb{N}$, consists in deciding if q is a N-residue modulo N^2.

We say that the decisional composite residuosity problem is hard for (q, N) if for any $\mathcal{A} \in \text{PPT}(\lambda)$, the probability that $\mathcal{A}$ solves the decisional composite residuosity problem is negligible in λ. In practice, the decisional composite residuosity problem is believed to be hard when N is a product of two large prime numbers. We formalize this notion in the following definition.

Definition 11 (Decisional Composite Residuosity assumption [Pai99])**.** *Let N be the product of two prime numbers p and q generated according to a security parameter $\eta \in \mathbb{N}$. The* decisional composite residuosity assumption *states that there exists a negligible function $\mu(\cdot)$ such that for any $\mathcal{A} \in \text{PPT}(\lambda)$, we have*

$$\left| \Pr[y \xleftarrow{\$} \mathbb{N} \colon \mathcal{A}(1^\lambda, y^N \mod N^2) = 1] - \Pr[z \xleftarrow{\$} [\![0, N-1]\!] \colon \mathcal{A}(1^\lambda, z) = 1] \right| \leq \mu(\lambda) \ .$$

2.4 Cryptographic Primitives

We now present cryptographic primitives that we use throughout this thesis.

2.4.1 Hash Function

A hash function transforms a given bit-string of arbitrary size into an other bit-string of a given length ℓ that we call the *hash value*. Such a function is secure when it is hard to inverse it, and when it is hard to find two bit-strings having the same hash value.

Definition 12 (Hash function [MvOV96])**.** *A* hash function *is a deterministic function $H \colon \{0,1\}^* \to \{0,1\}^\ell$. Such a function is said to be* secure *for the security parameter $\lambda \in \mathbb{N}$ when H verifies the three following properties:*

1. Pre-image resistance. *For any adversary $\mathcal{A} \in \text{PPT}(\lambda)$, there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr[h \xleftarrow{\$} \{0,1\}^\ell; m := \mathcal{A}(1^\lambda, H, h) \colon H(m) = h] \leq \mu(\lambda) \ .$$

2. Second pre-image resistance. *For any adversary $\mathcal{A} \in \text{PPT}(\lambda)$, there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr[m \xleftarrow{\$} \{0,1\}^*; h := H(m); m' := \mathcal{A}(1^\lambda, H, h) \colon H(m') = h \wedge m' \neq m] \leq \mu(\lambda) \ .$$

3. Collusion resistance. *For any adversary $\mathcal{A} \in \text{PPT}(\lambda)$, there exists a negligible function $\mu(\cdot)$ such that*

$$\Pr[(m_0, m_1) := \mathcal{A}(1^\lambda, H) \colon H(m_0) = H(m_1)] \leq \mu(\lambda) \ .$$

In practice, families hash functions as SHA-2 [Dan15] and SHA-3 [Dwo15] are considered secure.

Frequently, the random oracle used in the ROM (cf. Section 2.3.1) is instantiated using a hash function. Unfortunately, many hash functions share undesirable properties (e.g., length extension attacks) that make them unfit for such direct use as a random oracle. Instead, we use the HMAC construction [BCK96] with a known key as a random oracle. For a hash function H and a key K, HMAC is defined as $\mathsf{HMAC}(K, x) = H((K \oplus opad) \| H((K \oplus ipad) \| x))$, where *opad* and *ipad* are two constants defined in the RFC 2104 [KBC97].

2.4.2 Pseudo-Random Function

A pseudo-random function (PRF) F is a deterministic algorithm that takes as input a *key* k from the key space $\mathcal{K}$ and an *input data block* x from the input space $\mathcal{X}$, and outputs the *output data block* y equals to $F(k,x)$ that belongs to the output space $\mathcal{Y}$. Note that $\mathcal{K}$, $\mathcal{X}$, and $\mathcal{Y}$ are finite spaces. We say that F is defined over $(\mathcal{K},\mathcal{X},\mathcal{Y})$.

Intuitively, a PRF is said to be *secure* if for a randomly chosen key $k \in \mathcal{K}$, an adversary $\mathcal{A}$ cannot distinguish the function $F(k,\cdot)$ from a randomly chosen function $f \in \mathsf{Func}[\mathcal{X},\mathcal{Y}]$, i.e., the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$.

In order to formally define the security of a pseudo-random function F defined over $(\mathcal{K},\mathcal{X},\mathcal{Y})$, we introduce in Figure 2.1 the pseudo-random function distinguishing experiment, that we call the PRF experiment, for F against a q-query adversary $\mathcal{A}$, with $q \in \mathbb{N}^*$ and $b \in \{0,1\}$.

Experiment: $\mathsf{Exp}^{\text{prf-}b}_{F,q,\mathcal{A}}(\lambda)$

if $b = 0$ **then**
  $k \xleftarrow{\$} \mathcal{K}$, $f_b := F(k,\cdot)$
if $b = 1$ **then**
  $f_b \xleftarrow{\$} \mathsf{Func}[\mathcal{X},\mathcal{Y}]$
$\{x_1,\ldots,x_q\} := \mathcal{A}(1^\lambda, q, \mathcal{X})$
for $i \in [\![1,q]\!]$ **do**
  $y_i := f_b(x_i)$
$b_* := \mathcal{A}(1^\lambda, y_1, \ldots, y_q)$
return b_*

Figure 2.1: PRF experiment.

We define the pseudo-random function distinguishing advantage of a q-query adversary $\mathcal{A}$ with respect to F as

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

Definition 13 (Secure pseudo-random function)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter, and F be a pseudo-random function F defined over $(\mathcal{K},\mathcal{X},\mathcal{Y})$. We say that F is a* secure *pseudo-random function if for any $q \in \mathbb{N}^*$, there exists a negligible function $\mu(\cdot)$ such that*

$$\max_{\mathcal{A}\in \text{PPT}(\lambda)} \left\{\mathsf{Adv}^{\text{prf}}_{F,q,\mathcal{A}}(\lambda)\right\} \leq \mu(\lambda) .$$

In the rest of this thesis, we will simply denote $\mathsf{Exp}^{\text{prf-}b}_{F,q,\mathcal{A}}$ (resp. $\mathsf{Adv}^{\text{prf}}_{F,q,\mathcal{A}}$) by $\mathsf{Exp}^{\text{prf-}b}_{F,\mathcal{A}}$ (resp. $\mathsf{Adv}^{\text{prf}}_{F,\mathcal{A}}$).

2.4.3 Symmetric Encryption

A symmetric encryption scheme is an encryption scheme in which the same key is used for encryption and decryption.

Symmetric Encryption Scheme

We give the formal definition of a symmetric encryption scheme.

Definition 14 (Symmetric Encryption)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. A* symmetric encryption *scheme (SE) is a triple of polynomial-time algorithms $(\mathsf{G},\mathsf{E},\mathsf{D})$ such that*

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

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

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

In practice, a symmetric encryption scheme combines a *block cipher* encrypting a message of a determined length and a *block cipher mode of operation* allowing to process message of arbitrary length with a block cipher. Hence, we suppose that $\mathcal{K} = \{0,1\}^\lambda$, and $\mathcal{M} = \mathcal{C} = \{0,1\}^*$, unless otherwise specified. Also, the G algorithm consists in picking a key in $\mathcal{K}$ uniformly at random.

Order-Preserving Encryption Scheme

Order-preserving symmetric encryption (OPE) is a deterministic symmetric encryption scheme whose encryption function preserves numerical ordering of the plaintexts. This primitive was introduced in the database community by Agrawal et al. [AKSX04].

We first define an *order-preserving function.*

Definition 15 (Order-Preserving Function [BCLO09])**.** *For* $A, B \subseteq \mathbb{N}$*, a function* $f \colon A \to B$ *is* order-preserving *if for all* $x, y \in A$*,* $f(x) < f(y)$ *if and only if* $x > y$*.*

Definition 16 (Order-Preserving Encryption [BCLO09])**.** *We say that a symmetric encryption scheme defined by three polynomial-time algorithms* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *with key space* $\mathcal{K}$*, plaintext space* $\mathcal{M}$*, and ciphertext space* $\mathcal{C}$ *is an* order-preserving encryption *if* $\mathsf{E}(K, \cdot)$ *is an order-preserving function from* $\mathcal{M}$ *to* $\mathcal{C}$ *for all key* $K \in \mathcal{K}$*.*

The standard security definition for order-preserving encryption is the indistinguishability security against ordered chosen plaintext attacks [BCLO09].

For a given order-preserving encryption scheme $\Pi = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ defined over $(\mathcal{M}, \mathcal{C})$, and for an adversary $\mathcal{A} \in \text{PPT}(\lambda)$, we define in Figure 2.2 the IND-OCPA experiment with $b \in \{0,1\}$.

Experiment: $\mathsf{Exp}^{\text{indocpa-}b}_{\Pi,\mathcal{A}}(\lambda)$

$(X_0, X_1) := \mathcal{A}(1^\lambda)$ where $|X_0| = |X_1| = n$ and $\forall 1 \leq i, j \leq n,\ x_{0,i} < x_{0,j} \Leftrightarrow x_{1,i} < x_{1,j}$
$K := \mathsf{G}(1^\lambda)$
foreach $i \in [\![1, n]\!]$ **do**
  $y_{b,i} := \mathsf{E}(K, x_{b,i})$
$b_* := \mathcal{A}(1^\lambda, y_{b,1}, \ldots, y_{b,n})$
return b_*

Figure 2.2: IND-OCPA experiment.

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

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

Definition 17 (Indistinguishability against Ordered Chosen-Plaintext Attack [MRS17])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. An order-preserving encryption scheme Π achieves the indistinguishability security against ordered chosen-plaintext attack if there exists a negligible function $\mu(\cdot)$ such that*

$$\max_{\mathcal{A} \in \mathrm{PPT}(\lambda)} \left\{ \mathsf{Adv}^{\mathrm{indocpa}}_{\Pi,\mathcal{A}}(\lambda) \right\} \leq \mu(\lambda) \ .$$

It is well known that OPE scheme that achieves the indistinguishability security against ordered chosen-plaintext attack leak an important amount information on the plaintexts [DDC16, GMN+16, GSB+17, NKW15]. However, Maffei et al. [MRS17] show that the stronger security notion of *indistinguishability against frequency-analyzing ordered chosen plaintext attack* proposed by Kerschbaum [Ker15] leads to a contradiction. Hence, we consider in this thesis basic OPE schemes that achieves the indistinguishability security against ordered chosen-plaintext attack (IND-OCPA).

2.4.4 Asymmetric Encryption

An *asymmetric encryption* (or a public-key encryption) scheme allows a user to encrypt messages to another user using her *public key*. This public key is available for everyone, it can be for instance stored on a Web server. To decrypt such message, the corresponding *secret key* (only know by the owner of the public key) is required.

We give the formal definition of an asymmetric encryption scheme.

Definition 18 (Asymmetric Encryption)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. An* asymmetric encryption *scheme is a triple of polynomial-time algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ such that*

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

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

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

Indistinguishability Under Chosen Plaintext Attack

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

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

The basic definition of indistinguishability under chosen plaintext attack allows one adversary to submit a couple of plaintexts only one time. However, Bellare et al. [BBM00]

show that the indistinguishability security under chosen plaintext attack is equivalent to the indistinguishability security under *multiple chosen plaintexts* attack in a multi-user setting. That means the adversary can receive several public keys and choose several couples of plaintexts (m_0, m_1). For each of these couples, the adversary receives the encryption of m_b for the different public keys, where the same $b \in \{0,1\}$ is used each time.

More precisely, we use the Left-Or-Rigth definition given by Bellare et al. [BBM00]. Let $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme, $\mathcal{A} \in \text{PPT}(\lambda)$, and $(\alpha, \beta) \in \mathbb{N}^2$. For all $i \in [\![1, \beta]\!]$, the oracle $\mathcal{E}(pk_i, (\mathsf{LoR}_b(\cdot,\cdot), \alpha))$ takes as input a couple of plaintexts (m_0, m_1), and returns $\mathcal{E}(pk_i, m_b)$. Moreover, this oracle cannot be called more than α times.

We define the (α, β)-indistinguishability under chosen plaintext attack (IND-CPA) experiment, denoted $\mathsf{Exp}^{\text{indcpa-}b_{\alpha,\beta}}_{\Pi,\mathcal{A}}$ for the adversary $\mathcal{A}$ against Π in Figure 2.3.

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

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

Figure 2.3: IND-CPA experiment.

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

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

Definition 19 (Indistinguishability under multiple chosen plaintexts attack [BBM00])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. An* asymmetric encryption scheme Π *achieves the (α, β)-indistinguishability security under multiple chosen plaintexts attack, if there exists a negligible function $\mu(\cdot)$ such that*

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

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

Homomorphic Encryption

An asymmetric homomorphic encryption scheme is an asymmetric encryption scheme allowing to manipulate some ciphertexts $c_1, \ldots, c_n$ without the secret key in order to obtain the encryption of $f(m_1, \ldots, m_n)$ where $c_1, \ldots, c_n$ are the respective ciphertexts of plaintexts $m_1, \ldots, m_n$.

We say that the asymmetric encryption scheme is a *fully homomorphic encryption* scheme when any function $f(\cdot)$ can be computed from the ciphertexts. In his seminal paper, Gentry [Gen09] proves that such a fully homomorphic encryption schemes exists. Since, several fully homomorphic encryption schemes have been developed as the Brakerski-Gentry-Vaikuntanathan (BGV) cryptosystem [BGV12], the Gentry-Sahai-Waters (GSW) cryptosystem [GSW13], and the Cheon-Kim-Kim-Song (CKKS) cryptosystem [CKKS17]. Two major libraries `SEAL` [MR19] and `HElib` [HS19] implement CKKS and BGV cryptosystems respectively.

Due to the "fully" characteristic, such asymmetric encryption schemes are not yet efficient enough to be used in everyday life for arbitrary operations. Two other families of homomorphic encryption schemes exist:

- *Somewhat homomorphic encryption.* It supports homomorphic operations with additions and multiplications but allows only a limited number of operations.
- *Partially homomorphic encryption.* It supports only one type of operation: addition or multiplication.

In the following, we present the Paillier cryptosystem that is an asymmetric partially homomorphic encryption scheme with regard to the addition operation.

Paillier's Cryptosystem

Paillier's cryptosystem is an asymmetric encryption scheme. It is well known due to its homomorphic properties described in the following.

Definition 20 (Paillier's cryptosystem [Pai99])**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. The Paillier cryptosystem is an asymmetric encryption scheme defined by a triple of polynomial-time algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ such that:*

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

Theorem 1 ([Pai99])**.** *Paillier's cryptosystem achieves the indistinguishability security against chosen plaintext attack under the DCR assumption.*

The proof is given by Paillier [Pai99].
Now, we present the homomorphic properties of Paillier's cryptosystem.

Homomorphic Addition of Plaintexts. Let m_1 and m_2 be two plaintexts in $\mathbb{Z}_n$. The product of the two associated ciphertexts with the public key $pk = (n, g)$, denoted $c_1 := \mathcal{E}(pk, m_1) = g^{m_1} \cdot r_1^n \mod n^2$ and $c_2 := \mathcal{E}(pk, m_2) = g^{m_2} \cdot r_2^n \mod n^2$, is the encryption of the sum of m_1 and m_2. Indeed, we have:

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

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

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

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

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

Figure 2.4: Paillier interactive multiplicative homomorphic protocol [CDN01].

Interactive Homomorphic Multiplication of Ciphertexts. Cramer et al. [CDN01] show that a two-party protocol makes possible to perform multiplication over ciphertexts using additive homomorphic encryption schemes as Paillier's cryptosystem. More precisely, P_1 knows two ciphertexts $c_1, c_2 \in \mathbb{Z}_{n^2}^*$ of the plaintexts $m_1, m_2 \in \mathbb{Z}_n$ encrypted using the public key pk of P_2, she wants to obtain the ciphertext corresponding to $m_1 \cdot m_2$ without revealing to P_2 the plaintexts m_1 and m_2. In order to do that, P_1 has to interact with P_2 as described in Figure 2.4. First, P_1 randomly picks $\delta_1, \delta_2 \in \mathbb{Z}_n$ and sends to Alice $\alpha_1 := c_1 \cdot \mathcal{E}(pk, \delta_1)$ and $\alpha_2 := c_2 \cdot \mathcal{E}(pk, \delta_2)$. By decrypting respectively α_1 and α_2, P_2 recovers respectively $m_1 + \delta_1 \mod n$ and $m_2 + \delta_2 \mod n$. She sends to P_1 $c := \mathcal{E}(pk, (m_1 + \delta_1) \cdot (m_2 + \delta_2) \mod n)$. Then, P_1 can deduce the value of $\mathcal{E}(pk, m_1 \cdot m_2 \mod n)$ by computing $c \cdot (\mathcal{E}(pk, \delta_1 \cdot \delta_2 \mod n) \cdot c_1^{\delta_1} \cdot c_2^{\delta_2})^{-1}$.

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

2.5 Conclusion

We have defined the mathematical background and cryptographic primitives along their security properties that are required for the sequel of this thesis.

Part I

Matrix Multiplication

CHAPTER 3

Secure Matrix Multiplication with MapReduce

We propose four secure multiparty protocols to compute standard matrix multiplication using the MapReduce paradigm. We address the inherent security and privacy concerns that occur when outsourcing to a public cloud. Our goal is to enhance the two state-of-the-art protocols [LRU14] for MapReduce matrix multiplication with privacy guarantees such as: none of the nodes storing an input matrix can learn the other input matrix or the output matrix, and moreover, none of the nodes computing an intermediate result can learn the input or the output matrices. We consider two different approaches. The first one, called *Secure-Private*, assumes that cluster's nodes do not collude. On the contrary, the second one, called *Collision-Resistant-Secure-Private* assumes that cluster's nodes may collude. For each approach, we design secure versions of the two state-of-the-art protocols. This work has been conducted in collaboration with Xavier Bultel, Radu Ciucanu, and Pascal Lafourcade, and has been published in the paper "*Secure Matrix Multiplication with MapReduce*" [BCGL17] at ARES 2017 conference.

Contents

3.1 Introduction 29
3.1.1 Problem Statement 29
3.1.2 Contributions 30
3.1.3 Outline 30
3.2 Matrix Multiplication with MapReduce 30
3.2.1 Matrix Multiplication with Two MapReduce Rounds 31
3.2.2 Matrix Multiplication with One MapReduce Round 32
3.3 Secure Matrix Multiplication with MapReduce 32
3.3.1 Preprocessing for Secure Matrix Multiplication 33
3.3.2 SP Matrix Multiplication with MapReduce 33
3.3.3 CRSP Matrix Multiplication with MapReduce 35
3.3.4 Complexity Comparison 38
3.4 Experimental Results 38
3.4.1 Dataset and Settings 38
3.4.2 Results 38
3.5 Security Proofs 40
3.5.1 Security Proof for CRSP-2R and CRSP-1R Protocols 40
3.5.2 Security Proof for the SP-2R Protocol 43

3.5.3 Security Proof for the SP-1R Protocol 51
3.6 Conclusion 54

3.1 Introduction

We address the fundamental problem of *MapReduce matrix multiplication* from a privacy-preserving perspective, i.e., we develop protocols for which the public cloud cannot learn neither the input nor the output data. The matrix multiplication is also the original purpose for which the Google implementation of MapReduce was created. Indeed, such multiplications are needed by Google in the computation of the PageRank algorithm [DG04]. The standard protocols for MapReduce matrix multiplication, presented by Leskovec et al. [LRU14], use either two or one MapReduce rounds, where each round is composed of Map and Reduce functions. Their communication and computation cost analysis have been thoroughly analyzed [LRU14].

3.1.1 Problem Statement

Two distinct data owners respectively hold compatible matrices M and N such that $M \in \mathbb{R}^{a\times b}$ and $N \in \mathbb{R}^{b\times c}$ where $(a, b, c) \in (\mathbb{N}^*)^3$. A user, that we call the MapReduce's user, does not know the matrices M and N, and wants their product $P := M \times N$. First, M and N are sent to the distributed file system of some public cloud provider. We assume that the matrix M is initially spread over a set $\mathcal{M}$ of nodes of the public cloud, each of them storing a chunk of M, i.e., a set of elements of M. Similarly, the matrix N is initially spread over a set $\mathcal{N}$ of nodes of the public cloud. In the case of the protocol using one MapReduce round, the final result P is computed over a set $\mathcal{R}$ of nodes before it is sent to the user's nodes $\mathcal{P}$; in the case of the protocol using two MapReduce rounds, intermediate results are spread over a set $\mathcal{R}_1$ of nodes and the final result P is computed over a set $\mathcal{R}_2$ of nodes before it is sent to the user's nodes $\mathcal{P}$. We assume that data owners and the MapReduce's user are trustworthy while the cloud service provider is semi-honest, i.e., it dutifully executes the protocol but tries to deduce as much as possible information on matrices' elements. Moreover, we assume the public cloud and the MapReduce's user do not collude. We illustrate the architecture of the MapReduce matrix multiplication for one round in Figure 3.1 and for two rounds in Figure 3.2.

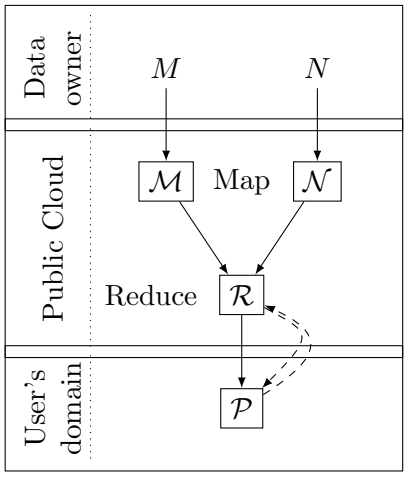


Figure 3.1: One Round.

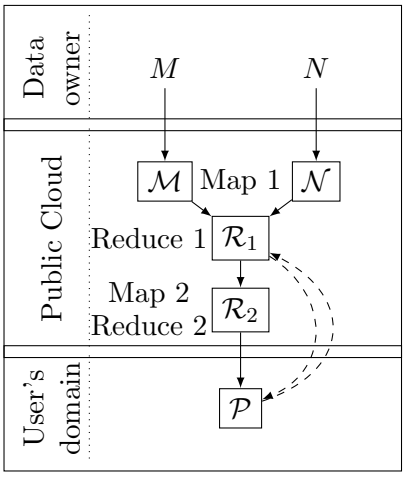


Figure 3.2: Two Rounds.

We expect the following properties.

1. The MapReduce's user cannot learn any information about input matrices M and N.

2. Set of nodes $\mathcal{M}$ and $\mathcal{N}$ cannot learn any information about matrices M, N and P.

3. Set of nodes $\mathcal{R}$ (for one MapReduce round), or $\mathcal{R}_1$ and $\mathcal{R}_2$ (for two MapReduce rounds) cannot learn any information about matrices M, N, and P.

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

3.1.2 Contributions

We propose protocols that extend the two standard protocols for MapReduce matrix multiplication (as found in Chapter 2 from [LRU14]) while ensuring data privacy, and remaining efficient from both computational and communication points of view. Our technique is based on the well-known Paillier's cryptosystem [Pai99], which is additive-homomorphic. The integration of this cryptosystem into the standard MapReduce protocols for matrix multiplication is not trivial. Indeed, Paillier's cryptosystem does not allow to execute directly in the encrypted domain the multiplications that are needed for matrix multiplication.

We propose the following contributions.

- Assuming that the public cloud's nodes do not collude, we design the *Secure-Private* (SP) approach. In this approach, our secure protocol using two MapReduce rounds is called SP-2R while the one using one MapReduce round is called SP-1R. Protocol SP-2R satisfies all aforementioned conditions. However, we show that if nodes $\mathcal{N}$ collude with nodes $\mathcal{R}_2$, then $\mathcal{R}_2$ can retrieve all elements of matrix N. On the contrary, protocol SP-1R only satisfies the first condition. Indeed, nodes of $\mathcal{N}$ and of $\mathcal{R}$ learn the content of matrix N since only the matrix M is encrypted.

- The second approach designs a sophisticated protocol that relies on additional communications to overcome these risks of collusions; this idea leds to our *Collision-Resistant-Secure-Private* (CRSP) approach, which satisfies all conditions enumerated in the problem statement even if the public cloud's nodes collude. In this approach, our secure protocol using two communications rounds is called CRSP-2R and the one using one communication round is called CRSP-1R.

- For each protocol, we give experimental results using Apache Hadoop® [Fou19b] open-source MapReduce implementation, and provide a security proof in the standard model.

3.1.3 Outline

We start by recalling the two state-of-the-art protocols for MapReduce matrix multiplication in Section 3.2. Then, we present and analyse our secure protocols for SP and CRSP approaches in Section 3.3. In Section 3.4, we show an experimental evaluation of the standard protocols and of our secure protocols using Hadoop [Fou19b], the Apache MapReduce implementation. Before to conclude, we prove in Section 3.5 the security of our protocols in the standard model.

3.2 Matrix Multiplication with MapReduce

We recall the two state-of-the-art protocols proposed by Leskovec et al. [LRU14]. Let M and N be two compatible matrices such that $M \in \mathbb{R}^{a \times b}$ and $N \in \mathbb{R}^{b \times c}$ where $(a, b, c) \in (\mathbb{N}^\star)^3$. We denote by $m_{i,j}$ the element of the matrix M which is in the i-th row and the

j-th column with $i \in [\![1, a]\!]$ and $j \in [\![1, b]\!]$. In the same way, we denote by $n_{j,k}$ the element of the matrix N which is in the j-th row and k-th column with $j \in [\![1, b]\!]$ and $k \in [\![1, c]\!]$. Moreover, we denote by P the product MN, and by $p_{i,k}$ the element of the matrix P which is in the i-th row and the k-th column with $i \in [\![1, a]\!]$ and $k \in [\![1, c]\!]$.

3.2.1 Matrix Multiplication with Two MapReduce Rounds

The standard matrix multiplication with two MapReduce rounds protocol [LRU14] is denoted MM-2R and composed of four functions: first Map function, first Reduce function, second Map function and second Reduce function. These four functions are given in Figure 3.3.

1st Map function:

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

1st Reduce function:

Input: $(key, values)$
// *key*: $j \in [\![1, b]\!]$
// *values*: collection of $(\texttt{M}, i, m_{i,j})$ or $(\texttt{N}, k, n_{j,k})$
foreach $(\texttt{M}, i, m_{i,j}) \in values$ **do**
  foreach $(\texttt{N}, k, n_{j,k}) \in values$ **do**
    $\texttt{emit}_{\mathcal{R}_1\to\mathcal{R}_2}((i, k), (m_{i,j} \cdot n_{j,k}))$

2nd Reduce function:

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

Figure 3.3: Map and Reduce functions for MM-2R protocol.

- *The First Map Function.* It consists of rewriting each element of matrices M and N in the form of key-value pairs such that elements needed to compute each element of the product MN share the same key. Hence, when nodes $\mathcal{M}$ receive chunks of matrix M from the data owner, they create pairs of the form $(j, (\texttt{M}, i, m_{i,j}))$, where $i \in [\![1, a]\!]$ and $j \in [\![1, b]\!]$, and send them to the set of nodes $\mathcal{R}_1$. In the same way, when nodes $\mathcal{N}$ receive chunks matrix N from the data owner, they create pairs of the form $(j, (\texttt{N}, k, n_{j,k}))$ where $j \in [\![1, b]\!]$ and $k \in [\![1, c]\!]$, and send them to the set of nodes $\mathcal{R}_1$. We stress that `M` and `N` in the values are the names of matrices, that can be encoded with a single bit, and not the matrices themselves.

- *The First Reduce Function.* To compute elements $p_{i,k}$ of P with $i \in [\![1, a]\!]$ and $k \in [\![1, c]\!]$, the first step is to compute each product $m_{i,j} \cdot n_{j,k}$ for $j \in [\![1, b]\!]$. Hence, from key-values pairs sent by the first Map function executed by nodes $\mathcal{M}$ and $\mathcal{N}$, the Reduce function executed on nodes $\mathcal{R}_1$ creates key-values pairs and sends them

to nodes $\mathcal{R}_2$ where for a key (i,k) with $i \in [\![1,a]\!]$ and $k \in [\![1,c]\!]$, values are $m_{i,j} \cdot n_{j,k}$ for $j \in [\![1,b]\!]$.

- *The Second Map Function.* In this second round, the Map function is only the identity function. We further omit it from the analysis of the computation cost.

- *The Second Reduce Function.* It is executed by nodes $\mathcal{R}_2$ and aggregates all values with the same key send by nodes $\mathcal{R}_1$ to obtain key-value pairs $((i,k), \sum_{j=1}^{b} m_{i,j} \cdot n_{j,k})$. These key-value pairs corresponds to the elements $p_{i,k}$ of matrix P with $i \in [\![1,a]\!]$ and $k \in [\![1,c]\!]$, and are sent to the MapReduce user's nodes $\mathcal{P}$.

3.2.2 Matrix Multiplication with One MapReduce Round

The standard matrix multiplication with one MapReduce round protocol [LRU14] is denoted MM-1R and is composed of two functions: the Map function and the Reduce function. The two functions are given in Figure 3.4.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of M or N
// *value*: collection of $(i,j,m_{i,j})$ or $(j,k,n_{j,k})$
foreach $(i,j,m_{i,j}) \in value$ **do**
  foreach $k \in [\![1,c]\!]$ **do**
    $\texttt{emit}_{\mathcal{M}\to\mathcal{R}}((i,k),(\texttt{M},j,m_{i,j}))$
foreach $(j,k,n_{j,k}) \in value$ **do**
  foreach $i \in [\![1,a]\!]$ **do**
    $\texttt{emit}_{\mathcal{N}\to\mathcal{R}}((i,k),(\texttt{N},j,n_{j,k}))$

Reduce function:

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

Figure 3.4: Map and Reduce functions for the MM-1R protocol.

- *The Map Function.* It is executed by nodes $\mathcal{M}$ and $\mathcal{N}$, and creates the sets of matrix elements that are needed to compute each element of matrix P, i.e., the product of MN. Since an element of M or N is used for many elements of the final result, the output of the Map function sent to nodes $\mathcal{R}$ gives key-value pairs where keys are (i,k) where $i \in [\![1,a]\!]$ is the row of M and $k \in [\![1,c]\!]$ is the column of N, and values are of the form $(\texttt{M},j,m_{i,j})$ and $(\texttt{N},j,n_{j,k})$.

- *The Reduce Function.* With the previous Map function, we obtain for a key (i,k), with $i \in [\![1,a]\!]$ and $k \in [\![1,c]\!]$, a set of values of the form $(\texttt{M},j,m_{i,j})$ and $(\texttt{N},j,n_{j,k})$ with $j \in [\![1,b]\!]$. Then, the Reduce function executed by nodes $\mathcal{R}$ computes the sum of $m_{i,j} \cdot n_{j,k}$ with $j \in [\![1,b]\!]$ for each key (i,k). These results are paired to the keys (i,k) corresponding to elements $p_{i,k}$ for $i \in [\![1,a]\!]$ and $k \in [\![1,c]\!]$, and sent to the MapReduce user's nodes $\mathcal{P}$.

3.3 Secure Matrix Multiplication with MapReduce

The standard MapReduce matrix multiplication with one and two rounds, presented in the previous Section, reveals all intermediate results to each set of nodes of the public cloud. For instance, when matrix multiplication is performed with two MapReduce rounds, nodes $\mathcal{R}_1$ know all elements of matrices M and N. We describe below MapReduce protocols with SP and CRSP approaches. We assume that MapReduce's user has a Paillier public key denoted pk which is available to the data owners and the public cloud. Since we use Paillier's cryptosystem, the matrix multiplication is computed modulo n, where n is the modulo of pk.

3.3.1 Preprocessing for Secure Matrix Multiplication

Instead of outsourcing the two matrices in clear to the public cloud for privacy reasons, each data owner performs a preprocessing on its own matrix according to the considered approach and the number of MapReduce rounds. The aim of these preprocessings is to allow the public cloud to compute the matrix multiplication while required privacy constraints are satisfied (cf. Section 3.1.1). To run a preprocessing algorithm, data owners need the Paillier public key pk of the MapReduce's user where $pk := (n, g)$, where n is the product of two prime numbers generated according to a security parameter λ, and $g \in \mathbb{Z}^*_{n^2}$.

Two preprocessing algorithms, namely *Preprocessing I* and *Preprocessing II*, are presented in Figure 3.5. Preprocessing I algorithm has for input the Paillier public key $pk := (n, g)$ of the user and a matrix $A \in \mathbb{Z}_n^{u \times v}$ with $(u, v) \in (\mathbb{N}^*)^2$. It computes the encrypted matrix $A^* \in \mathbb{Z}_{n^2}^{u \times v}$ using the Paillier public key pk of the user such that each element $a^*_{i,j} \in A^*$ is the Paillier encryption of element $a_{i,j} \in A$, with $i \in [\![1, u]\!]$ and $j \in [\![1, v]\!]$. Preprocessing I algorithm is used in each protocol of SP and CRSP approaches. However, if both matrices are encrypted using it, it implies the public cloud must run the Paillier interactive multiplicative homomorphic protocol (cf. Figure 2.4 on page 24) to compute elements of the final result. The resulted communication overhead is tolered for the CRSP approach since we assume that public cloud's nodes can collude but is too much costly for the SP approach assuming that public cloud's nodes do not collude.

Therefore, Preprocessing II algorithm uses the idea of adding a random mask to ensure privacy of elements of the matrix. Preprocessing II algorithm has for input the Paillier public key $pk_{\mathcal{R}_2} := (n', g')$ of public cloud's nodes $\mathcal{R}_2$ such that $n' < n$, and a matrix $A \in \mathbb{Z}_n^{u \times v}$ with $(u, v) \in (\mathbb{N}^*)^2$. It computes the encrypted matrix $A^* \in (\mathbb{Z}_n \times \mathbb{Z}_{n'^2})^{u \times v}$ using a random mask and the Paillier public key $pk_{\mathcal{R}_2}$. Each element $a^*_{i,j} \in A^*$, with $i \in [\![1, u]\!]$ and $j \in [\![1, v]\!]$, is a tuple whereas the first term is the addition of element $a_{i,j} \in A$ with a random mask modulo n, and where the second term is the Paillier encryption of the random mask using $pk_{\mathcal{R}_2}$. Preprocessing II algorithm is used only for the SP approach with two MapReduce rounds.

Finally, considering matrix of dimension $u \cdot v$, Preprocessing I algorithm has a complexity of $\mathcal{O}(u \cdot v \cdot \mathcal{C}_{\mathcal{E}})$ and Preprocessing II algorithm has a complexity of $\mathcal{O}(u \cdot v \cdot (\mathcal{C}_+ + \mathcal{C}_{\mathcal{E}} + \mathcal{C}_\$))$, where $\mathcal{C}_+$ is the cost for an addition, $\mathcal{C}_\$$ is the cost to pick a random element, and $\mathcal{C}_{\mathcal{E}}$ is the cost for a Paillier encryption.

3.3.2 SP Matrix Multiplication with MapReduce

The SP matrix multiplication with MapReduce uses the Paillier's cryptosystem and the idea of adding a random mask to ensure privacy of elements of matrices. However, this is only possible when the computation uses two MapReduce rounds and when nodes of the public cloud do not collude.

Preprocessing I:

Input: $pk, A \in \mathbb{Z}_n^{u \times v}$
foreach $i \in [\![1, u]\!]$ **do**
  foreach $j \in [\![1, v]\!]$ **do**
    $a^*_{i,j} := \mathcal{E}(pk, a_{i,j})$
$A^* := (a^*_{i,j})_{i \in [\![1,u]\!], j \in [\![1,v]\!]}$
return A^*

Preprocessing II:

Input: $pk_{\mathcal{R}_2}, A \in \mathbb{Z}_n^{u \times v}$
foreach $i \in [\![1, u]\!]$ **do**
  foreach $j \in [\![1, v]\!]$ **do**
    $\tau_{i,j} \xleftarrow{\$} \mathbb{Z}_{n'}$
    $a^*_{i,j} := (a_{i,j} + \tau_{i,j}, \mathcal{E}(pk_{\mathcal{R}_2}, \tau_{i,j}))$
$A^* := (a^*_{i,j})_{i \in [\![1,u]\!], j \in [\![1,v]\!]}$
return A^*

Figure 3.5: Preprocessing algorithms for secure matrix multiplication.

SP Matrix Multiplication with Two MapReduce Rounds

The SP-2R protocol is composed of four functions: the first Map function, the first Reduce function, the second Map function and the second Reduce function. The four functions are given in Figure 3.6. For this protocol, the data owner of matrix M perfoms the *Preprocessing I* algorithm while the data owner of matrix N performs the *Preprocessing II* algorithm given in Figure 3.5. Assignments of the Preprocessing I algorithm to the owner of matrix M and of the Preprocessing II algorithm to the owner of matrix N is arbitrary and may switch.

- *The First SP Map Function.* It is executed by nodes $\mathcal{M}$ and $\mathcal{N}$. It consists in rewriting each element of matrices M^* and N^* sent by data owners in the form of key-value pairs such that elements needed to compute each encryption of element of P share the same key. Hence, nodes $\mathcal{M}$ creates pairs of the form $(j, (\mathtt{M}, i, m^*_{i,j}))$, where $i \in [\![1, a]\!]$ and $j \in [\![1, b]\!]$, and are sent to the set of nodes $\mathcal{R}_1$. In the same way, nodes $\mathcal{N}$ creates pairs of the form $(j, (\mathtt{N}, k, n^*_{j,k}))$ where $j \in [\![1, b]\!]$ and $k \in [\![1, c]\!]$, and are sent to the set of nodes $\mathcal{R}_1$.

- *The First SP Reduce Function.* It is executed on the set of nodes $\mathcal{R}_1$. As we have seen in the standard MapReduce, the first Reduce function produces key-value pairs where the key is equal to (i, k), with $i \in [\![1, a]\!]$ and $k \in [\![1, c]\!]$, and values is equal to $m_{i,j} \cdot n_{j,k}$ for $j \in [\![1, b]\!]$. In the SP approach, keys are also equal to (i, k) but values are tuples equal to $(\mathcal{E}(pk, m_{i,j})^{n_{j,k}+\tau_{j,k}}, \mathcal{E}(pk, m_{i,j}), \mathcal{E}(pk_{\mathcal{R}_2}, \tau_{j,k}))$. Thus, nodes $\mathcal{R}_1$ can compute $\mathcal{E}(pk, m_{i,j} \cdot n_{j,k}) \cdot \mathcal{E}(pk, m_{i,j})^{\tau_{j,k}}$ using homomorphic properties of Paillier's cryptosystem which is equal to $\mathcal{E}(pk, m_{i,j})^{n_{j,k}+\tau_{j,k}}$ in the tuple. The mask removal is later on done in nodes $\mathcal{R}_2$.

- *The Second SP Map Function.* As for MM-2R protocol, the second Map function of the SP approach is the identity function.

- *The Second SP Reduce Function.* It is executed on $\mathcal{R}_2$, and multiplies all values associated to the same key (i, k). Moreover, nodes $\mathcal{R}_2$ remove all masks for each value using $\mathcal{E}(pk, m_{i,j})$ and $\mathcal{E}(pk_{\mathcal{R}_2}, \tau_{j,k})$ emitted by $\mathcal{R}_1$. In fact, to compute $\mathcal{E}(pk, m_{i,j} \cdot$

1st Map function:

Input: $(key, value)$
// *key*: id of a chunk of M^* or N^*
// *value*: collection of $(i, j, \mathcal{E}(pk, m_{i,j}))$ or $(j, k, (n_{j,k} + \tau_{j,k}, \mathcal{E}(pk_{\mathcal{R}_2}, t_{j,k})))$
foreach $(i, j, \mathcal{E}(pk, m_{i,j})) \in value$ **do**
  $\texttt{emit}_{\mathcal{M}\to\mathcal{R}_1}(j, (\texttt{M}, i, \mathcal{E}(pk, m_{i,j})))$
foreach $(j, k, (n_{j,k} + \tau_{j,k}, \mathcal{E}(pk_{\mathcal{R}_2}, t_{j,k}))) \in value$ **do**
  $\texttt{emit}_{\mathcal{N}\to\mathcal{R}_1}(j, (\texttt{N}, k, (n_{j,k} + \tau_{j,k}, \mathcal{E}(pk_{\mathcal{R}_2}, t_{j,k}))))$

1st Reduce function:

Input: $(key, values)$
// *key*: $j \in [\![1, b]\!]$
// *values*: collection of $(\texttt{M}, i, \mathcal{E}(pk, m_{i,j}))$ or $(\texttt{N}, k, (n_{j,k} + \tau_{j,k}, \mathcal{E}(pk_{\mathcal{R}_2}, \tau_{j,k})))$
foreach $(\texttt{M}, i, \mathcal{E}(pk, m_{i,j})) \in values$ **do**
  foreach $(\texttt{N}, k, (n_{j,k} + \tau_{j,k}, \mathcal{E}(pk_{\mathcal{R}_2}, \tau_{j,k}))) \in values$ **do**
    $\texttt{emit}_{\mathcal{R}_1\to\mathcal{R}_2}((i, k), (\mathcal{E}(pk, m_{i,j})^{n_{j,k}+\tau_{j,k}}, \mathcal{E}(pk, m_{i,j}), \mathcal{E}(pk_{\mathcal{R}_2}, \tau_{j,k})))$

2nd Reduce function:

Input: $(key, values)$
// *key*: $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$
// *values*: collection of $(v_1, v_2, v_3) := (\mathcal{E}(pk, m_{i,j})^{n_{j,k}+\tau_{j,k}}, \mathcal{E}(pk, m_{i,j}), \mathcal{E}(pk_{\mathcal{R}_2}, \tau_{j,k}))$
$\texttt{emit}_{\mathcal{R}_2\to\mathcal{P}}\Big((i, k), \prod_{(v_1,v_2,v_3)\in values} v_1 \cdot \big(v_2^{\mathcal{D}(sk_{\mathcal{R}_2}, v_3)}\big)^{-1}\Big)$

Figure 3.6: Map and Reduce functions for the SP-2R protocol.

$n_{j,k}$), nodes $\mathcal{R}_2$ compute

$$\begin{aligned}\mathcal{E}(pk, m_{i,j})^{n_{j,k}+\tau_{j,k}} \cdot \big(\mathcal{E}(pk, m_{i,j})^{\mathcal{D}(sk_{\mathcal{R}_2}, \mathcal{E}(pk_{\mathcal{R}_2}, \tau_{j,k}))}\big)^{-1} &= \\ &= \mathcal{E}(pk, m_{i,j})^{n_{j,k}} \cdot \mathcal{E}(pk, m_{i,j})^{\tau_{j,k}} \cdot \big(\mathcal{E}(pk, m_{i,j})^{\tau_{j,k}}\big)^{-1} \\ &= \mathcal{E}(pk, m_{i,j} \cdot n_{j,k}) \ .\end{aligned}$$

SP Matrix Multiplication with One MapReduce Round

The SP-1R protocol is composed of two functions: the Map function, and the first Reduce function. The two functions are given in Figure 3.7. For this protocol, the data owner of matrix M perfoms the *Preprocessing I* algorithm while the data owner of matrix N does not perform any preprocessing. Assignment of the Preprocessing I algorithm to the owner of matrix M is arbitrary and can be inverted with the data owner of the matrix N.

- *The SP Map Function.* It is executed by nodes $\mathcal{M}$ and $\mathcal{N}$. It consists in rewriting each element of matrices M^* and N sent by data owners in the form of key-value pairs such that elements needed to compute each encryption of element of P share the same key. Hence, nodes $\mathcal{M}$ create pairs of the form $((i, k), (\texttt{M}, j, m^*_{i,j}))$ and nodes $\mathcal{N}$ creates pairs of the form $((i, k), (\texttt{N}, j, n_{j,k}))$ for $i \in [\![1, a]\!]$, $j \in [\![1, b]\!]$, and $k \in [\![1, c]\!]$. Nodes $\mathcal{M}$ and $\mathcal{N}$ send these key-value pairs to the set of nodes $\mathcal{R}$.

- *The SP Reduce Function.* It is executed by nodes $\mathcal{R}$, and uses homomorphic property of Paillier's cryptosystem. In fact, instead to summing all products $m_{i,j} \cdot n_{j,k}$; it mutltiplies, for each key (i, k) with $i \in [\![1, a]\!]$ and $k \in [\![1, c]\!]$, all encrypted values $\mathcal{E}(pk, m_{i,j})^{n_{j,k}}$ corresponding to $\mathcal{E}(pk, m_{i,j} \cdot n_{j,k})$.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of M^* or N
// *value*: collection of $(i, j, \mathcal{E}(pk, m_{i,j}))$ or $(j, k, n_{j,k})$
foreach $(i, j, \mathcal{E}(pk, m_{i,j})) \in value$ **do**
  foreach $k \in [\![1, b]\!]$ **do**
    $\texttt{emit}_{\mathcal{M}\to\mathcal{R}}((i, k), (\texttt{M}, j, \mathcal{E}(pk, m_{i,j})))$
foreach $(j, k, n_{j,k}) \in value$ **do**
  foreach $i \in [\![1, a]\!]$ **do**
    $\texttt{emit}_{\mathcal{N}\to\mathcal{R}}((i, k), (\texttt{N}, j, n_{j,k}))$

Reduce function:

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

Figure 3.7: Map and Reduce functions for SP-1R protocol.

3.3.3 CRSP Matrix Multiplication with MapReduce

In order to keep the content of both matrices private, the CRSP matrix multiplication with MapReduce uses the Paillier's cryptosystem along with the Paillier interactive multiplicative homomorphic protocol presented in Figure 2.4 on page 24.

CRSP Matrix Multiplication with Two MapReduce Rounds

The CRSP-2R protocol is composed of four functions: the first Map function, the first Reduce function, the second Map function and the second Reduce function. The four functions are given in Figure 3.8.

- *The First Map Function.* Unlike the first Map function of the SP approach, elements of both matrices M and N are encrypted. The first Map function executed on nodes $\mathcal{M}$ and $\mathcal{N}$ works as for the SP approach but on encrypted elements of both matrices. It produces key-values pairs $(j, (\texttt{M}, i, m^*_{i,j}))$ and $(j, (\texttt{N}, k, n^*_{j,k}))$ where $i \in [\![1, a]\!]$, $j \in [\![1, b]\!]$, and $k \in [\![1, c]\!]$, sent to the set of nodes $\mathcal{R}_1$. Hence, nodes $\mathcal{M}$ and $\mathcal{N}$ learn nothing about elements of matrices M and N.

- *The First Reduce Function.* It has to perform multiplications of encrypted values having the same key on $\mathcal{R}_1$, i.e., for a key (i, k) with $i \in [\![1, a]\!]$ and $k \in [\![1, c]\!]$, $\mathcal{R}_1$ multiplies $\mathcal{E}(pk, m_{i,j})$ by $\mathcal{E}(pk, n_{j,k})$ for $j \in [\![1, b]\!]$. We use the Paillier interactive multiplicative protocol to perform mutiplication of two ciphertexts with the Paillier's cryptosystem. This protocol is denoted Paillier.Inter and presented in Figure 2.4 on page 24. Hence, nodes $\mathcal{R}_1$ interacts with the client's set of nodes $\mathcal{P}$ to perform the multiplications. It sends key-values pairs to the set of nodes $\mathcal{R}_2$.

- *The Second Map Function.* It consists in the identity function.

- *The Second Reduce Function.* Since nodes $\mathcal{R}_2$ receives key-value pairs where values v is equal to $\mathcal{E}(pk, m_{i,j} \cdot n_{j,k})$, nodes $\mathcal{R}_2$ uses the homomorphic property of Paillier's cryptosystem to compute the sum of all encrypted values associated to the same key (i, k) with $i \in [\![1, a]\!]$ and $k \in [\![1, c]\!]$. Precisely, it computes $\prod_{v \in values} v$ for a key (i, k) which represents $\mathcal{E}(pk, \sum_j m_{i,j} \cdot n_{j,k})$.

1st Map function:

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

1st Reduce function:

Input: $(key, values)$
// *key*: $j \in [\![1, b]\!]$
// *values*: collection of $(\texttt{M}, i, \mathcal{E}(pk, m_{i,j}))$ or $(\texttt{N}, k, \mathcal{E}(pk, n_{j,k}))$
foreach $(\texttt{M}, i, \mathcal{E}(pk, m_{i,j})) \in values$ **do**
  foreach $(\texttt{N}, k, \mathcal{E}(pk, n_{j,k})) \in values$ **do**
    $\texttt{emit}_{\mathcal{R}_1\to\mathcal{R}_2}((i, k), \mathsf{Paillier.Inter}(\mathcal{E}(pk, m_{i,j}), \mathcal{E}(pk, n_{j,k})))$

2nd Reduce function:

Input: $(key, values)$
// *key*: $(i, k) \in [\![1, a]\!] \times [\![1, c]\!]$
// *values*: collection of $v := \mathcal{E}(pk, m_{i,j} \cdot n_{j,k})$
$\texttt{emit}_{\mathcal{R}_2\to\mathcal{P}}\big((i, k), \prod_{v\in values} v\big)$

Figure 3.8: Map and Reduce functions for the CRSP-2R protocol.

CRSP Matrix Multiplication with One MapReduce Round

Finally, we present the CRSP-1R protocol composed of two functions: the Map function and the Reduce function. The two functions are given in Figure 3.9.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of M^* or N^*
// *value*: collection of $(i, j, \mathcal{E}(pk, m_{i,j}))$ or $(j, k, \mathcal{E}(pk, n_{j,k}))$
foreach $(i, j, \mathcal{E}(pk, m_{i,j})) \in value$ **do**
  foreach $k \in [\![1, c]\!]$ **do**
    $\texttt{emit}_{\mathcal{M}\to\mathcal{R}}((i, k), (\texttt{M}, j, \mathcal{E}(pk, m_{i,j})))$
foreach $(j, k, \mathcal{E}(pk, n_{j,k})) \in value$ **do**
  foreach $i \in [\![1, a]\!]$ **do**
    $\texttt{emit}_{\mathcal{N}\to\mathcal{R}}((i, k), (\texttt{N}, j, \mathcal{E}(pk, n_{j,k})))$

Reduce function:

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

Figure 3.9: Map and Reduce functions for the CRSP-1R protocol.

- *The Map Function.* It is executed by nodes $\mathcal{M}$ and $\mathcal{N}$. As for the Map function

of the SP-1R protocol, it consists in rewritting each element of M^* and N^* sent by data owners. Nodes $\mathcal{M}$ create pairs of the form $((i,k),\mathtt{M},j,m^*_{i,j})$ and nodes $\mathcal{N}$ create pairs of the form $((i,k),\mathtt{N},j,n^*_{j,k})$ for $i \in [\![1,a]\!]$, $j \in [\![1,b]\!]$, and $k \in [\![1,c]\!]$. Nodes $\mathcal{M}$ and $\mathcal{N}$ send pairs to the set of nodes $\mathcal{R}$. Hence, $\mathcal{M}$, $\mathcal{N}$ and $\mathcal{R}$ cannot learn any information on elements of matrices M and N.

- *The Reduce Function.* It is executed by the set of nodes $\mathcal{R}$. For a key (i,k), with $i \in [\![1,a]\!]$ and $k \in [\![1,c]\!]$, it takes as input values of the form $(\mathtt{M},j,\mathcal{E}(pk,m_{i,j}))$ and $(\mathtt{N},j,\mathcal{E}(pk,n_{j,k}))$, for $j \in [\![1,b]\!]$. To compute $\sum_{j=1}^{b} m_{i,j} \cdot n_{j,k}$ from these encrypted values, the Reduce function uses the Paillier interactive multiplicative homomorphic protocol denoted Paillier.Inter and presented in Figure 2.4 on page 24.

3.3.4 Complexity Comparison

In Table 3.1, we summarize the computation and the communication costs for our two approaches using one or two communication rounds and the two standard MapReduce protocols for two square matrices of order n. In the computation cost, we quantify by 1 the cost to read a key-value pair. In our communication cost analysis, we measure the total size of the data that is emitted from public cloud nodes and quantify by 1 the cost to send a key-value pair. The CRSP approach satisfies all privacy constraints and resists to collusions but requires a communication overhead.

Table 3.1: Computation and communication costs (big-$\mathcal{O}$) analysis of MapReduce matrix multiplication protocols. Let $C_\times$ (resp. C_+, C_{exp}, $C_{\mathcal{E}}$, $C_{\mathcal{D}}$, C_{inv}, $C_\$$) be the cost of multiplication (resp. addition, exponentiation, encryption, decryption, inversion, sampling).

Protocols	*Computation cost*	*Comm. cost*
MM-2R	$(C_\times + C_+) \cdot n^3$	$n^3 + 3 \cdot n^2$
MM-1R		$2 \cdot n^3 + n^2$
SP-2R	$(2C_\times + 2C_{\text{exp}} + C_{\mathcal{D}} + C_{\text{inv}}) \cdot n^3$	$3 \cdot n^3 + 4 \cdot n^2$
SP-1R	$(C_\times + C_{\text{exp}}) \cdot n^3$	$2 \cdot n^3 + n^2$
CRSP-2R	$(4 \cdot C_{\mathcal{E}} + 7 \cdot C_\times + 2 \cdot C_{\mathcal{D}} + 2 \cdot C_{\text{exp}} + 2 \cdot C_\$ + C_{\text{inv}}) \cdot n^3$	$3 \cdot n^3 + 3 \cdot n^2$
CRSP-1R		$5 \cdot n^3 + n^2$

3.4 Experimental Results

We present the experimental results for the two state-of-the-art protocols [LRU14], and for our secure SP and CRSP approaches.

3.4.1 Dataset and Settings

For each experiment, we generate two random matrices composed of elements in $[\![0,10]\!]$. We measure the CPU time of the matrix multiplication for matrices' dimension from 90 to 450 for the two state-of-the-art protocols, and from 90 to 300 for our secure approaches, in steps of 30.

As mentionned in Section 3.3, both secure approaches are based on the Paillier's cryptosystem. We use Gaillier[†], a Go implementation of the Paillier's cryptosystem. Note that Gaillier is not an optimized implementation. Hence, we use it with a 64-bit RSA modulus as proof of concept.

[†] https://github.com/actuallyachraf/gomorph

3.4.2 Results

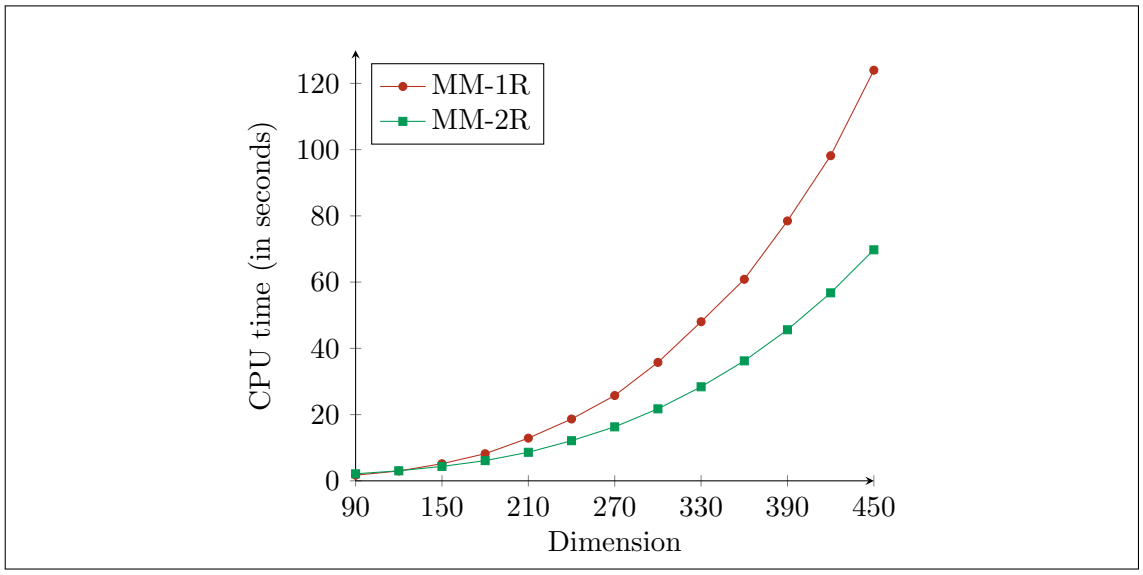


Figure 3.10: CPU time vs the matrices' dimension for the two state-of-the-art protocols [LRU14] computing the matrix multiplication.

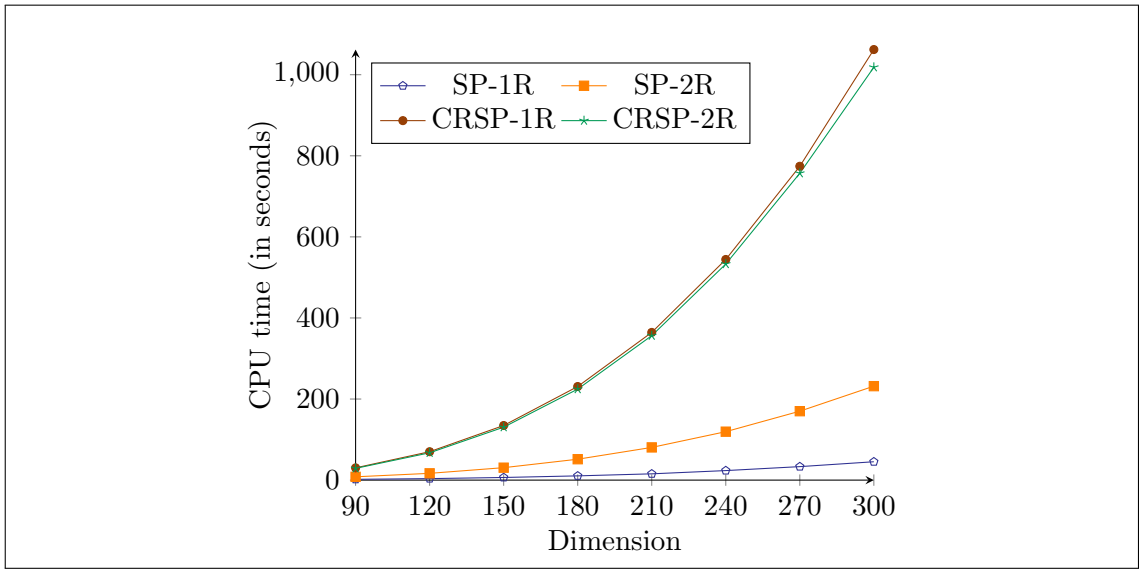


Figure 3.11: CPU time vs the matrices' dimension for our SP and CRSP approaches computing the matrix multiplication.

Figure 3.10 presents experimental results of the two state-of-the-art protocols [LRU14] while Figure 3.11 presents those for our SP and CRSP approaches.

First, we observe in Figure 3.10 that the MM-2R protocol is more efficient than the MM-1R protocol. The trend is visible when matrices' dimension is larger than 180. This is reasonable since MM-1R requires more communication than MM-2R. Moreover, MM-2R uses two MapReduce rounds, hence Apache Hadoop® has to execute twice as many disk operations.

In Figure 3.11, we remark than the protocol SP-1R is much faster than the protocol SP-2R. Indeed, SP-2R has more privacy guarantee than SP-1R and requires the public cloud to perform a Paillier decryption, which is computationally expensive. Finally, the

CPU time of CRSP approach is much important since it uses Paillier interactive multiplicative homomorphic protocol which implies extra Paillier encryptions and decryptions.

3.5 Security Proofs

We provide formal security proof for CRSP-2R, CRSP-1R, SP-2R, and SP-1R protocols. We use the standard multiparty computations definition of security against semi-honest adversaries [Lin17].

3.5.1 Security Proof for CRSP-2R and CRSP-1R Protocols

The CRSP approach assumes that public cloud's nodes may collude, hence in a security point of view, the security proofs of CRSP-2R and CRSP-1R protocols are the same. In fact, the only difference between the two protocols is that the public cloud has sets of nodes $\mathcal{M}$, $\mathcal{N}$, $\mathcal{R}_1$, and $\mathcal{R}_2$ for the two rounds case, and has sets of nodes $\mathcal{M}$, $\mathcal{N}$, and $\mathcal{R}$ for the one round case. Hence, these sets of nodes is considered as a unique set of nodes in both cases when they collude.

We model both protocols (i.e., CRSP-2R and CRSP-1R protocols) with four parties P_M, P_N, $P_{\mathcal{C}}$, and $P_{\mathcal{P}}$ using respective inputs $I := (I_M, I_N, I_{\mathcal{C}}, I_{\mathcal{P}}) \in \mathcal{I}$ and a function $g := (g_M, g_N, g_{\mathcal{C}}, g_{\mathcal{P}})$ such that

- P_M is the data owner of M. It has the input $I_M := (M, pk)$, where M is its private matrix and pk is the Paillier's public key of the user. P_M returns $g_M(I) := \perp$ because it does not learn anything.
- P_N is the data owner of N. It has the input $I_N := (N, pk)$, where N is its private matrix and pk is the Paillier's public key of the user. P_N returns $g_N(I) := \perp$ because it does not learn anything.
- $P_{\mathcal{C}}$ is the public cloud's nodes that represents the collusion between sets of nodes $\mathcal{M}$, $\mathcal{N}$, $\mathcal{R}_1$, and $\mathcal{R}_2$ for the two-rounds protocol, and between sets of nodes $\mathcal{M}$, $\mathcal{N}$, and $\mathcal{R}$ for the one-round protocol. It has the input $I_{\mathcal{C}} := pk$, where pk is the Paillier's public key of the user. $P_{\mathcal{C}}$ returns $g_{\mathcal{C}}(I) := (a, b, c) \in (\mathbb{N}^*)^3$ because it learns M, N, and P's dimensions.
- $P_{\mathcal{P}}$ is the set nodes $\mathcal{P}$ of the user. It has the input $I_{\mathcal{P}} := (pk, sk)$, where (pk, sk) is the Paillier's key pair of the MapReduce user. $P_{\mathcal{P}}$ returns $g_{\mathcal{P}}(I) := P$ because the user obtains the result of the matrix multiplication at the end of both protocols.

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

The security of CRSP protocols is given by Theorem 2.

Theorem 2. *Assume Paillier's cryptosystem is IND-CPA, then CRSP protocols securely compute the matrix mutiplication in the presence of semi-honest adversaries even if public cloud's nodes collude.*

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

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

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

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

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

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

Simulator: $\mathcal{S}_{\mathcal{C}}^{\text{CRSP}}(1^\lambda, pk, \perp)$

foreach $i \in [\![1, a]\!]$ **do**
  foreach $j \in [\![1, b]\!]$ **do**
    $\alpha_{i,j} \xleftarrow{\$} \mathbb{Z}_n$
$M^* := \left(\mathcal{E}(pk, \alpha_{i,j})\right)_{i\in[\![1,a]\!],j\in[\![1,b]\!]}$
foreach $j \in [\![1, b]\!]$ **do**
  foreach $k \in [\![1, c]\!]$ **do**
    $\beta_{j,k} \xleftarrow{\$} \mathbb{Z}_n$
$N^* := \left(\mathcal{E}(pk, \beta_{j,k})\right)_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
foreach $i \in [\![1, a]\!]$ **do**
  foreach $k \in [\![1, c]\!]$ **do**
    foreach $j \in [\![1, b]\!]$ **do**
      $(r_{i,j,k}, s_{i,j,k}, t_{i,j,k}) \xleftarrow{\$} (\mathbb{Z}_n)^3$
      $x_{i,j,k} := \mathcal{E}(pk, \alpha_{i,j}) \cdot \mathcal{E}(pk, r_{i,j,k})$
      $y_{i,j,k} := \mathcal{E}(pk, \beta_{j,k}) \cdot \mathcal{E}(pk, s_{i,j,k})$
      $z_{i,j,k} := \mathcal{E}(pk, t_{i,j,k})$
    $p^*_{ik} := \prod_{j=1}^{b} z_{i,j,k}$
$\mathsf{view} := \left(M^*, N^*, \{(x_{i,j,k}, y_{i,j,k}), z_{i,j,k}\}_{i\in[\![1,a]\!],j\in[\![1,b]\!],k\in[\![1,c]\!]}, \{p^*_{ik}\}_{i\in[\![1,a]\!],k\in[\![1,c]\!]}\right)$
return view

Figure 3.12: Simulator $\mathcal{S}_{\mathcal{C}}^{\text{CRSP}}$ for the proof of Lemma 2.

Proof. We recall that $P_{\mathcal{C}}$ is the collusion of set of nodes $\mathcal{M}$, $\mathcal{N}$, $\mathcal{R}_1$, $\mathcal{R}_2$ for the two rounds protocol, and the collusion of $\mathcal{M}$, $\mathcal{N}$, $\mathcal{R}$ for the one round protocol. In CRSP protocols, $P_{\mathcal{C}}$ receives M^* and N^* from the data owners. Moreover, $\mathcal{R}_1$ (for the two rounds case) or $\mathcal{R}$ (for the one round case) sends couples of ciphertexts $(x_{i,j,k}, y_{i,j,k})$ to $P_{\mathcal{P}}$ and receives all corresponding ciphertexts $z_{i,j,k}$ returned by $P_{\mathcal{P}}$ to compute multiplication on encrypted coefficients, where $i \in [\![1, a]\!]$, $j \in [\![1, b]\!]$, and $k \in [\![1, c]\!]$. It also contains all values p^*_{ik} of the encrypted matrix P^* sent by $\mathcal{R}_2$ (for the two rounds case) or by $\mathcal{R}$ (for the one round case) to $P_{\mathcal{P}}$. Simulator of $P_{\mathcal{R}}$ is given in Figure 3.12.

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

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

where μ is a non-negligible function in λ. We show how to build a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}$ has a non-negligible advantage to win the IND-CPA experiment on the Paillier's cryptosystem. Then we conclude the proof by contraposition.

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

foreach $i \in [\![1,a]\!]$ **do**
  foreach $j \in [\![1,b]\!]$ **do**
    $m_{i,j} \xleftarrow{\$} \mathbb{Z}_n$
$M := (m_{i,j})_{i\in[\![1,a]\!],j\in[\![1,b]\!]}$
foreach $j \in [\![1,b]\!]$ **do**
  foreach $k \in [\![1,c]\!]$ **do**
    $n_{j,k} \xleftarrow{\$} \mathbb{Z}_n$
$N := (n_{j,k})_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
foreach $i \in [\![1,a]\!]$ **do**
  foreach $k \in [\![1,c]\!]$ **do**
    foreach $j \in [\![1,b]\!]$ **do**
      $(v_{i,j,k}, w_{i,j,k}, \theta_{i,j,k}) \xleftarrow{\$} (\mathbb{Z}_n)^3$
      $x_{i,j,k} := \mathcal{E}(pk, v_{i,j,k})$
      $y_{i,j,k} := \mathcal{E}(pk, w_{i,j,k})$
      $z_{i,j,k} := \mathcal{E}\big(pk, \mathsf{LoR}_b(m_{i,j} \cdot n_{j,k}, \theta_{i,j,k})\big)$
    $p^*_{ik} := \prod_{j=1}^{b} z_{i,j,k}$
foreach $i \in [\![1,a]\!]$ **do**
  foreach $j \in [\![1,b]\!]$ **do**
    $\mu_{i,j} \xleftarrow{\$} \mathbb{Z}_n$
    $m^*_{i,j} := \mathcal{E}\big(pk, \mathsf{LoR}_b(m_{i,j}, \mu_{i,j})\big)$
$M^* := (m^*_{i,j})_{i\in[\![1,a]\!],j\in[\![1,b]\!]}$
for $j \in [\![1,b]\!]$ **do**
  for $k \in [\![1,c]\!]$ **do**
    $\delta_{j,k} \xleftarrow{\$} \mathbb{Z}_n$
    $n^*_{j,k} := \mathcal{E}\big(pk, \mathsf{LoR}_b(n_{j,k}, \delta_{j,k})\big)$
$N^* := (n^*_{j,k})_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
$\mathsf{view} := \big(M^*, N^*, \{(x_{i,j,k}, y_{i,j,k}), z_{i,j,k}\})_{i\in[\![1,a]\!],j\in[\![1,b]\!],k\in[\![1,c]\!]}, \{p^*_{ik}\}_{i\in[\![1,a]\!],k\in[\![1,c]\!]}\big)$
$b_* := D(\mathsf{view})$
return b_*

Figure 3.13: Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 2.

Adversary $\mathcal{A}$ is presented in Figure 3.13. At the end of its execution, $\mathcal{A}$ uses the distinguisher D to compute the bit b^* before returning it. First, we remark that

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

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

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

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

probability that $\mathcal{A}$ wins the experiment, i.e., $b_* = b$

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

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

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

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

Simulator: $\mathcal{S}_{\mathcal{P}}^{\text{CRSP}}(1^\lambda, (pk, sk), P)$

for $i \in [\![1, a]\!]$ **do**
  for $k \in [\![1, c]\!]$ **do**
    for $j \in [\![1, b]\!]$ **do**
      $(r_{i,j,k}, s_{i,j,k}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
      $x_{i,j,k} := \mathcal{E}(pk, r_{i,j,k})$
      $y_{i,j,k} := \mathcal{E}(pk, s_{i,j,k})$
      $z_{i,j,k} := \mathcal{E}(pk, r_{i,j,k} \cdot s_{i,j,k})$
$P^* := (\mathcal{E}(pk, p_{ik}))_{i\in[\![1,a]\!],k\in[\![1,c]\!]}$
$\mathsf{view} := \left(\{(x_{i,j,k}, y_{i,j,k}), z_{i,j,k}\}_{i\in[\![1,a]\!],j\in[\![1,b]\!],k\in[\![1,c]\!]}, P^*\right)$
return view

Figure 3.14: Simulator $\mathcal{S}_{\mathcal{P}}^{\text{CRSP}}$ for the proof of Lemma 3.

Proof. Simulator $\mathcal{S}_{\mathcal{P}}^{\text{CRSP}}$ is presented in Figure 3.14. The view of $P_{\mathcal{P}}$ contains the couple of ciphertexts $(x_{i,j,k}, y_{i,j,k})$ sent by $P_{\mathcal{C}}$ and the answer $z_{i,j,k}$ sent by $P_{\mathcal{P}}$ to $P_{\mathcal{C}}$ that contains the encryption of the multiplication of $x_{i,j,k}$ and $y_{i,j,k}$, for $i \in [\![1, a]\!]$, $j \in [\![1, b]\!]$, and $k \in [\![1, c]\!]$. Since $x_{i,j,k}$ and $y_{i,j,k}$ are randomized by $P_{\mathcal{C}}$, there are indistinguishable to random ciphertexts in the $P_{\mathcal{P}}$ point of view. The view of $P_{\mathcal{P}}$ also contains $P^* := (\mathcal{E}(pk, p_{i,j}))_{i\in[\![1,a]\!],k\in[\![1,c]\!]}$ that is sent by $P_{\mathcal{C}}$. Finally, $\mathcal{S}_{\mathcal{P}}^{\text{CRSP}}(1^\lambda, (pk, sk), P)$ describes exactly the same distribution as $\mathsf{view}_{\mathcal{P}}^{\text{CRSP}}(I, \lambda)$, which concludes the proof. □

3.5.2 Security Proof for the SP-2R Protocol

We start by modeling the SP-2R protocol. The SP approach assumes that public cloud's nodes do not collude, hence we consider each set of nodes of the public cloud as an individual party. We model the SP-2R protocol with seven parties P_M, P_N, $P_{\mathcal{M}}$, $P_{\mathcal{N}}$, $P_{\mathcal{R}_1}$, $P_{\mathcal{R}_2}$, and $P_{\mathcal{P}}$. They use respective inputs $I := (I_M, I_N, I_{\mathcal{M}}, I_{\mathcal{N}}, I_{\mathcal{R}_1}, I_{\mathcal{R}_2}, I_{\mathcal{P}}) \in \mathcal{I}$ and a function $g := (g_M, g_N, g_{\mathcal{M}}, g_{\mathcal{N}}, g_{\mathcal{R}_1}, g_{\mathcal{R}_2}, g_{\mathcal{P}})$ such that

- P_M is the data owner of M. It has the input $I_M := (M, pk)$, where M is its private matrix and pk is the Paillier's public key of the user. P_M returns $g_M(I) := \perp$ because it does not learn anything.
- P_N is the data owner of N. It has the input $I_N := (N, pk_{\mathcal{R}_2})$, where N is its private matrix and $pk_{\mathcal{R}_2}$ is the Paillier's public key of the set of nodes $\mathcal{R}_2$ of the public cloud. P_N returns $g_N(I) := \perp$ because it does not learn anything.

- $P_{\mathcal{M}}$ is the set of nodes $\mathcal{M}$ of the public cloud. It has the input $I_{\mathcal{M}} := pk$ where pk is the Paillier's public key of the user, and returns $g_{\mathcal{M}}(I) := (a, b) \in (\mathbb{N}^*)^2$ because $P_{\mathcal{M}}$ learns the size of M.
- $P_{\mathcal{N}}$ is the set of nodes $\mathcal{N}$ of the public cloud. It has the input $I_{\mathcal{N}} := (pk, pk_{\mathcal{R}_2})$ where pk (resp. $pk_{\mathcal{R}_2}$) is the Paillier's public key of the user (resp. of the set of nodes $\mathcal{R}_2$ of the public cloud). It returns $g_{\mathcal{N}}(I) := (b, c) \in (\mathbb{N}^*)^2$ because $P_{\mathcal{N}}$ learns the size of N.
- $P_{\mathcal{R}_1}$ is the set of nodes $\mathcal{R}_1$ of the public cloud. It has the input $I_{\mathcal{R}_1} := pk$ where pk is the Paillier's public key of the user, and returns $g_{\mathcal{R}_1}(I) := (a, b, c) \in (\mathbb{N}^*)^3$ because $P_{\mathcal{R}_1}$ learns the size of M and N.
- $P_{\mathcal{R}_2}$ is the set of nodes $\mathcal{R}_2$ of the public cloud. It has the input $I_{\mathcal{R}_2} := (pk, pk_{\mathcal{R}_2}, sk_{\mathcal{R}_2})$ where pk is the Paillier's public key of the user, and $(sk_{\mathcal{R}_2}, pk_{\mathcal{R}_2})$ is the Paillier's key pair of $\mathcal{R}_2$. It returns $g_{\mathcal{R}_2}(I) := (a, b, c) \in (\mathbb{N}^*)^3$ because $P_{\mathcal{R}_2}$ learns the size of M and N.
- $P_{\mathcal{P}}$ is the set of nodes $\mathcal{P}$ of the user. It has the input $I_{\mathcal{P}} := (pk, sk)$ where (pk, sk) is the Paillier's key pair of the user. It returns $g_{\mathcal{P}}(I) := P$ because the user learns the result of the matrix multiplication at the end of the protocol.

As for the CRSP protocols, we consider that $P_{\mathcal{R}_2}$ sends the product of the encrypted matrices to $P_{\mathcal{P}}$ instead of storing them in a database.

The security of SP-2R protocol is given by Theorem 3.

Theorem 3. *Assume Paillier's cryptosystem is IND-CPA, then the* SP-2R *protocol securely computes the matrix mutiplication in the presence of semi-honest adversaries if public cloud's nodes do not collude.*

The security proof for the SP-2R protocol is decomposed in Lemma 4 for P_M and P_N, Lemma 5 for $P_{\mathcal{M}}$ and $P_{\mathcal{N}}$, Lemma 7 for $P_{\mathcal{R}_1}$, Lemma 8 for $P_{\mathcal{R}_2}$, and Lemma 9 for $P_{\mathcal{P}}$.

Lemma 4. *There exists probabilistic polynomial-time simulators* $\mathcal{S}_M^{\text{SP-2R}}$ *and* $\mathcal{S}_N^{\text{SP-2R}}$ *such that*

$$\left\{\mathcal{S}_M^{\text{SP-2R}}(1^\lambda, I_M, g_M(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_M^{\text{SP-2R}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}},$$
$$\left\{\mathcal{S}_N^{\text{SP-2R}}(1^\lambda, I_N, g_N(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_N^{\text{SP-2R}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Proof. In SP-2R protocol, P_M has input (M, pk) and sends M^* (the encryption of M) to $P_{\mathcal{M}}$ using Preprocessing I algorithm presented in Figure 3.5 on page 34. Hence, simulator $\mathcal{S}_M^{\text{SP-2R}}$ performs the same computation that Preprocessing I algorithm using M and pk and describes exactly the same distribution as $\mathsf{view}_M^{\text{SP-2R}}(I,\lambda)$. In the same way, the view of P_N contains the encryption of N that is sent to $\mathcal{N}$. Hence, simulator $\mathcal{S}_N^{\text{SP-2R}}$ performs the same computation that Preprocessing II algorithm 3.5 and describes exactly the same distribution as $\mathsf{view}_N^{\text{SP-2R}}(I,\lambda)$. □

Lemma 5. *There exists probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}$ *such that*

$$\left\{\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{M}}, g_{\mathcal{M}}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_{\mathcal{M}}^{\text{SP-2R}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Proof. In the SP-2R protocol, $P_{\mathcal{M}}$ receives the encryption of M from P_M, and sends it to $\mathcal{R}_1$ by rewriting each element in the form of key-value pairs. Hence, the view of $P_{\mathcal{M}}$ only contains the encryption of M. Simulator $\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}$ is given in Figure 3.15.

Simulator: $\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}(1^\lambda, pk, (a,b))$

foreach $i \in [\![1,a]\!]$ **do**
  foreach $j \in [\![1,b]\!]$ **do**
    $\alpha_{i,j} \xleftarrow{\$} \mathbb{Z}_n$
$M^* := (\mathcal{E}(pk, \alpha_{i,j}))_{i\in[\![1,a]\!], j\in[\![1,b]\!]}$
$\mathsf{view}_{\mathcal{M}} := M^*$
return $\mathsf{view}_{\mathcal{M}}$

Figure 3.15: Simulator $\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}$ for the proof of Lemma 5.

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

for $i \in [\![1,a]\!]$ **do**
  for $j \in [\![1,b]\!]$ **do**
    $(m_{i,j}, \alpha_{i,j}) \xleftarrow{\$} \left(\mathbb{Z}_n\right)^2$
    $m^*_{i,j} := \mathcal{E}(pk, \mathsf{LoR}_b(m_{i,j}, \alpha_{i,j}))$
$M^* := (m^*_{i,j})_{i\in[\![1,a]\!], j\in[\![1,b]\!]}$
$\mathsf{view}_{\mathcal{M}} := M^*$
$b_* := D(\mathsf{view}_{\mathcal{M}})$
return b_*

Figure 3.16: Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 5.

Let $\lambda \in \mathbb{N}$ be the security parameter used for the Paillier's cryptosystem. Assume there exists a polynomial-time distinguisher D and a non-negligible function $\mu(\cdot)$ such that for all $I \in \mathcal{I}$, we have

$$\left| \Pr[D(\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{M}}, g_{\mathcal{M}}(I))) = 1] - \Pr[D(\mathsf{view}_{\mathcal{M}}^{\text{SP-2R}}(I,\lambda)) = 1] \right| = \mu(\lambda) \ .$$

We build a probabilistic polynomial-time adversary $\mathcal{A}$ such that it has a non-negligible advantage to win the IND-CPA experiment on the Paillier's cryptosystem. Then we conclude the proof by contraposition. $\mathcal{A}$ is given in Figure 3.16. At the end of its execution, it uses D to compute the bit b_* before returning it.

First, we remark that

$$\Pr\left[\mathsf{Exp}^{\text{indcpa-0}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] = \Pr\left[D(\mathsf{view}_{\mathcal{M}}^{\text{SP-2R}}(I,\lambda)) = 1\right] \ .$$

Indeed, when $b = 0$, the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol SP-2R. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the real protocol.

On the other hand, we have

$$\Pr\left[\mathsf{Exp}^{\text{indcpa-1}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] = \Pr\left[D(\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{M}}, g_{\mathcal{M}}(I))) = 1\right] \ .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}$. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the simulator.

Finally, the probability that $\mathcal{A}$ wins the IND-CPA experiment, i.e., $b_* = b$, is equal to

$$\begin{aligned}
\mathsf{Adv}^{\text{indcpa}}_{\text{Paillier},\mathcal{A}}(\lambda) &= \left| \Pr\left[\mathsf{Exp}^{\text{indcpa-1}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{indcpa-0}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] \right| \\
&= \left| \Pr\left[D(\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{M}}, g_{\mathcal{M}})) = 1\right] - \Pr\left[D(\mathsf{view}_{\mathcal{M}}^{\text{SP-2R}}(I,\lambda)) = 1\right] \right| \\
&= \mu(\lambda) \ ,
\end{aligned}$$

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

Lemma 6. *There exists probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}$ *such that*

$$\left\{\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}(1^{\lambda}, I_{\mathcal{N}}, g_{\mathcal{N}}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_{\mathcal{N}}^{\text{SP-2R}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} .$$

Simulator: $\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}(1^{\lambda}, pk, (b,c))$

foreach $i \in [\![1,b]\!]$ **do**
  foreach $j \in [\![1,c]\!]$ **do**
    $(\beta_{j,k}, \gamma_{jk}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
$N^* := (\beta_{j,k}, \mathcal{E}(pk, \gamma_{j,k}))_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
$\mathsf{view}_{\mathcal{N}} := N^*$
return $\mathsf{view}_{\mathcal{N}}$

Figure 3.17: Simulator $\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}$ for the proof of Lemma 6.

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

for $j \in [\![1,b]\!]$ **do**
  for $k \in [\![1,c]\!]$ **do**
    $(\beta_{j,k}, \tau_{j,k}, \gamma_{j,k}) \xleftarrow{\$} (\mathbb{Z}_n)^3$
    $n^*_{i,j} := (\beta_{j,k}, \mathcal{E}(pk, \mathsf{LoR}_b(\tau_{j,k}, \gamma_{j,k})))$
$N^* := (n^*_{i,j})_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
$\mathsf{view}_{\mathcal{N}} := N^*$
$b_* := D(\mathsf{view}_{\mathcal{N}})$
return b_*

Figure 3.18: Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 6.

Proof. In the SP-2R protocol, P_N uses the Preprocessing II algorithm (cf. Figure 3.5 on page 34) to compute the encryption of N sent to $P_{\mathcal{N}}$, then $P_{\mathcal{N}}$ sends it to $\mathcal{R}_1$ by rewriting each element in the form of key-value pairs. Hence, the view of $P_{\mathcal{N}}$ only contains the encryption of N.

Let $\lambda \in \mathbb{N}$ be the security parameter used for the Paillier's cryptosystem. Assume there exists a polynomial-time distinguisher D and a non-negligible function $\mu(\cdot)$ such that for all $I \in \mathcal{I}$, we have

$$\left|\Pr[D(\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}(1^{\lambda}, I_{\mathcal{M}}, g_{\mathcal{M}}(I))) = 1] - \Pr[D(\mathsf{view}_{\mathcal{N}}^{\text{SP-2R}}(I,\lambda)) = 1]\right| = \mu(\lambda) .$$

We build a probabilistic polynomial-time adversary $\mathcal{A}$ such that it has a non-negligible advantage to win the IND-CPA experiment on the Paillier's cryptosystem. Then we conclude the proof by contraposition. $\mathcal{A}$ is given in Figure 3.18. At the end of its execution, it uses D to compute the bit b_* before returning it.

First, we remark that

$$\Pr\left[\mathsf{Exp}_{\text{Paillier},\mathcal{A}}^{\text{indcpa-0}}(\lambda) = 1\right] = \Pr\left[D(\mathsf{view}_{\mathcal{N}}^{\text{SP-2R}}(I,\lambda)) = 1\right] .$$

Indeed, when $b = 0$, the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol SP-2R. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the real protocol.

On the other hand, we have

$$\Pr\left[\mathsf{Exp}_{\text{Paillier},\mathcal{A}}^{\text{indcpa-1}}(\lambda) = 1\right] = \Pr\left[D(\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}(1^{\lambda}, I_{\mathcal{M}}, g_{\mathcal{M}}(I))) = 1\right] .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}$. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the simulator.

Finally, the probability that $\mathcal{A}$ wins the IND-CPA experiment, i.e., $b_* = b$, is equal to

$$\begin{aligned}\mathsf{Adv}^{\text{indcpa}}_{\text{Paillier},\mathcal{A}}(\lambda) &= \left|\Pr\left[\mathsf{Exp}^{\text{indcpa-1}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{indcpa-0}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right]\right| \\ &= \left|\Pr\left[D(\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{N}}, g_{\mathcal{N}})) = 1\right] - \Pr\left[D(\mathsf{view}_{\mathcal{N}}^{\text{SP-2R}}(I,\lambda)) = 1\right]\right| \\ &= \mu(\lambda)\ ,\end{aligned}$$

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

Lemma 7. *There exists probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{R}_1}^{\text{SP-2R}}$ *such that*

$$\left\{\mathcal{S}_{\mathcal{R}_1}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{R}_1}, g_{\mathcal{R}_1}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}_{\mathcal{R}_1}^{\text{SP-2R}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}\ .$$

Simulator: $\mathcal{S}_{\mathcal{R}_1}^{\text{SP-2R}}(1^\lambda, (pk, pk_{\mathcal{R}_2}), (a,b,c))$

foreach $i \in [\![1,a]\!]$ **do**
  foreach $j \in [\![1,b]\!]$ **do**
    $\alpha_{i,j} \xleftarrow{\$} \mathbb{Z}_n$
    $m^*_{i,j} := \mathcal{E}(pk, \alpha_{i,j})$
foreach $j \in [\![1,b]\!]$ **do**
  foreach $c \in [\![1,c]\!]$ **do**
    $(\beta_{j,k}, \gamma_{j,k}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
    $t_{j,k} := \mathcal{E}(pk_{\mathcal{R}_2}, \gamma_{j,k})$
    $n^*_{j,k} := (\beta_{j,k}, t_{j,k})$
$M^* := (m^*_{i,j})_{i\in[\![1,a]\!], j\in[\![1,b]\!]}$
$N^* := (n^*_{j,k})_{j\in[\![1,b]\!], k\in[\![1,c]\!]}$
foreach $i \in [\![1,a]\!]$ **do**
  foreach $j \in [\![1,b]\!]$ **do**
    foreach $k \in [\![1,c]\!]$ **do**
      $x_{i,j,k} := (m^*_{i,j})^{\beta_{j,k}}$
      $y_{i,j,k} := m^*_{i,j}$
      $z_{i,j,k} := t_{j,k}$
$\mathsf{view}_{\mathcal{R}_1} := \left(M^*, N^*, \{(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})\}_{i\in[\![1,a]\!], j\in[\![1,b]\!], k\in[\![1,c]\!]}\right)$
return $\mathsf{view}_{\mathcal{R}_1}$

Figure 3.19: Simulator $\mathcal{S}_{\mathcal{R}_1}^{\text{SP-2R}}$ for the proof of Lemma 7.

Proof. In the SP-2R protocol, $P_{\mathcal{R}_1}$ receives the encryption of M from $P_{\mathcal{M}}$, the encryption of N from $P_{\mathcal{N}}$, and sends tuples $(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})$ to $P_{\mathcal{R}_2}$ where $i \in [\![1,a]\!]$, $j \in [\![1,b]\!]$, and $k \in [\![1,c]\!]$ that correspond to key-value pairs emited by the first Reduce function executed by $P_{\mathcal{R}_1}$. The simulator for $P_{\mathcal{R}_1}$, denoted $\mathcal{S}_{\mathcal{R}_1}^{\text{SP-2R}}$, is given in Figure 3.19.

Let $\lambda \in \mathbb{N}$ be the security parameter used for the Paillier's cryptosystem. Assume there exists a polynomial-time distinguisher D and a non-negligible function $\mu(\cdot)$ such that for all $I \in \mathcal{I}$, we have

$$\left|\Pr[D(\mathcal{S}_{\mathcal{R}_1}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{M}}, g_{\mathcal{M}}(I))) = 1] - \Pr[D(\mathsf{view}_{\mathcal{R}_1}^{\text{SP-2R}}(I,\lambda)) = 1]\right| = \mu(\lambda)\ .$$

We build a probabilistic polynomial-time adversary $\mathcal{A}$ such that it has a non-negligible advantage to win the IND-CPA experiment on the Paillier's cryptosystem. Then we

Adversary: $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot)),\mathcal{E}(pk_{\mathcal{R}_2},\mathsf{LoR}_b(\cdot,\cdot))}$

for $i \in [\![1,a]\!]$ **do**
  for $j \in [\![1,b]\!]$ **do**
    $(m_{i,j}, \alpha_{i,j}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
    $m^*_{i,j} := \mathcal{E}(pk, \mathsf{LoR}_b(m_{i,j}, \alpha_{i,j}))$
for $j \in [\![1,b]\!]$ **do**
  for $k \in [\![1,c]\!]$ **do**
    $(\beta_{j,k}, \tau_{j,k}, \gamma_{j,k}) \xleftarrow{\$} (\mathbb{Z}_n)^3$
    $t_{j,k} := \mathcal{E}(pk_{\mathcal{R}_1}, \mathsf{LoR}_b(\tau_{j,k}, \gamma_{j,k}))$
    $n^*_{j,k} := (\beta_{j,k}, t_{j,k})$
$M^* := (m^*_{i,j})_{i\in[\![1,a]\!],j\in[\![1,b]\!]}$
$N^* := (n^*_{j,k})_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
for $i \in [\![1,a]\!]$ **do**
  for $j \in [\![1,b]\!]$ **do**
    for $k \in [\![1,c]\!]$ **do**
      $x_{i,j,k} := (m^*_{i,j})^{\beta_{j,k}}$
      $y_{i,j,k} := m^*_{i,j}$
      $z_{i,j,k} := t_{j,k}$
$\mathsf{view}_{\mathcal{R}_1} := \left(M^*, N^*, \{(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})\}_{i\in[\![1,a]\!],j\in[\![1,b]\!],k\in[\![1,c]\!]}\right)$
$b_* := D(\mathsf{view}_{\mathcal{R}_1})$
return b_*

Figure 3.20: Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot)),\mathcal{E}(pk_{\mathcal{R}_2},\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 7.

conclude the proof by contraposition. $\mathcal{A}$ is given in Figure 3.20. At the end of its execution, it uses D to compute the bit b_* before returning it.

First, we remark that

$$\Pr\left[\mathsf{Exp}^{\text{indcpa-0}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] = \Pr\left[D(\mathsf{view}^{\text{SP-2R}}_{\mathcal{R}_1}(I,\lambda)) = 1\right] .$$

Indeed, when $b = 0$, the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol SP-2R. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the real protocol.

On the other hand, we have

$$\Pr\left[\mathsf{Exp}^{\text{indcpa-1}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] = \Pr\left[D(\mathcal{S}^{\text{SP-2R}}_{\mathcal{R}_1}(1^\lambda, I_{\mathcal{M}}, g_{\mathcal{M}}(I))) = 1\right] .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}^{\text{SP-2R}}_{\mathcal{R}_1}$. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the simulator.

Finally, the probability that $\mathcal{A}$ wins the IND-CPA experiment, i.e., $b_* = b$, is equal to

$$\begin{aligned}
\mathsf{Adv}^{\text{indcpa}}_{\text{Paillier},\mathcal{A}}(\lambda) &= \left|\Pr\left[\mathsf{Exp}^{\text{indcpa-1}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{indcpa-0}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right]\right| \\
&= \left|\Pr\left[D(\mathcal{S}^{\text{SP-2R}}_{\mathcal{R}_1}(1^\lambda, I_{\mathcal{R}_1}, g_{\mathcal{R}_1})) = 1\right] - \Pr\left[D(\mathsf{view}^{\text{SP-2R}}_{\mathcal{R}_1}(I,\lambda)) = 1\right]\right| \\
&= \mu(\lambda) ,
\end{aligned}$$

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

Lemma 8. *There exists probabilistic polynomial-time simulator* $\mathcal{S}^{\text{SP-2R}}_{\mathcal{R}_2}$ *such that*

$$\left\{\mathcal{S}^{\text{SP-2R}}_{\mathcal{R}_2}(1^\lambda, I_{\mathcal{R}_2}, g_{\mathcal{R}_2}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}^{\text{SP-2R}}_{\mathcal{R}_2}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} .$$

Simulator: $\mathcal{S}_{\mathcal{R}_2}^{\text{SP-2R}}(1^\lambda, (pk, pk_{\mathcal{R}_2}, sk_{\mathcal{R}_2}), (a, b, c))$

for $i \in [\![1, a]\!]$ **do**
  for $j \in [\![1, b]\!]$ **do**
    $\alpha_{i,j} \xleftarrow{\$} \mathbb{Z}_n$
    $m^*_{i,j} := \mathcal{E}(pk, \alpha_{i,j})$
for $j \in [\![1, b]\!]$ **do**
  for $k \in [\![1, c]\!]$ **do**
    $(\beta_{j,k}, \tau_{j,k}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
    $t_{j,k} := \mathcal{E}(pk_{\mathcal{R}_2}, \tau_{j,k})$
    $n^*_{j,k} := (\beta_{j,k}, t_{j,k})$
$M^* := (m^*_{i,j})_{i \in [\![1,a]\!], j \in [\![1,b]\!]}$
$N^* := (n^*_{j,k})_{j \in [\![1,b]\!], k \in [\![1,c]\!]}$
for $i \in [\![1, a]\!]$ **do**
  for $k \in [\![1, c]\!]$ **do**
    for $j \in [\![1, b]\!]$ **do**
      $x_{i,j,k} := (m^*_{i,j})^{\beta_{j,k}}$
      $y_{i,j,k} := m^*_{i,j}$
      $z_{i,j,k} := t_{j,k}$
    $p^*_{ik} := \prod_{j=1}^{b} x_{i,j,k} \cdot \left(y_{i,j,k}^{\mathcal{D}(sk_{\mathcal{R}_2}, z_{i,j,k})}\right)^{-1}$
$P^* := (p^*_{ik})_{i \in [\![1,a]\!], j \in [\![1,b]\!], k \in [\![1,c]\!]}$
$\mathsf{view}_{\mathcal{R}_2} := \left(M^*, N^*, \{(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})\}_{i \in [\![1,a]\!], j \in [\![1,b]\!], k \in [\![1,c]\!]}, P^*\right)$
return $\mathsf{view}_{\mathcal{R}_2}$

Figure 3.21: Simulator $\mathcal{S}_{\mathcal{R}_2}^{\text{SP-2R}}$ for the proof of Lemma 8.

Proof. In the SP-2R protocol, $P_{\mathcal{R}_2}$ receives tuples $(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})$, where $i \in [\![1, a]\!]$, $i \in [\![1, a]\!]$, and $i \in [\![1, a]\!]$, computed by the first Reduce function and sent by $P_{\mathcal{R}_1}$. $P_{\mathcal{R}_2}$ sends the encryption of P, denoted P^*. The simulator for $P_{\mathcal{R}_2}$, denoted $\mathcal{S}_{\mathcal{R}_2}^{\text{SP-2R}}$, is given in Figure 3.21.

Let $\lambda \in \mathbb{N}$ be the security parameter used for the Paillier's cryptosystem. Assume there exists a polynomial-time distinguisher D and a non-negligible function $\mu(\cdot)$ such that for all $I \in \mathcal{I}$, we have

$$\left| \Pr[D(\mathcal{S}_{\mathcal{R}_2}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{M}}, g_{\mathcal{M}}(I))) = 1] - \Pr[D(\mathsf{view}_{\mathcal{R}_2}^{\text{SP-2R}}(I, \lambda)) = 1] \right| = \mu(\lambda) \ .$$

We build a probabilistic polynomial-time adversary $\mathcal{A}$ such that it has a non-negligible advantage to win the IND-CPA experiment on the Paillier's cryptosystem. Then we conclude the proof by contraposition. $\mathcal{A}$ is given in Figure 3.22. At the end of its execution, it uses D to compute the bit b_* before returning it.

First, we remark that

$$\Pr\left[\mathsf{Exp}_{\text{Paillier},\mathcal{A}}^{\text{indcpa-0}}(\lambda) = 1\right] = \Pr\left[D(\mathsf{view}_{\mathcal{R}_2}^{\text{SP-2R}}(I, \lambda)) = 1\right] \ .$$

Indeed, when $b = 0$, the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol SP-2R. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the real protocol.

On the other hand, we have

$$\Pr\left[\mathsf{Exp}_{\text{Paillier},\mathcal{A}}^{\text{indcpa-1}}(\lambda) = 1\right] = \Pr\left[D(\mathcal{S}_{\mathcal{R}_2}^{\text{SP-2R}}(1^\lambda, I_{\mathcal{M}}, g_{\mathcal{M}}(I))) = 1\right] \ .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{R}_2}^{\text{SP-2R}}$. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the simulator.

Adversary: $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot)),\mathcal{E}(pk_{\mathcal{R}_2},\mathsf{LoR}_b(\cdot,\cdot))}$

for $i \in [\![1,a]\!]$ **do**
  for $j \in [\![1,b]\!]$ **do**
    $(m_{i,j}, \alpha_{i,j}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
    $m^*_{i,j} := \mathcal{E}(pk, \mathsf{LoR}_b(m_{i,j}, \alpha_{i,j}))$
for $j \in [\![1,b]\!]$ **do**
  for $k \in [\![1,c]\!]$ **do**
    $(\beta_{j,k}, \tau_{j,k}, \gamma_{j,k}) \xleftarrow{\$} (\mathbb{Z}_n)^3$
    $t_{j,k} := \mathcal{E}(pk_{\mathcal{R}_2}, \mathsf{LoR}_b(\tau_{j,k}, \gamma_{j,k}))$
    $n^*_{j,k} := (\beta_{j,k}, t_{j,k})$
$M^* := (m^*_{i,j})_{i\in[\![1,a]\!],j\in[\![1,b]\!]}$
$N^* := (n^*_{j,k})_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
for $i \in [\![1,a]\!]$ **do**
  for $k \in [\![1,c]\!]$ **do**
    for $j \in [\![1,b]\!]$ **do**
      $x_{i,j,k} := (m^*_{i,j})^{\beta_{j,k}}$
      $y_{i,j,k} := m^*_{i,j}$
      $z_{i,j,k} := t_{j,k}$
    $p^*_{ik} := \prod_{j=1}^{b} x_{i,j,k} \cdot \left(y_{i,j,k}^{\mathcal{D}(sk_{\mathcal{R}_2}, z_{i,j,k})}\right)^{-1}$
$P^* := (p^*_{ik})_{i\in[\![1,a]\!],k\in[\![1,c]\!]}$
$\mathsf{view}_{\mathcal{R}_2} := \left(M^*, N^*, \{(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})\}_{i\in[\![1,a]\!],j\in[\![1,b]\!],k\in[\![1,c]\!]}, P^*\right)$
$b_* := D(\mathsf{view}_{\mathcal{R}_2})$
return b_*

Figure 3.22: Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot)),\mathcal{E}(pk_{\mathcal{R}_2},\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 8.

Finally, the probability that $\mathcal{A}$ wins the IND-CPA experiment, i.e., $b_* = b$, is equal to

$$\begin{aligned}\mathsf{Adv}^{\text{indcpa}}_{\text{Paillier},\mathcal{A}}(\lambda) &= \left|\Pr\left[\mathsf{Exp}^{\text{indcpa-1}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{indcpa-0}}_{\text{Paillier},\mathcal{A}}(\lambda) = 1\right]\right| \\ &= \left|\Pr\left[D(\mathcal{S}^{\text{SP-2R}}_{\mathcal{R}_2}(1^\lambda, I_{\mathcal{R}_2}, g_{\mathcal{R}_2})) = 1\right] - \Pr\left[D(\mathsf{view}^{\text{SP-2R}}_{\mathcal{R}_2}(I,\lambda)) = 1\right]\right| \\ &= \mu(\lambda)\ ,\end{aligned}$$

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

Lemma 9. *There exists a probabilistic polynomial-time simulator* $\mathcal{S}^{\text{SP-2R}}_{\mathcal{P}}$ *such that*

$$\left\{\mathcal{S}^{\text{SP-2R}}_{\mathcal{P}}(1^\lambda, I_{\mathcal{P}}, g_{\mathcal{P}}(I))\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\mathsf{view}^{\text{SP-2R}}_{\mathcal{P}}(I,\lambda)\right\}_{I\in\mathcal{I},\lambda\in\mathbb{N}}.$$

Simulator: $\mathcal{S}^{\text{SP-2R}}_{\mathcal{P}}(1^\lambda, (pk, sk), P)$

$(p_{i,k})_{i[\![1,a]\!],k\in[\![1,c]\!]} := P$
$P^* := (\mathcal{E}(pk, p_{i,k}))_{i[\![1,a]\!],k\in[\![1,c]\!]}$
$\mathsf{view}_{\mathcal{P}} := P^*$
return $\mathsf{view}_{\mathcal{P}}$

Figure 3.23: Simulator $\mathcal{S}^{\text{SP-2R}}_{\mathcal{P}}$ for the proof of Lemma 9.

Proof. Simulator $\mathcal{S}^{\text{SP-2R}}_{\mathcal{P}}$ is presented in Figure 3.23. The view of $P_{\mathcal{P}}$ only contains the encryption of elements of matrix P sent by $P_{\mathcal{R}_2}$. Hence, $\mathcal{S}^{\text{SP-2R}}_{\mathcal{P}}$ describes exactly the same distribution as $\mathsf{view}^{\text{SP-2R}}_{\mathcal{P}}(I,\lambda)$, which concludes the proof. □

3.5.3 Security Proof for the SP-1R Protocol

We start by modeling the SP-1R. The SP assumes that public cloud's nodes do not collude, hence we consider each set of nodes of the public cloud as an individual party. We model the SP-1R protocol with six parties P_M, P_N, $P_{\mathcal{M}}$, $P_{\mathcal{N}}$, $P_{\mathcal{R}}$, and $P_{\mathcal{P}}$. They use respective inputs $I := (I_M, I_N, I_{\mathcal{M}}, I_{\mathcal{N}}, I_{\mathcal{R}}, I_{\mathcal{M}}) \in \mathcal{I}$ and a function $g := (g_M, g_N, g_{\mathcal{M}}, g_{\mathcal{N}}, g_{\mathcal{R}}, g_{\mathcal{P}})$ such that

- P_M is the data owner of M. It has the input $I_M := (M, pk)$, where M is its private matrix and pk is the Paillier's public key of the user. P_M returns $g_M(I) := \perp$ because it does not learn anything.
- P_N is the data owner of N. It has the input $I_N := (N, pk)$, where N is its private matrix and pk is the Paillier's public key of the user. P_N returns $g_N(I) := \perp$ because it does not learn anything.
- $P_{\mathcal{M}}$ is the set of nodes $\mathcal{M}$ of the public cloud. It has the input $I_{\mathcal{M}} := pk$ where pk is the Paillier's public key of the user, and returns $g_{\mathcal{M}}(I) := (a, b) \in (\mathbb{N}^*)^2$ because $P_{\mathcal{M}}$ learns the size of M.
- $P_{\mathcal{N}}$ is the set of nodes $\mathcal{N}$ of the public cloud. It has the input $I_{\mathcal{N}} := pk$ where pk is the Paillier's public key of the user, and returns $g_{\mathcal{N}}(I) := (N, a, b)$ where $(a, b) \in (\mathbb{N}^*)^2$ because $P_{\mathcal{N}}$ learns the matrix N.
- $P_{\mathcal{R}}$ is the set of nodes $\mathcal{R}$ of the public cloud. It has the input $I_{\mathcal{R}} := pk$ where pk is the Paillier's public key of the user. It returns $g_{\mathcal{R}}(I) := (N, a, b, c)$ because $P_{\mathcal{R}}$ learns the matrix N and dimensions of matrices M and N.
- $P_{\mathcal{P}}$ is the set of nodes $\mathcal{P}$ of the user. It has the input $I_{\mathcal{P}} := (pk, sk)$ where (pk, sk) is a Paillier's key pair of the user, and returns $g_{\mathcal{P}}(I) := P$ because the user learns the result of the matrix multiplication at the end of the protocol.

The security of the SP-1R protocol is given by Theorem 4.

Theorem 4. *Assume Paillier's cryptosystem is IND-CPA, then the* SP-1R *protocol securely computes the matrix mutiplication in the presence of semi-honest adversaries.*

The proof for Theorem 4 is decomposed in Lemma 10 for P_M, Lemma 11 for P_N, Lemma 12 for $P_{\mathcal{M}}$, Lemma 13 for $P_{\mathcal{N}}$, Lemma 14 for $P_{\mathcal{R}}$, and Lemma 15 for $P_{\mathcal{P}}$.

Lemma 10. *There exists probabilistic polynomial-time simulator* $\mathcal{S}_M^{\text{SP-1R}}$ *such that*

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

Proof of Lemma 10 is the same than Lemma 1.

Lemma 11. *There exists probabilistic polynomial-time simulator* $\mathcal{S}_N^{\text{SP-1R}}$ *such that*

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

Proof. In the SP-1R protocol, P_N as for input the matrix N. P_N does not perform preprocessing on N, and sends it to $P_{\mathcal{N}}$. Hence, $\mathcal{S}_N^{\text{SP-1R}}$ is the identity function and describes exactly the same distribution as $\mathsf{view}_N^{\text{SP-1R}}(I, \lambda)$. □

Lemma 12. *There exists probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{M}}^{\text{SP-1R}}$ *such that*

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

Proof of Lemma 12 is the same than Lemma 5.

Lemma 13. *There exists probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{N}}^{\text{SP-1R}}$ *such that*

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

Proof. In the SP-1R protocol, $P_{\mathcal{N}}$ receives N from P_N, and executes the Map function (cf. Figure 3.7 on page 36) to send output to $P_{\mathcal{R}}$ by rewriting each element of N in the form of key-value pairs. Hence, the view of $P_{\mathcal{N}}$ only contains N. Therefore, $\mathcal{S}_{\mathcal{N}}^{\text{SP-1R}}$ is the identity function and describes exactly the same distribution as $\mathsf{view}_{\mathcal{N}}^{\text{SP-1R}}(I,\lambda)$. □

Lemma 14. *There exists probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{R}}^{\text{SP-1R}}$ *such that*

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

Simulator: $\mathcal{S}_{\mathcal{R}}^{\text{SP-1R}}(1^\lambda, pk, (N,a,b,c))$

foreach $i \in [\![1,a]\!]$ **do**
  foreach $j \in [\![1,b]\!]$ **do**
    $\alpha_{i,j} \xleftarrow{\$} \mathbb{Z}_n$
    $m^*_{i,j} := \mathcal{E}(pk, \alpha_{i,j})$
$M^* := (m^*_{i,j})_{i\in[\![1,a]\!],j\in[\![1,b]\!]}$
foreach $j \in [\![1,b]\!]$ **do**
  foreach $k \in [\![1,c]\!]$ **do**
    $n_{j,k} \xleftarrow{\$} \mathbb{Z}_n$
$N := (n_{j,k})_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
foreach $i \in [\![1,a]\!]$ **do**
  foreach $k \in [\![1,c]\!]$ **do**
    foreach $j \in [\![1,b]\!]$ **do**
      $x_{i,j,k} := (m^*_{i,j})^{n_{j,k}}$
    $p^*_{ik} := \prod_{j=1}^{b} x_{i,j,k}$
$P^* := (p^*_{ik})_{i\in[\![1,a]\!],k\in[\![1,c]\!]}$
$\mathsf{view}_{\mathcal{R}} := (M^*, N, P^*)$
return $\mathsf{view}_{\mathcal{R}}$

Figure 3.24: Simulator $\mathcal{S}_{\mathcal{R}}^{\text{SP-1R}}$ for the proof of Lemma 14.

Proof. In the SP-1R protocol, $P_{\mathcal{R}}$ receives M^* (the encryption of M) from P_M, and N from P_N that corresponds to the output of the Map function (cf. Figure 3.7 on page 36). Hence, it learns N, and dimensions of M and N. It sends P^* (the encryption of P) to $P_{\mathcal{P}}$. Simulator $\mathcal{S}_{\mathcal{R}}^{\text{SP-1R}}$ for $P_{\mathcal{R}}$ is given in Figure 3.24.

Let $\lambda \in \mathbb{N}$ be a security parameter used for the Paillier's cryptosystem. Assume there exists a polynomial-time distinguisher D and a non-negligible function $\mu(\cdot)$ such that for all $I \in \mathcal{I}$, we have

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

We build a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}$ has a non-negligible advantage to win the IND-CPA experiment on the Paillier's cryptosystem. Then we conclude the proof by contraposition. Adversary $\mathcal{A}$ is presented in Figure 3.25. At the end of its execution, $\mathcal{A}$ uses the distinguisher D to compute the bit b_* before returning it.

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

foreach $i \in [\![1,a]\!]$ **do**
  foreach $j \in [\![1,b]\!]$ **do**
    $(m_{i,j},\alpha_{i,j}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
    $m^*_{i,j} := \mathcal{E}(pk,\mathsf{LoR}_b(m_{i,j},\alpha_{i,j}))$
$M^* := (m^*_{i,j})_{i\in[\![1,a]\!],j\in[\![1,b]\!]}$
foreach $j \in [\![1,b]\!]$ **do**
  foreach $k \in [\![1,c]\!]$ **do**
    $n_{j,k} \xleftarrow{\$} \mathbb{Z}_n$
$N := (n_{j,k})_{j\in[\![1,b]\!],k\in[\![1,c]\!]}$
foreach $i \in [\![1,a]\!]$ **do**
  foreach $k \in [\![1,c]\!]$ **do**
    foreach $j \in [\![1,b]\!]$ **do**
      $x_{i,j,k} := (m^*_{i,j})^{n_{j,k}}$
    $p^*_{ik} := \prod_{j=1}^{b} x_{i,j,k}$
$P^* := (p^*_{ik})_{i\in[\![1,a]\!],k\in[\![1,c]\!]}$
$\mathsf{view}_{\mathcal{R}} := (M^*,N,P^*)$
$b_* := D(\mathsf{view}_{\mathcal{R}})$
return b_*

Figure 3.25: Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot)),\mathcal{E}(pk_{\mathcal{R}_2},\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 14.

First, we remark that

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

Indeed, when $b=0$, the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol SP-1R. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the real protocol.

On the other hand, we have

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

When $b=1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{R}}$. Then the probability that the experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the simulator.

Finally, the probability that $\mathcal{A}$ wins the IND-CPA experiment, i.e., $b_*=b$, is equal to

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

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

Lemma 15. *There exists a probabilistic polynomial-time simulator $\mathcal{S}^{\text{SP-1R}}_{\mathcal{P}}$ such that*

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

Proof of Lemma 15 is the same than Lemma 9.

3.6 Conclusion

We have presented efficient protocols for MapReduce matrix multiplication that enjoy privacy guarantees such as: none of the nodes storing an input matrix can learn the other input matrix or the output matrix, and moreover, none of the nodes computing an intermediate result can learn the input or the output matrices. To achieve our goal, we have relied on Paillier's cryptosystem and we developed two different approaches: one resisting to collusions between public cloud's nodes called CRSP for Collision-Resistant-Secure-Private, and another which does not require communication overhead called SP for Secure-Private. We have thoroughly compared these two approaches with respect to three fundamental criteria: computation cost, communication cost, and privacy guarantees.

Looking forward to future work, we aim to investigate the matrix multiplication with privacy guarantees in different big data systems (such as Spark or Flink) whose users also tend to outsource data and computations similarly to MapReduce.

CHAPTER 4

Secure Strassen-Winograd Matrix Multiplication with MapReduce

We propose a secure multiparty protocol to compute matrix multiplication with the Strassen-Winograd algorithm using the MapReduce paradigm. The Strassen-Winograd algorithm (SW) is one of the most efficient matrix multiplication algorithm. Our first contribution is to redesign SW algorithm with the MapReduce programming model that allows to process big data sets in parallel on a cluster. Moreover, our main contribution is to address the inherent security and privacy concerns that occur when outsourcing data to a public cloud. We propose a secure approach of SW with MapReduce denoted S2M3, for *Secure Strassen-winograd Matrix Multiplication with Mapreduce*. This work has been conducted in collaboration with Radu Ciucanu, Pascal Lafourcade, and Lihua Ye, and has been published in the paper "*Secure Strassen-Winograd Matrix Multiplication with MapReduce*" at SECRYPT 2019 conference.

Contents

4.1 Introduction 57
4.1.1 Problem Statement 57
4.1.2 Contributions 57
4.1.3 Outline 58
4.2 Strassen-Winograd Algorithm 58
4.2.1 Strassen-Winograd Algorithm for 2-Power Size Matrices 58
4.2.2 Padding and Peeling: On a Quest for All Dimensions 59
4.3 Strassen-Winograd Matrix Multiplication 60
4.3.1 Strassen-Winograd MapReduce Protocol 60
4.3.2 Strassen-Winograd MapReduce Protocol with the Dynamic Padding Method 63
4.3.3 Strassen-Winograd MapReduce Protocol with the Dynamic Peeling Method 65
4.4 Secure Strassen-Winograd Matrix Multiplication 67
4.4.1 Preprocessing for Secure Strassen-Winograd Matrix Multiplication 67
4.4.2 Secure Approach 69
4.4.3 Secure Strassen-Winograd Matrix Multiplication Protocol 69
4.4.4 Secure Strassen-Winograd Matrix Multiplication with the Dynamic Padding Method 70
4.4.5 Secure Strassen-Winograd Matrix Multiplication with the Dynamic Peeling Method 71

4.5 Experimental Results 73
4.5.1 Dataset and Settings 73
4.5.2 Results 76
4.6 Security Proofs 76
4.6.1 Security Proof for S2M3 Protocol 77
4.6.2 Security Proof for S2M3-Pad Protocol 81
4.6.3 Security Proof for S2M3-Peel Protocol 82
4.7 Conclusion 82

4.1 Introduction

Many of applications deal with matrices whose size is often very large. Whereas a naive matrix multiplication algorithm has cubic complexity, many research efforts have been made to propose more efficient algorithms. One of the most efficient algorithms is Strassen-Winograd [Mac16] (denoted as SW in the sequel), the first sub-cubic time algorithm, with an exponent $\log_2 7 \approx 2.81$. The best algorithm known to date [Gal14] has an exponent ≈ 2.38. Although many of the sub-cubic algorithms are not necessarily suited for practical use as their hidden constant in the big-O notation is huge, the SW algorithm and its variants emerged as a class of matrix multiplication algorithms in widespread use.

4.1.1 Problem Statement

Two distinct data owners respectively hold compatible square matrices M and N of dimension $d \in \mathbb{N}^*$. A user (who does not know the matrices M and N) wants their product $P := MN$. M and N are sent to the distributed file system of some public cloud provider. We assume that the matrices M (resp. N) is initially spread over a set $\mathcal{M}$ (resp. $\mathcal{N}$) of nodes of the public cloud storing a chunk of M (resp. N), i.e., a set of elements of M (resp. N). The final result P is computed over sets of nodes $\mathcal{D}_1, \ldots, \mathcal{D}_\ell, \mathcal{C}_1, \ldots, \mathcal{C}_\ell$ before it is sent to the user's nodes $\mathcal{P}$, where ℓ depends on the dimension d. Moreover, we assume that data owners (resp. the user) cannot collude with the public cloud and the user (resp. the public cloud and data owners). We illustrate the architecture of Strassen-Winograd matrix multiplication with MapReduce in Figure 4.1.

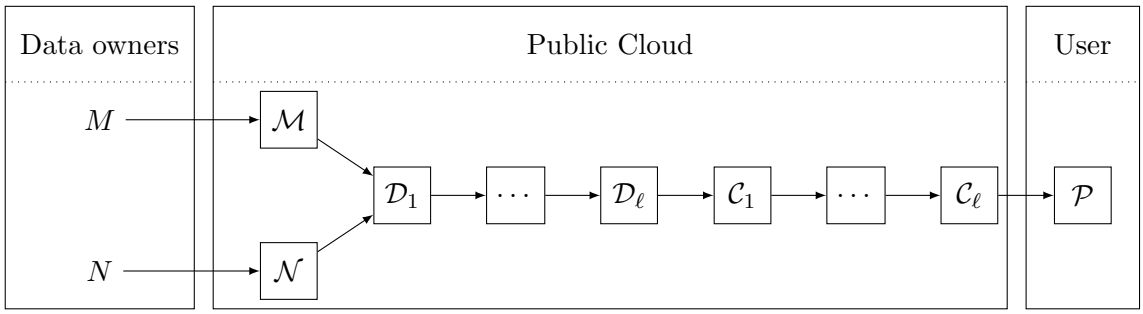


Figure 4.1: Architecture of Strassen-Winograd matrix multiplication with MapReduce.

We expect the following properties:

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

4.1.2 Contributions

We redesign the Strassen-Winograd algorithm for the MapReduce paradigm model, and propose a secure approach that satisfies our aforementioned problem statement. More precisely:

- Our first contribution is a MapReduce version of the SW matrix multiplication algorithm. We call our protocol SM3 for *Strassen-Winograd Matrix Multiplication with MapReduce*. It improves the efficiency of the computation compared to the standard matrix multiplication with MapReduce, as found in Chapter 2 of [LRU14].
- Our second contribution is a privacy-preserving version of the aforementioned protocol. Our new protocol S2M3 (for *Secure Strassen-Winograd Matrix Multiplication*

with MapReduce) relies on the MapReduce paradigm and on Paillier-like public-key cryptosystem. The public cloud performs the multiplication on the encrypted data. At the end of the computation, the public cloud sends the result to the user that queried the matrix multiplication result. The user has just to decrypt the result to discover the matrix multiplication result. The public cloud cannot learn none of the input or output matrices. We formally prove, using the standard security model, that our S2M3 protocol satisfies the aforementioned security property. Moreover, we assume that cloud's nodes may collude that give us the possibility to avoid the recursivity in the security proof.

- To show the practical efficiency of SM3 and S2M3 protocols, we present a proof-of-concept using the Apache Hadoop [Fou19b] open-source implementation of MapReduce.

4.1.3 Outline

We start by recalling the Strassen-Winograd algorithm in Section 4.2. Then, we present our MapReduce protocol for Strassen-Winograd algorithm in Section 4.3. In Section 4.4, we present our secure approach for Strassen-Winograd matrix multiplication with MapReduce. In Section 4.5, we show an experimental evaluation of our two MapReduce protocols (with and without security) using Hadoop [Fou19b], the Apache MapReduce implementation. Before to conclude, we prove in Section 4.6 the security of our protocol in the standard model.

4.2 Strassen-Winograd Algorithm

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

4.2.1 Strassen-Winograd Algorithm for 2-Power Size Matrices

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

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

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

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

- 8 additions

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

- 7 recursive SW matrix multiplications

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

- 7 final additions

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

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

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

4.2.2 Padding and Peeling: On a Quest for All Dimensions

Three methods, namely *static padding*, *dynamic padding* and *dynamic peeling* [HJJ+96] allow to perform Strassen-Winograd matrix multiplication with two compatible square matrices of arbitrary dimension, i.e., dimension that is equal to a 2-power integer or not.

Static Padding

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

Dynamic Padding

The *dynamic padding* method checks if the dimension of original matrices M and N is even or not. If not, it pads matrices with an extra column and and extra row of zeros. In that way, SW is able to produce the four quadrants of equal dimension and the 8 additions as described in Section 4.2.

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

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

Dynamic Peeling

Instead of adding an extra row and an extra column to make matrices even sized as in the dynamic padding method, the *dynamic peeling* method removes a row and a column. Let M and N be two square matrices of size d. If d is an odd number, the dynamic peeling builds four quadrants for each matrix as illustrated in Figure 4.2.

Since d is an odd number, quadrants $M_{11} := (m_{i,j})_{i,j\in[\![1,d-1]\!]}$ and $N_{11} := (n_{i,j})_{i,j\in[\![1,d-1]\!]}$ are square matrices of even size. Moreover, we define $M_{12} := (m_{i,d})_{i\in[\![1,d-1]\!]}$, $M_{21} :=$

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

Figure 4.2: Dynamic peeling for matrices M and N.

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

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

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

4.3 Strassen-Winograd Matrix Multiplication with MapReduce

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

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

4.3.1 Strassen-Winograd MapReduce Protocol

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

Deconstruction Phase

We present the deconstruction phase of SM3. The Map function (resp. the Reduce function) of the deconstruction phase is presented in Figure 4.3 (resp. Figure 4.4).

- *The Map Function.* It is run only during the first MapReduce round of the deconstruction phase by sets of nodes $\mathcal{M}$ and $\mathcal{N}$. It consists in rewritting each matrix element sent by data owners in the form of key-value pair such that they share the same key initialized to 0. Hence, when the set of nodes $\mathcal{M}$ receives chunks of M from the owner, the Map function creates for each matrix element $m_{i,j}$ the key-value

Map function:

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

Figure 4.3: Map function for the deconstruction phase of the SM3 protocol.

Reduce function:

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

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

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

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

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

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

Figure 4.4: Reduce function for the deconstruction phase of the SM3 protocol.

pair $(0, (\texttt{M}, i, j, m_{i,j}, d))$, where d is the dimension of M. Likewise, when the set of nodes $\mathcal{N}$ receives chunks of N from the owner, the Map function creates for each matrix element $n_{k,l}$ the key-value pair $(0, (\texttt{N}, k, l, n_{k,l}, d))$. We stress that `M` and `N` in the values are the names of matrices, that can be encoded with a single bit, and not the matrices themselves. During other rounds of the deconstruction phase, the Map function is the identity function.

- *The Reduce Function.* It is executed by nodes $\mathcal{D}_s$, with $s \in [\![1, \ell]\!]$. Each key is

associated to two matrices sent from previous nodes. When $s = 1$, matrices are M and N and are sent by nodes $\mathcal{M}$ and $\mathcal{N}$. When $s \in [\![2, \ell]\!]$, the two matrices correspond to a recursive matrix multiplication and are sent by $\mathcal{D}_{s-1}$. The Reduce function follows the Strassen-Winograd algorithm using these two matrices. Since the Strassen-Winograd algorithm needs to compute 7 recursive matrix multiplications, the Reduce function produces key-value pairs for 7 different keys where each key is associated to a pair of submatrices to multiply. These key-value pairs are sent to the next nodes of the deconstruction phase $\mathcal{D}_{s+1}$. For the last round of the deconstruction phase, i.e., when $s = \ell$, matrix multiplications are degenerated into scalar multiplications. Hence, the Reduce function produces key-values pairs with the result of scalar multiplications and sends them to the set of nodes $\mathcal{C}_1$.

Combination Phase

We present the combination phase of SM3. In this phase, the Map function is just the identity function. The Reduce function of the combination phase is presented in Figure 4.5.

Reduce function:

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

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

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

$$(p_{v,w})_{v,w \in [\![1, 2\cdot\delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$$

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

Figure 4.5: Reduce function for the combination phase of the SM3 protocol.

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

4.3.2 Strassen-Winograd MapReduce Protocol with the Dynamic Padding Method

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

Deconstruction Phase

We present the deconstruction phase of SM3-Pad. The difference compared to the SM3 protocol is the use of the dynamic padding in the Reduce function. Hence, each time it is required, the Reduce function adds an extra column and an extra row of zeros to both matrices in order they have an even size dimension. The Map function (resp. the Reduce function) of the deconstruction phase is presented in Figure 4.6 (resp. Figure 4.7).

Map function:

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

Figure 4.6: Map function for the deconstruction phase of the SM3-Pad protocol.

- *The Map Function.* It is executed only during the first MapReduce round of the deconstruction phase by the set of nodes $\mathcal{D}_1$. The only difference with the Map function of SM3 is the adding of the padding flag denoted pad and initialized to the empty string ε. For other rounds, the Map function is the identity function.
- *The Reduce Function.* It is executed during each MapReduce round of the deconstruction phase. The difference with the Reduce function of SM3 is that before to split both matrices formed from received key-value pairs, it checks if the matrices' dimension is odd or not. If that is the case, the padding flag pad is updated, i.e., it concatenates the character P (for padding), otherwise it concatenated the character E (for even). Moreover, when the matrices' dimension is odd, the Reduce function adds an extra column and an extra row of zeros to both matrices. Hence, matrices have an even dimension and can be splitted as in the Reduce function of SM3. Since, the Reduce function adds an extra dimensions to matrices when their dimension is odd, the deconstruction phase runs on $\lceil \log_2(d) \rceil$ MapReduce rounds.

Combination Phase

We present the combination phase of SM3-Pad. This phase deals with the extra column and the extra row added during the deconstruction phase using the padding flag pad introduced in the deconstruction phase. Indeed, when two padded matrices are multiplied, the result has also an extra column and an extra row. In this phase, the Map function is just the identity function. The Reduce function of the combination phase is presented in Figure 4.8.

Reduce function:

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

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

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

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

Figure 4.7: Reduce function for the deconstruction phase of the SM3-Pad protocol.

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

Reduce function:

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

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

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

$$(p_{k,l})_{k,l \in [\![1, 2 \cdot \delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$$

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

Figure 4.8: Reduce function for the combination phase of the SW-Pad protocol.

4.3.3 Strassen-Winograd MapReduce Protocol with the Dynamic Peeling Method

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

Deconstruction Phase

We present the deconstruction phase of SM3-Peel. When it is required, i.e., when matrices' dimension is odd, it uses the dynamic peeling to split the two matrices. The Map function is exactly the same than the Map function of the deconstruction phase of the SM3 protocol and is presented in Figure 4.3. The Reduce function of the deconstruction phase is presented in Figure 4.15.

- *The Map Function.* It is executed only during the first MapReduce round of the deconstruction phase, i.e., by set of nodes $\mathcal{D}_1$. It works as the Map function of SM3 protocol. The Map function for other rounds of the deconstruction phase is the identity function.

- *The Reduce Function.* It is executed during each MapReduce round of the deconstruction phase, i.e., by sets of nodes $\mathcal{D}_s$ with $s \in [\![1, \ell]\!]$ where $\ell := \lfloor \log_2(d) \rfloor$. When $\mathcal{D}_s$ receives for a certain key the two matrices to multiply denoted M_0 and N_0, it checks the parity of their dimension. If it is odd, the Reduce function splits M_0 and

Reduce function:

Input: $(key, values)$
// *key*: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 6]\!]$ for $z \in [\![0, e]\!]$
// *values*: collection of $(\mathtt{M}, i, j, m_{ij}, \delta)$ or $(\mathtt{N}, j, k, n_{jk}, \delta)$ or $(\mathtt{P}, j, k, p_{jk}, \delta_p)$

foreach $(\mathtt{P}, i, j, p_{ij}, \delta_p) \in values$ **do**
  $\mathtt{emit}(t_0 \cdots t_e, (\mathtt{Q}, i, j, p_{ij}, \delta_p))$

$M := (m_{ij})_{(\mathtt{M},i,j,m_{ij},\delta) \in values}$
$N := (n_{jk})_{(\mathtt{N},j,k,n_{jk},\delta) \in values}$

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

Figure 4.9: Reduce function for the deconstruction phase of the SW-Peel protocol.

N_0 using the dynamic peeling method and obtains

$$M_0 := \begin{bmatrix} M' & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \ , \quad N_0 := \begin{bmatrix} N' & N_{12} \\ N_{21} & N_{22} \end{bmatrix} \ .$$

Then, it follows the Strassen-Winograd algorithm for the multiplication between M'

and N' blocks, and compute standard matrix multiplication for other blocks.

Otherwise, the Reduce function follows the Strassen-Winograd algorithm with matrices M_0 and N_0.

Combination Phase

We present the combination phase of SM3-Peel. This phase combines results of recursive matrix multiplications to compute matrix $P := MN$. As for the combination phase of the SM3-Pad protocol, it has to deal with the dynamic peeling method used during the deconstruction phase. In this phase, the Map function is just the identity function. The Reduce function of the combination phase is presented in Figure 4.10.

- *The Reduce Function.* It is executed by nodes $\mathcal{C}_s$ with $s \in [\![1, \ell]\!]$ where $\ell := \lfloor \log_2(d) \rfloor$. For the same key $t \in \{0,7\}^\ell$, three different cases are possible depending on the associated values.

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

4.4 Secure Strassen-Winograd Matrix Multiplication with MapReduce

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

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

4.4.1 Preprocessing for Secure Strassen-Winograd Matrix Multiplication

In order to avoid the public cloud from learning the content of the two matrices and the result of their product, each data owner performs a preprocessing on its own matrix. This

Reduce function:

Input: $(key, values)$

// *key*: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 6]\!]$ for $z \in [\![0, e]\!]$

// *values*: collection of $(u, v, w, r_{v,w}, \delta)$ or $(\mathtt{Q}, i, j, q_{i,j}, \delta_{\mathtt{q}})$ such that $u \in [\![0, 6]\!]$, $i, j \in [\![1, \delta]\!]$

if $\exists\big((\mathtt{Q}, i, j, q_{i,j}, \delta_{\mathtt{q}}) \wedge (u, v, w, r_{v,w}, \delta)\big) \in values$ **then**

$Q := (q_{i,j})_{(\mathtt{Q},i,j,q_{i,j},\delta_{\mathtt{q}}) \in values}$

foreach $u \in [\![0, 6]\!]$ **do**

$R_u := (r_{v,w})_{(u,v,w,r_{v,w},\delta) \in values}$

$C_1 := R_0 + R_1 \qquad C_5 := C_4 + R_2$

$C_2 := R_0 + R_5 \qquad C_6 := C_3 - R_3$

$C_3 := C_2 + R_6 \qquad C_7 := C_3 - R_4$

$C_4 := C_2 + R_4$

$(c_{i,j})_{i,j \in [\![1, 2\cdot\delta]\!]} := \begin{bmatrix} C_1 & C_5 \\ C_6 & C_7 \end{bmatrix}$

$c'_{i,j} := \begin{cases} c_{i,j} + q_{i,j} & \text{if} \quad i, j \in [\![1, 2 \cdot \delta]\!] \\ q_{i,j} & \text{if} \quad \max_{i,j \in [\![1, 2\cdot\delta+1]\!]}(i, j) = 2 \cdot \delta + 1 \end{cases}$

$\delta' := 2 \cdot \delta + 1$

if $\delta' < d$ **then**

foreach $(i, j) \in [\![1, \delta']\!]^2$ **do**

$\mathtt{emit}_{\mathcal{C} \to \mathcal{C}}(t_0 \cdots t_{e-1}, (t_e, i, j, c'_{i,j}, \delta'))$

else

foreach $(i, j) \in [\![1, d]\!]^2$ **do**

$\mathtt{emit}_{\mathcal{C} \to \mathcal{C}}((i, j), c'_{i,j})$

else if $\big\{\exists(\mathtt{Q}, i, j, q_{i,j}, \delta_{\mathtt{q}}) \in values\big\} \wedge \big\{\nexists(u, v, w, r_{v,w}, \delta) \in values\big\}$ **then**

foreach $(\mathtt{Q}, i, j, q_{i,j}, \delta_{\mathtt{q}}) \in values$ **do**

$\mathtt{emit}(t_0 \cdots t_e, (\mathtt{Q}, i, j, q_{i,j}, \delta_{\mathtt{q}}))$

else if $\big\{\nexists(\mathtt{Q}, i, j, q_{i,j}, \delta_q) \in values\big\} \wedge \big\{\exists(u, v, w, r_{v,w}, \delta) \in values\big\}$ **then**

foreach $u \in [\![0, 6]\!]$ **do**

$R_u := (r_{v,w})_{(u,v,w,r_{v,w},\delta) \in values}$

$C_1 := R_0 + R_1 \qquad C_5 := C_4 + R_2$

$C_2 := R_0 + R_5 \qquad C_6 := C_3 - R_3$

$C_3 := C_2 + R_6 \qquad C_7 := C_3 - R_4$

$C_4 := C_2 + R_4$

$(c_{i,j})_{i,j \in [\![1, 2\cdot\delta]\!]} := \begin{bmatrix} C_1 & C_5 \\ C_6 & C_7 \end{bmatrix}$

$\delta' := 2 \cdot \delta$

if $\delta' < d$ **then**

foreach $(i, j) \in [\![1, \delta']\!]^2$ **do**

$\mathtt{emit}_{\mathcal{C} \to \mathcal{C}}(t_0 \cdots t_{e-1}, (t_e, i, j, c'_{i,j}, \delta'))$

else

foreach $(i, j) \in [\![1, d]\!]^2$ **do**

$\mathtt{emit}_{\mathcal{C} \to \mathcal{C}}((i, j), c'_{i,j})$

Figure 4.10: Reduce function for the combination phase of the SW-Peel protocol.

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

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

4.4.2 Secure Approach

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

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

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

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

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

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

Moreover, secure approaches use the Paillier interactive multiplicative homomorphic protocol denoted Paillier.Inter and presented in Figure 2.4 on page 24.

4.4.3 Secure Strassen-Winograd Matrix Multiplication Protocol

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

Deconstruction Phase

We present the deconstruction phase of S2M3. The Map function is the same than for SM3 protocol presented in Figure 4.3. The only difference is that it operates on encrypted matrices M^* and N^* sent by data owners after the preprocessing.

The Reduce function is presented in Figure 4.11. Since it operates on encrypted matrices, we use functions Paillier.Add and Paillier.Sub to add or subtract two matrices. Moreover, it uses the Paillier interactive multiplicative homomorphic protocol Paillier.Inter during the last round of the decomposition phase to the encryption of the multiplication of two elements.

Reduce function:

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

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

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

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

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

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

Figure 4.11: Reduce function for the deconstruction phase of the S2M3 protocol.

Combination Phase

As for SM3 protocol, the Map function of the combination phase is the identity function. The Reduce function of the combination phase is presented in Figure 4.12. The only difference compared to S2M3 protocol is the use of Paillier.Add and Paillier.Sub functions for addition and subtraction of encrypted matrices.

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

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

Deconstruction Phase

The Map function is the same than for SM3-Pad protocol presented in Figure 4.6. The only difference is that it operates on encrypted matrices M^* and N^* sent by data owners

Reduce function:

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

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

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

$$(p_{v,w})_{v,w \in [\![1,2\cdot\delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$$

if $\delta < d$ **then**
  $\delta' := 2 \cdot \delta$
  $\ell' := \log_2(\delta')$
  foreach $(v, w) \in [\![1, 2 \cdot \delta']\!]^2$ **do**
    $\texttt{emit}_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t_0 \cdots t_{e-1}, (t_e, i, j, p_{v,w}, 2 \cdot \delta'))$
else
  foreach $(v, w) \in [\![1, d]\!]^2$ **do**
    $\texttt{emit}_{\mathcal{D}_\ell \to \mathcal{P}}((v, w), p_{v,w})$

Figure 4.12: Reduce function for the combination phase of the S2M3 protocol.

after the preprocessing.

The Reduce function is presented in Figure 4.13. Note that it pads matrices with encryption of zero instead of zero as for SM3-Pad protocol. Since it operates on encrypted matrices, we use Paillier.Add and Paillier.Sub functions to add or subtract two encrypted matrices. Moreover, it uses the Paillier interactive multiplicative homomorphic protocol Paillier.Inter during the last round of the deconstruction phase to the encryption of the multiplication of two elements.

Combination Phase

As for SM3-Pad, the Map function of the combination phase is the identity function. The Reduce function is presented in Figure 4.14. It works as the Reduce function of the combination phase of SM3-Pad protocol.

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

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

Deconstruction Phase

The Map function is the same than for SM3-Peel protocol presented in Figure 4.3. The only difference is that it operates on encrypted matrices M^* and N^* sent by data owners after the preprocessing.

Reduce function:

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

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

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

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

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

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

Figure 4.13: Reduce function for the deconstruction phase of the S2M3-Pad protocol.

The Reduce function is presented in Figure 4.15. Since it operates on encrypted matrices, it uses Paillier.Add and Paillier.Sub functions to add or subtract two encrypted matrices. Moreover, it uses the Paillier interactive multiplicative homomorphic protocol during the last round of the decomposition phase since the multiplication is performed over encrypted values.

Reduce function:

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

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

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

$$(p_{k,l})_{k,l \in [\![1, 2 \cdot \delta]\!]} := \begin{bmatrix} P_1 & P_5 \\ P_6 & P_7 \end{bmatrix}$$

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

Figure 4.14: Reduce function for the combination phase of the S2M3-Pad protocol.

Combination Phase

The Map function of the combination phase for the S2M3-Peel protocol is the identity function. The Reduce function of S2M3-Peel protocol is presented in Figure 4.16. It works as the Reduce function of the SM3-Peel protocol.

4.5 Experimental Results

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

4.5.1 Dataset and Settings

For each experiment, we generate two random square matrices and of order d such that $240 \leq d \leq 450$ for no-secure protocols, and $90 \leq d \leq 300$ for secure protocols. Elements of both matrices are in $[\![0, 10]\!]$. For each order d, we perform matrix multiplication with SM3 and S2M3 protocols using static padding, dynamic padding, and dynamic peeling methods. We also compare the results to the standard matrix multiplication using one MapReduce round [LRU14] and the secure approach denoted CRSP-1R presented in Section 3.3.3 of the previous chapter.

For each experiment, we stop the Strassen-Winograd recursive matrix multiplication when the dimension of matrices is less than 16. Then, we use the MM-1R protocol for the

Reduce function:

Input: $(key, values)$
// *key*: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 7]\!]$ for $z \in [\![0, e]\!]$
// *values*: collection of $(\mathtt{M}, i, j, m^*_{i,j}, \delta)$ or $(\mathtt{N}, j, k, n^*_{j,k}, \delta)$ or $(\mathtt{P}, j, k, p^*_{a,z}, \delta_p)$

foreach $(\mathtt{P}, i, j, p^*_{a,z}, \delta_p) \in values$ **do**
  $\mathtt{emit}(t_0 \cdots t_e, (\mathtt{P}, i, j, p^*_{a,z}, \delta_p))$

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

if $\delta \not\equiv 0 \pmod 2$ **then**
  $\begin{bmatrix} M' & M_{12} \\ M_{21} & M_{22} \end{bmatrix} := M^*$, $\begin{bmatrix} N' & N_{12} \\ N_{21} & N_{22} \end{bmatrix} := N^*$,

  such that $\begin{cases} M' := (m_{ij})_{i,j \in [\![1,\delta-1]\!]} \\ M_{12} := (m_{ij})_{i \in [\![1,\delta-1]\!], j=\delta} \\ M_{21} := (m_{ij})_{i=\delta, j \in [\![1,\delta-1]\!]} \\ M_{22} := (m_{ij})_{i=\delta, j=\delta} \end{cases}$, and $\begin{cases} N' := (n_{ij})_{i,j \in [\![1,\delta-1]\!]} \\ N_{12} := (n_{ij})_{i \in [\![1,\delta-1]\!], j=\delta} \\ N_{21} := (n_{ij})_{i=\delta, j \in [\![1,\delta-1]\!]} \\ N_{22} := (n_{ij})_{i=\delta, j=\delta} \end{cases}$

  $(q_{i,j})_{i,j \in [\![1,\delta]\!]} := \begin{bmatrix} M_{12}N_{21} & M'N_{12} + M_{12}N_{22} \\ M_{21}N' + M_{22}N_{21} & M_{21}N_{22} + M_{22}N_{21} \end{bmatrix}$
  $\delta' := (\delta - 1)/2$
  foreach $(a, z) \in [\![1, \delta]\!]^2$ **do**
    $\mathtt{emit}(t_0 \cdots t_e, (\mathtt{P}, i, j, q^*_{a,z}, \delta'))$
else
  $M' := M^*$, $N' := N^*$
  $\delta' := \delta/2$
  `// Split` M' `and` N' `into four quadrants of equal dimension`
  $\begin{bmatrix} M'_{11} & M'_{12} \\ M'_{21} & M'_{22} \end{bmatrix} := M'$, $\begin{bmatrix} N'_{11} & N'_{12} \\ N'_{21} & N'_{22} \end{bmatrix} := N'$

  $S_1 := \mathsf{Paillier.Add}(pk, M'_{21}, M'_{22})$ $T_1 := \mathsf{Paillier.Sub}(pk, N'_{12}, N'_{11})$
  $S_2 := \mathsf{Paillier.Sub}(pk, S_1, M'_{11})$ $T_2 := \mathsf{Paillier.Sub}(pk, N'_{22}, T_1)$
  $S_3 := \mathsf{Paillier.Sub}(pk, M'_{11}, M'_{21})$ $T_3 := \mathsf{Paillier.Sub}(pk, N'_{22}, N'_{12})$
  $S_4 := \mathsf{Paillier.Sub}(pk, M'_{12}, S_2)$ $T_4 := \mathsf{Paillier.Sub}(pk, T_2, N'_{21})$

  $L := \big[[M'_{11}, N'_{11}], [M'_{12}, N'_{21}], [S_4, N'_{22}], [M'_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\big]$
  if $\delta > 2$ **then**
    $\ell' := \log_2(d/\delta')$
    foreach $u \in [\![0, 6]\!]$ **do**
      $(a'_{ij})_{i,j \in [\![1,\delta']\!]} := L[u][0]$
      $(b'_{jk})_{j,k \in [\![1,\delta']\!]} := L[u][1]$
      foreach $(v, w) \in [\![1, \delta']\!]^2$ **do**
        $\mathtt{emit}_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t \| u, (\mathtt{M}, v, w, a'_{ij}, \delta'))$
        $\mathtt{emit}_{\mathcal{D}_{\ell'} \to \mathcal{D}_{\ell'+1}}(t \| u, (\mathtt{N}, v, w, b'_{ij}, \delta'))$
  else
    foreach $u \in [\![0, 6]\!]$ **do**
      $\mathtt{emit}_{\mathcal{D}_\ell \to \mathcal{C}_1}(t, (u, 1, 1, \mathsf{Paillier.Sub}(L[u][0], L[u][1]), 1))$

Figure 4.15: Reduce function for the deconstruction phase of the S2M3-Peel protocol.

no-secure approach, and the CRSP-1R protocol for the secure approach. Both protocols are presented in the previous chapter.

Reduce function:

Input: $(key, values)$
// *key*: $t_0 \cdots t_e$ such that $e \in [\![0, \ell]\!]$ and $t_z \in [\![0, 7]\!]$ for $z \in [\![0, e]\!]$
// *values*: collection of $(u, v, w, r^*_{v,w}, \delta)$ or $(\mathtt{P}, a, z, p^*_{a,z}, \delta_p)$ such that $u \in [\![1, 7]\!]$

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

Figure 4.16: Reduce function for the combination phase of the S2M3-Peel protocol.

Our secure protocols are based on the Paillier's cryptosystem. We use Gaillier[†], a Go implementation of the Paillier's cryptosystem. Note that Gaillier is not an optimized implementation. Hence, we use it with a 64-bit RSA modulus as proof of concept.

[†] https://github.com/actuallyachraf/gomorph

4.5.2 Results

In Figure 4.17, we present CPU times for the no-secure SM3 protocols using the static padding method denoted SM3-sPad, the dynamic padding method denoted SM3-Pad, or the dynamic peeling method denoted SM3-Peel. Moreover, we compare them to the standard matrix multiplication using one MapReduce round [LRU14] presented in the previous chapter and denoted CRSP-1R.

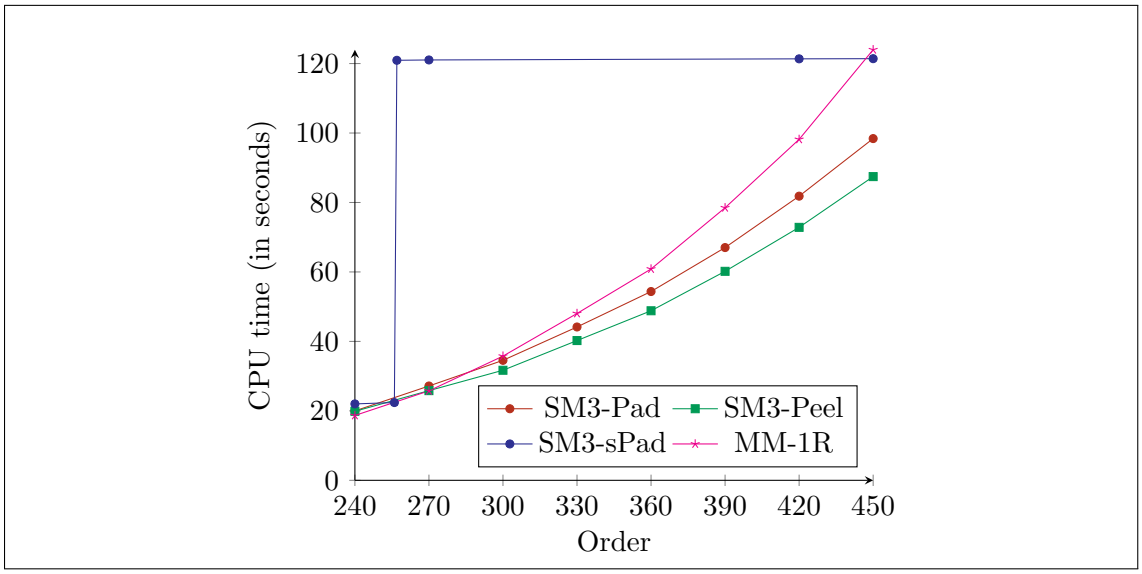


Figure 4.17: CPU time vs order of matrices for the state-of-the-art MM-1R protocol using one MapReduce round [LRU14] and for our SM3 protocols using static and dynamic padding methods, and dynamic peeling method.

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

We present in Figure 4.18, CPU times for secure protocols computing the Strassen-Winograd matrix multiplication. We compare them to our secure standard matrix multiplication protocol CRSP-1R using one MapReduce round and presented in Section 3.3.3 of the previous chapter.

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

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

4.6 Security Proofs

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

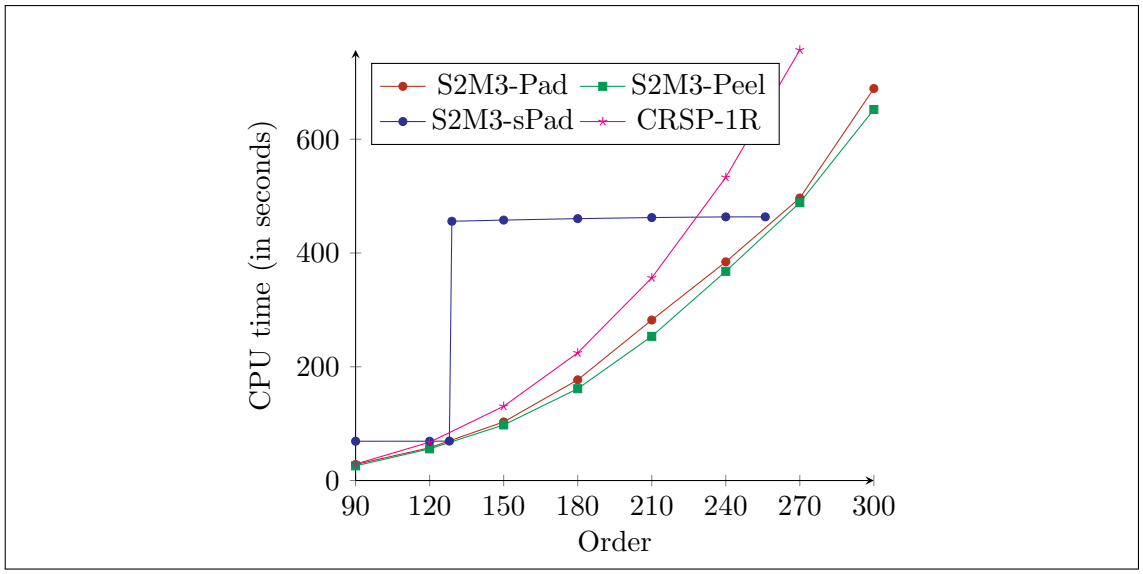


Figure 4.18: CPU time vs order of matrices for the CRSP-1R protocol presented in Chapter 3 and for our S2M3 protocols using static and dynamic padding methods, and dynamic peeling method.

saries [Lin17].

4.6.1 Security Proof for S2M3 Protocol

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

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

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

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

The security of S2M3 protocol is given in Theorem 5.

Theorem 5. *Assume Paillier's cryptosystem is IND-CPA, then* S2M3 *securely computes the matrix multiplication in the presence of semi-honest adversaries even if public cloud's nodes collude.*

The security proof for S2M3 protocol (Theorem 5) is decomposed in Lemma 16 for parties P_M and P_N, Lemma 17 for party $P_{\mathcal{C}}$, and Lemma 18 for party $P_{\mathcal{P}}$.

Lemma 16. *There exists probabilistic polynomial-time simulators $\mathcal{S}_M^{\text{S2M3}}$ and $\mathcal{S}_N^{\text{S2M3}}$ such that:*

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

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

Lemma 17. *Assume Paillier's cryptosystem is IND-CPA, then there exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}$ such that:*

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

Function: SW-Deconstruction
SW-Deconstruction$(A, B, U, \mathsf{view}, pk)$:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} := A\,, \quad \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} := B$$

$S_1 := \mathsf{Paillier.Add}(pk, A_{21}, A_{22})$ $\quad$ $T_1 := \mathsf{Paillier.Sub}(pk, B_{12}, B_{11})$
$S_2 := \mathsf{Paillier.Sub}(pk, S_1, A_{11})$ $\quad$ $T_2 := \mathsf{Paillier.Sub}(pk, B_{22}, T_1)$
$S_3 := \mathsf{Paillier.Sub}(pk, A_{11}, A_{21})$ $\quad$ $T_3 := \mathsf{Paillier.Sub}(pk, B_{22}, B_{12})$
$S_4 := \mathsf{Paillier.Sub}(pk, A_{12}, S_2)$ $\quad$ $T_4 := \mathsf{Paillier.Sub}(pk, T_2, B_{21})$

$L := \left[[A_{11}, B_{11}], [A_{12}, B_{21}], [S_4, B_{22}], [A_{22}, T_4], [S_1, T_1], [S_2, T_2], [S_3, T_3]\right]$

foreach $[A_0, B_0] \in L$ **do**
  if $dim(A_0) = dim(B_0) = 1$ **then**
    $U := U \cup \{[A_0, B_0]\}$
  else
    $\mathsf{view} := \mathsf{view} \cup \{[A_0, B_0]\}$
    SW-Deconstruction$(A_0, B_0, U, \mathsf{view}, pk)$

Figure 4.19: Function SW-Deconstruction for simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}$ presented in Figure 4.21.

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

Simulator of $P_{\mathcal{C}}$ is given in Figure 4.21. Function SW-Deconstruction presented in Figure 4.19 simulates the public cloud's view during the deconstruction phase. The view contains all submatrices corresponding the recursive matrices multiplications. Moreover, the last set of nodes of the deconstruction phase $\mathcal{D}_\ell$ sends couples of ciphertexts (x_i, y_i), with $i \in [\![1, 7^\ell]\!]$, to $P_{\mathcal{P}}$ and receives all corresponding ciphertexts $R_i^{(\ell)}$ returned by $P_{\mathcal{P}}$ to compute multiplication on encrypted coefficients.

Function SW-Combination presented in Figure 4.20 simulates the public cloud's view during one round of the combination phase. For each round, it combines submatrices according the SW algorithm.

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

Figure 4.20: Function SW-Combination for simulator $\mathcal{S}_{\mathcal{C}}^{\mathrm{S2M3}}$ presented in Figure 4.21.

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

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

Figure 4.21: Simulator $\mathcal{S}_{\mathcal{C}}^{\mathrm{S2M3}}$ for the proof of Lemma 17.

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

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

where μ is a non-negligible function in λ. We show how to build a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}$ has a non-negligible advantage to win the IND-CPA experiment on the Paillier's cryptosystem. Then we conclude the proof by contraposition. Adversary $\mathcal{A}$ is presented in Figure 4.22. At the end of its execution, $\mathcal{A}$ uses the distinguisher D to compute the bit b^* before returning it. First, we remark that:

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

Indeed, when $b = 0$, the view that $\mathcal{A}$ uses as input for D is computed as in real S2M3 protocol. Then the probability that the experiment returns 1 is equal to the probability

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

$U := \emptyset$
$\mathsf{view} := \emptyset$
foreach $i \in [\![1,d]\!]$ **do**
  foreach $j \in [\![1,d]\!]$ **do**
    $(m_{i,j}, n_{i,j}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
    $(\alpha_{i,j}, \beta_{i,j}) \xleftarrow{\$} (\mathbb{Z}_n)^2$
$M^* := \big(\mathcal{E}(pk,\mathsf{LoR}_b(m_{i,j}, \alpha_{i,j}))\big)_{i,j\in[\![1,d]\!]}$
$N^* := \big(\mathcal{E}(pk,\mathsf{LoR}_b(n_{i,j}, \beta_{i,j}))\big)_{i,j\in[\![1,d]\!]}$
$\mathsf{SW\text{-}Deconstruction}(M^*, N^*, U, \mathsf{view}, pk)$
foreach $i \in [\![1,7^\ell]\!]$ **do**
  $(r_i, s_i, t_i) \xleftarrow{\$} (\mathbb{Z}_n)^3$
  $x_i := \mathcal{E}(pk, r_i)$
  $y_i := \mathcal{E}(pk, s_i)$
  $R_i^{(\ell)} := \mathcal{E}(pk,\mathsf{LoR}_b(U[i-1][0] \cdot U[i-1][1], t_i))$
  $\mathsf{view} := \mathsf{view} \cup \{R_i^{(\ell)}\}$
foreach $k \in [\![\ell,1]\!]$ **do**
  foreach $j \in [\![1,7^{k-1}]\!]$ **do**
    $(R_j^{(k-1)}, \mathsf{view}') := \mathsf{SW\text{-}Combination}(R_{7\cdot j-6}^{(k)}, \ldots, R_{7\cdot j}^{(k)})$
    $\mathsf{view} := \mathsf{view} \cup \mathsf{view}'$
return view

Figure 4.22: Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 17.

that the distinguisher returns 1 on inputs computed as in real protocol. On the other hand, we have:

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

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

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

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

Lemma 18. *There exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}$ such that:*

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

Proof. Simulator $\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}$ is presented in Figure 4.23. The view of $P_{\mathcal{P}}$ contains the couple of ciphertexts (x_i, y_i) sent by the set of nodes $\mathcal{D}_\ell$ during the deconstruction phase run by $P_{\mathcal{C}}$ and the answer z_i sent by $P_{\mathcal{P}}$ to $P_{\mathcal{C}}$ that contains the encryption of the multiplication of x_i and y_i, for $i \in [\![1,7^\ell]\!]$. Since x_i and y_i are randomized by $P_{\mathcal{C}}$, there are indistinguishable

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

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

Figure 4.23: Simulator $\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}$ for the proof of Lemma 18.

to random ciphertexts in the $P_{\mathcal{P}}$ point of view. The view of $P_{\mathcal{P}}$ also contains $P^* := (\mathcal{E}(pk, p_{i,j}))_{i,j \in [\![1,d]\!]}$ that is sent by $P_{\mathcal{C}}$. Finally, $\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}(1^\lambda, (pk, sk), P)$ describes exactly the same distribution as $\mathsf{view}_{\mathcal{P}}^{\text{S2M3}}(I, \lambda)$, which concludes the proof. □

4.6.2 Security Proof for S2M3-Pad Protocol

S2M3-Pad protocol is modeled as the S2M3 protocol by parties P_M, P_N, $P_{\mathcal{C}}$, and $P_{\mathcal{P}}$. The security of S2M3-Pad protocol is given in Theorem 6.

Theorem 6. *Assume Paillier's cryptosystem is IND-CPA, then* S2M3-Pad *securely computes the matrix multiplication in the presence of semi-honest adversaries even if public cloud's nodes collude.*

The security proof for S2M3-Pad protocol (Theorem 6) is decomposed in Lemma 19 for parties P_M and P_N, Lemma 20 for party $P_{\mathcal{C}}$, and Lemma 21 for party $P_{\mathcal{P}}$.

Lemma 19. *There exists two probabilistic polynomial-time simulators* $\mathcal{S}_M^{\text{S2M3-Pad}}$ *and* $\mathcal{S}_N^{\text{S2M3-Pad}}$ *such that:*

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

Proof. Since the view of P_M and P_N are exactly the same than for the S2M3 protocol, the proof is the same than Lemma 16. □

Lemma 20. *Assume Paillier's cryptosystem is IND-CPA, then there exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{C}}^{\text{S2M3-Pad}}$ *such that:*

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

Proof. The only difference with the S2M3 protocol is the adding of the padding (with encryption of zeros) during the deconstruction phase, and the removing of the padding during the combination phase. In the security point of view, it is equivalent to deal with padded encrypted matrices or not. Hence, the proof is the same than Lemma 17. □

Lemma 21. *There exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{P}}^{\text{S2M3-Pad}}$ *such that:*

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

Proof. The proof is exactly the same than Lemma 18. □

4.6.3 Security Proof for S2M3-Peel Protocol

S2M3-Peel protocol is modeled as the S2M3 and S2M3-Pad protocols by parties P_M, P_N, $P_{\mathcal{C}}$, and $P_{\mathcal{P}}$. The security of S2M3-Peel protocol is given in Theorem 7.

Theorem 7. *Assume Paillier's cryptosystem is IND-CPA, then* S2M3-Peel *securely computes the matrix multiplication in the presence of semi-honest adversaries even if public cloud's nodes collude.*

The security proof for S2M3-Peel protocol (Theorem 7) is decomposed in Lemma 22 for parties P_M and P_N, Lemma 23 for party $P_{\mathcal{C}}$, and Lemma 24 for party $P_{\mathcal{P}}$.

Lemma 22. *There exists two probabilistic polynomial-time simulators* $\mathcal{S}_M^{\text{S2M3-Peel}}$ *and* $\mathcal{S}_N^{\text{S2M3-Peel}}$ *such that:*

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

Proof. Since the view of P_M and P_N are exactly the same than for the S2M3 protocol, the proof is the same than Lemma 16. □

Lemma 23. *Assume Paillier's cryptosystem is IND-CPA, then there exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{C}}^{\text{S2M3-Peel}}$ *such that:*

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

Proof. The only difference with the S2M3 protocol is that encrypted matrices are splitted according to the dynamic peeling during the deconstruction phase when it is required. Moreover, blocks multiplications are performed during the deconstruction phase and used during the combination phase to build the corresponding matrix. In the security point of view, these blocks multiplications do not give any information to the adversary. Hence we can consider only the blocks multiplication corresponding to the standard Strassen-Winograd matrix multiplication. Therefore, the security proof is the same than Lemma 17. □

Lemma 24. *There exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{P}}^{\text{S2M3-Peel}}$ *such that:*

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

Proof. The proof is exactly the same than Lemma 18. □

4.7 Conclusion

We have presented SM3, a protocol to compute the Strassen-Winograd matrix multiplication using the MapReduce paradigm. Moreover, we extend this protocol to SM3-Pad and SM3-Peel protocols using respectively the dynamic padding and the dynamic peeling methods allowing square matrix multiplication of arbitrary dimension. We have also presented a secure approach for these three protocols denoted S2M3, S2M3-Pad, and S2M3-Peel satisfying privacy guarantees such that the public cloud does not learn any information on input matrices and on the output matrix. To achieve our goal, we have relied on the well-known Paillier's cryptosystem and on its homomorphic properties. We have compared our three no-secure protocols (resp. secure protocols) with the state-of-the-art MapReduce protocol of Leskovec et al. [LRU14] (with the CRSP protocol proposed by

Bultel et al. [BCGL17]) computing matrix multiplication, and shown that our protocols are more efficient.

Looking forward to future work, we aim to investigate the matrix multiplication with privacy guarantees in different big data systems (e.g., Spark, Flink) whose users also tend to outsource data and computations as MapReduce.

Part II

Relational-Algebra Operations

CHAPTER 5

Secure Intersection with MapReduce

In this chapter, we present our secure multiparty protocol called SI, for *Secure Intersection*, computing the intersection of $n \geq 2$ relations using the MapReduce paradigm. Our protocol relies on classic cryptographic primitives such as pseudo-random function, and asymmetric encryption scheme. The protocol allows an external user to query the intersection of $n \geq 2$ relations owned by n different data owners. At the end of the protocol the user only learns the intersection of the n relations even if she colludes with the public cloud. This work has been conducted in collaboration with Radu Ciucanu, Pascal Lafourcade, and Lihua Ye and has been published in the paper "*Secure Intersection with MapReduce*" [CGLY19a] at the 16$^{\text{th}}$ *International Conference on Security and Cryptography* (SECRYPT, 2019).

Contents

5.1 Introduction 88
5.1.1 Intersection with MapReduce 88
5.1.2 Problem statement 89
5.1.3 Contributions 90
5.1.4 Outline 90
5.2 Standard and Secure Intersection with MapReduce 90
5.2.1 Standard Intersection with MapReduce 91
5.2.2 Secure Intersection with MapReduce 91
5.2.3 Proof of Correctness 93
5.2.4 Complexity of Original and Secure Protocols 95
5.3 Experimental Results 95
5.3.1 Settings 95
5.3.2 Varying the Number of Tuples 96
5.3.3 Varying the Number of Intersected Relations 96
5.4 Security Proof 97
5.4.1 Modeling of our SI Protocol 97
5.4.2 Proof 98
5.5 Conclusion 103

5.1 Introduction

We consider the problem of intersection of an arbitrary number of relations, each of them belonging to a different data owner. We rely on the popular MapReduce [DG04] paradigm for outsourcing data and computations to a semi-honest public cloud. Our goal is to compute the intersection of these relations while preserving the data privacy of the data owners. We design a protocol based on asymmetric cryptography where each data owner performs some computation on their respective relation. The resulted relations, called *encrypted* relations are sent to the public cloud. The public cloud cannot learn neither the input nor the output data, it learns only the cardinal of each relation, and the cardinal of the intersection of these relations. At the end of the computation, the public cloud sends the result to the final user who just has to decrypt the received data using her secret key. Moreover, if the cloud and the user collude, i.e., the cloud knows the user's secret key, then they cannot learn other information than the relation intersection result still known by the user.

5.1.1 Intersection with MapReduce

A protocol to compute the intersection between two relations with MapReduce is presented in Chapter 2 of [LRU14]. We stress that intersection between relations can be viewed as intersection between sets where elements of these sets correspond to the tuples of relations having the same schema. In Chapter 2 of [LRU14], the public cloud receives two relations from their respective owner. A collection of cloud nodes has chunks of these two relations. The *Map* function creates for each tuple t a *key-value* pair (t, t) where *key* and *value* are equal to the tuple. Then, the key-value pairs are grouped by key, i.e., key-value pairs output by the map phase which have the same key are sent to the same reducer. For each key, the *Reduce* function checks if the considered key is associated to two values. If it is the case, i.e., tuple t is present in both relations, then the public cloud produces and sends the pair $(-, t)$ to the user. The dash value "$-$" corresponds to the empty value, we use it to be consistent with the MapReduce paradigm. If it is not useful in this chapter, we need it in Chapters 3 and 4 since we run several rounds of MapReduce. Hence, all tuples received by the user correspond to the tuples that are in both relations. We keep the key-value pair notation in order to be consistence with the MapReduce paradigm. However, the key is irrelevant at the end of the protocol, so we omit to write it. We illustrate this approach with the following example considering three relations.

Example. We consider three relations: NSA, GCHQ, and Mossad. Each relation is owned by their respective data owner. These three relations have the same schema composed of only one attribute, namely "Suspect's Identity". They are defined in Figure 5.1.

Relation NSA

Suspect's Identity
F654
U840
X098

Relation GCHQ

Suspect's Identity
F654
M349
P027

Relation Mossad

Suspect's Identity
F654
M349
U840

Figure 5.1: Relations NSA, GCHQ, and Mossad.

An external user, called *Interpol*, wants to receive the intersection of these three relations denoted Interpol. We illustrate the execution of intersection computation with MapReduce for this setting in Figure 5.8. First, each data owner outsources their respective relation into the public cloud. Then, the public cloud runs the map function on each

relation and sends the output to the master controller in order to sort key-value pairs by key. Then, the master controller sends key-value pairs sharing the same key to the same reducer. In our example, we obtain 5 reducers since there are 5 different suspect's identities. The reducer associated to the key `F654` has three values since the identity `F654` is present in the three relations NSA, GCHQ, and Mossad. The reducer associated to the key M349 has two values since the identity M349 is only present in relations GCHQ and Mossad. Other reducers are associated to only one value since the corresponding suspect's identity is present in only one relation. For each reducer, the public cloud runs the reduce function and sends the tuple $(-, \texttt{ID})$ to the user if the suspect's identity `ID` is present is the three relations, else the public cloud sends nothing. In our example, we observe that the user *Interpol* only receives the pair $(-, \texttt{F654})$ since the suspect's identity `F654` is present in the three relations NSA, GCHQ, and Mossad.

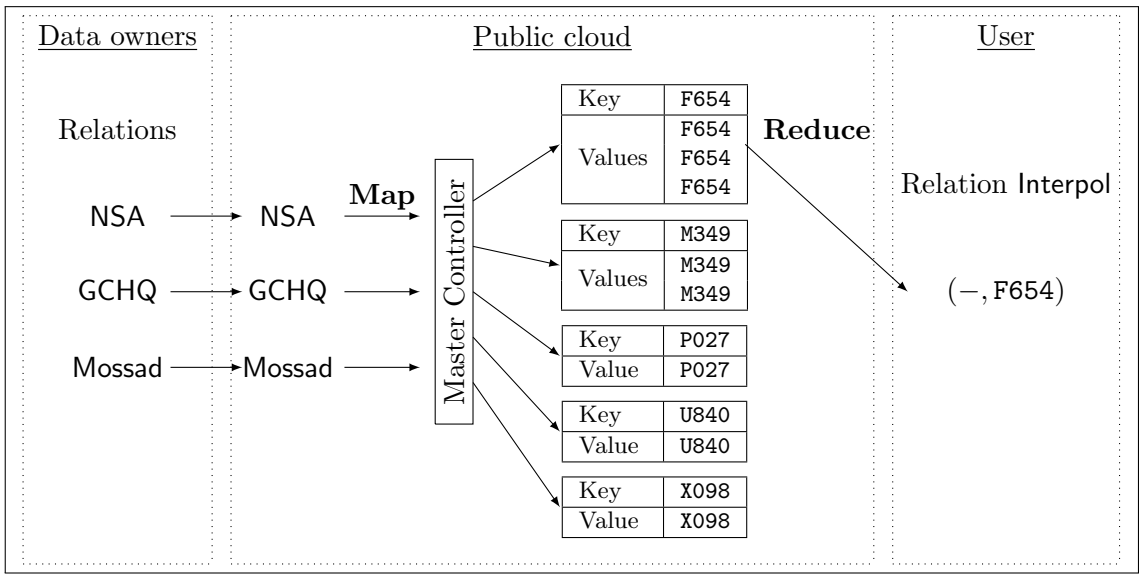


Figure 5.2: Example of intersection with MapReduce between three relations. First, data owners outsource their respective relation on the public cloud. The public cloud runs the Map function, then the Reduce function verifies if a key is associated to a list of three values. If that is the case, the public cloud produces and sends tuples corresponding to the intersection to the user.

5.1.2 Problem statement

We assume $n + 2$ parties: n data owners, a public cloud, and a user that we call the MapReduce's user. This user is authorized to query the intersection of the relations of the data owners.

First, each data owner outsources its respective relation R_i, with $i \in [\![1, n]\!]$, to the distributed file system of some public cloud provider. We assume that each relation R_i is initially spread over a set $\mathcal{R}_i$ of nodes of the public cloud, each of them storing a chunk of R_i, i.e., a set of tuples of R_i. Then, the result of the intersection $\cap_{i=1}^{n} R_i$ is computed over a set $\mathcal{C}$ of nodes before it is sent to the user's nodes $\mathcal{U}$. We illustrate the architecture in Figure 5.3.

We assume that data owners are trustworthy while the cloud service provider is semi-honest [Lin17], i.e., it dutifully executes the protocol but tries to deduce as much as possible information on relations and on the result of the intersection. Moreover, we assume that the public cloud can collude with the user, i.e., they share all their respective private

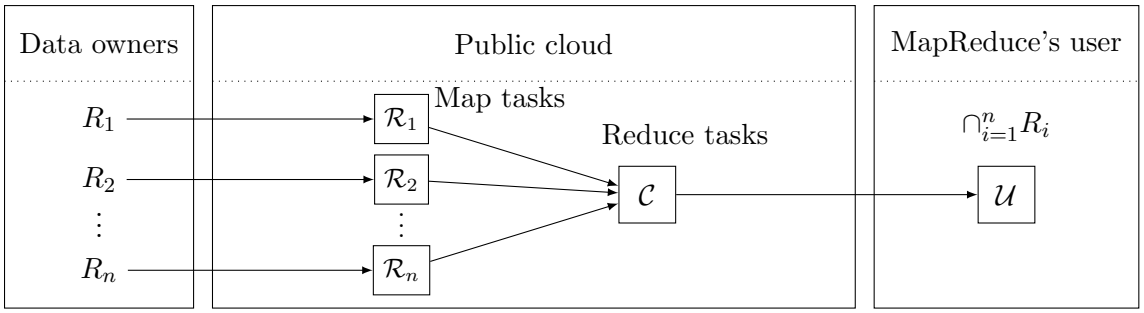


Figure 5.3: The system architecture.

information. We want that the user that queried the intersection of these n relations along the public cloud learn nothing else than the intersection of the n relations.

In the original protocol [LRU14], tuples of each relation are not encrypted, hence the public cloud learns all the content of each relation and the result of the intersection that it sends to the user as illustrated in Figure 5.8. In order to preserve the privacy of data owners, the cloud should not learn any plain input data, contrary to what happens for the original protocol.

5.1.3 Contributions

We revisit the standard protocol for the computation of intersection with MapReduce (cf. Chapter 2 from [LRU14]) and propose a new protocol called SI (for Secure Intersection with MapReduce) that satisfies our aforementioned problem statement. More precisely:

- Our protocol SI guarantees that the user who queries the intersection of the n relations learns only the final result, i.e., tuples that are present in the n relations. Moreover, the public cloud does not learn information about the input data that belongs to the data owners, it learns only the cardinal of each relation and the cardinal of the intersection. SI also satisfies the problem setting in the presence of collusion between the user and the public cloud.
- To show the practical scalability of SI, we present experimental results using the MapReduce open-source implementation Apache Hadoop® 3.2.0 [Fou19b].
- We give a security proof of our secure protocol in the random oracle model.

5.1.4 Outline

We first recall the standard protocol [LRU14] and present our secure protocol SI in Section 5.2. In Section 5.3, we show experimental evaluations of SI considering intersection between two relations on different number of tuples, and considering intersection between different number of relations. Before concluding this chapter, we prove in Section 5.4 the security properties of SI in the random oracle model.

5.2 Standard and Secure Intersection with MapReduce

We consider $n \geq 2$ data owners, each of them owning a relation. These n relations have the same schema and are denoted $R_1, \ldots, R_n$. We first recall in Section 5.2.1 the standard MapReduce protocol to perform the intersection of n relations, i.e., a simple generalization of the binary protocol presented in Chapter 2 of [LRU14]. This protocol obviously does not satisfy privacy properties of our problem setting since the public cloud learns all

tuples of each relation sent by the respective data owner, and the intersection of these n relations sent to the user. Then we present in Section 5.2.2 our secure protocol denoted SI computing the intersection of n relations using the MapReduce paradigm. We prove that contrary to the standard protocol our protocol SI guarantees that the public cloud learns only cardinals of relations R_i for $i \in [\![1, n]\!]$. Moreover, if the public cloud and the user collude, then they learn the intersection of these n relations that the user already knows, and cardinals of relations R_i for $i \in [\![1, n]\!]$ that the public cloud already knows.

5.2.1 Standard Intersection with MapReduce

In the protocol given by Leskovec et al. [LRU14], the Map function creates for each tuple t of each relation R_i, with $i \in [\![1, n]\!]$, a key-value pair where the key and the value are equal to the tuple t. For a key t, the associated reducer receives a list of tuples a collection of tuples t. Hence, if a tuple t is only present in one relation, the reducer receives a collection only composed of one tuple t. On the contrary, if a tuple t' is present in all the n relations, the reducer receives a collection of n tuples equal to t'. If a key t is associated to a collection of n tuples t, then the Reduce function produces the key-value pair $(-, t)$ and sends it to the user. Otherwise, it produces nothing. All key-value pairs outputted by the Reduce function constitute the result of the intersection of the n relations. We present the protocol of Leskovec et al. [LRU14] computing the intersection protocol with MapReduce in Figure 5.4.

Map function:
// *key*: id of a chunk of R_i
// *value*: collection of tuples $t \in R_i$
foreach $t \in R_i$ **do**
  | $\texttt{emit}_{\mathcal{R}_i \to \mathcal{C}}(t, t)$

Reduce function:
// *key*: tuple $t \in \cup_{i=1}^{n} R_i$
// *values*: collection of tuples t
if $|values| = n$ **then**
  | $\texttt{emit}_{\mathcal{C} \to \mathcal{U}}(-, t)$

Figure 5.4: MapReduce protocol to compute the intersection of n relations.

We now consider a *semi-honest* public cloud performing the intersection of n relations with MapReduce. In such a scenario, the public cloud learns all the content of each relation along with the intersection of these n relations.

5.2.2 Secure Intersection with MapReduce

In order to perform intersection computation with MapReduce in a privacy-preserving way between $n \geq 2$ relations, our protocol needs a pseudo-random function and an asymmetric encryption scheme. The pseudo-random function is denoted F and defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, while the asymmetric encryption scheme is denoted $\Pi := (\mathcal{G}, \mathcal{E}, \mathcal{D})$. We assume that F is a secure pseudo-random function as defined in Definition 13 on page 19 and Π is an IND-CPA asymmetric encryption scheme as defined in Definition 19 on page 22. Furthermore, we assume that the length of the output pseudo-random values and the length of ciphertext output by the asymmetric encryption scheme match.

We start by presenting the preprocessing algorithm of our secure MapReduce protocol named SI for *Secure Intersection*.

Preprocessing for SI Protocol

Before outsourcing their relation to the public cloud, data owners perform a key setup and a preprocessing on their respective relation R_i to obtain an *encrypted* relation denoted R_i^*, with $i \in [\![1, n]\!]$.

First, we need a secret key $k_1 \in \mathcal{K}$ that is shared between all the n data owners. Moreover we need $n-1$ other secret keys $k_i \in \mathcal{K}$ with $i \in [\![2, n]\!]$ such that $k_i \neq k_j$ for $i \neq j$. Key k_i with $i \geq 2$ is shared between the owner of relation R_1 and the owner of relation R_i. Hence, the owner of relation R_1 has a set of n secret keys equal to $\{k_1, k_2, \ldots, k_n\}$ while the owner of relation R_i, for $2 \leq i \leq n$, has a set of secret keys equals to $\{k_1, k_i\}$. We stress that the choice of owner of relation R_1 knowing all the secret keys is arbitrary, and we call the associated relation, i.e., R_1, the *main* relation.

We now present the preprocessing algorithm in Figure 5.5.

Preprocessing:

Input: R_i with $i \in [\![1, n]\!]$
$R_i^* := \emptyset$
if $i = 1$ **then**
  foreach $t \in R_1$ **do**
    $M := \oplus_{j=2}^{n} F(k_j, t)$
    $R_1^* := R_1^* \cup \{(F(k_1, t), \mathcal{E}(pk, t) \oplus M)\}$
else
  foreach $t \in R_i$ **do**
    $R_i^* := R_i^* \cup \{(F(k_1, t), F(k_i, t))\}$
return R_i^*

Figure 5.5: Preprocessing algorithm of SI protocol.

The aim of this preprocessing is to protect relations in order to avoid the public cloud to learn tuples of each relation and the result of the intersection sent to the MapReduce's user. Moreover, this preprocessing is in agreement with the MapReduce paradigm. Indeed, each encrypted relation R_i^* is composed of tuples under the key-value pair form.

First of all, each key of pairs of R_i^* is a pseudo-random evaluation of a tuple using the secret key k_1 known by each data owner. Since a pseudo-random function is deterministic, equal tuples share the same value of key. Hence, the map phase sends these key-value pairs to the same reducer as expected.

Moreover, each value of key-value pairs of the encrypted relation R_1^* is equal to the encryption of the tuple using the asymmetric encryption scheme Π with the user public key pk xored by $n-1$ pseudo-random evaluations of the tuple using secret keys $k_2, \ldots, k_n$. More precisely, for each tuple $t \in R_1$, the preprocessing computes the key-value pair equals to $(F(k_1, t), \mathcal{E}(pk, t) \oplus (\oplus_{j=2}^{n} F(k_j, t)))$. Hence, when the public cloud receives such key-value pairs and colludes with the user, it cannot learn the value of tuples since the asymmetric encryption is protected by pseudo-random evaluations, and secret keys $k_1, \ldots, k_n$ are not known by the public cloud.

The complexity of the preprocessing algorithm of our secure approach is $O(N \cdot \mathcal{C}_{\mathcal{E}} + N \cdot (3 \cdot n - 2) \cdot \mathcal{C}_F)$, where n is the number of relations, $N := \max_{i \in [\![1,n]\!]} |R_i|$, $\mathcal{C}_{\mathcal{E}}$ is the cost of encryption, and $\mathcal{C}_F$ is the cost of a pseudo-random evaluation.

Map and Reduce Phases of SI Protocol

The preprocessing presented in Figure 5.5 outputs an encrypted relation whose tuples are of the key-value pair form. Hence, once the public cloud receives the n encrypted relations

R_i^* (for $i \in [\![1, n]\!]$) from the data owners, it runs the Map function that is simply the identity function.

After the grouping by key, the Reduce function checks if the current key $F(k_1, t)$, for $t \in \cup_{i=1}^{N} R_i$, is associated to a list of n values. If that is the case, it means that the n relations contain the tuple associated to the current key. Then the Reduce function uses these n values to perform an exclusive or, and obtains the asymmetric encryption of the tuple $\mathcal{E}(pk, t)$ due the property of the exclusive or.

Map function:
// *key*: id of a chunk of R_i^* with $i \in [\![1, n]\!]$
// *values*: collection of $(F(k_1, t), \mathcal{E}(pk, t) \oplus (\oplus_{j=2}^{n} F(k_j, t)))$ or $(F(k_1, t), F(k_j, t))$
// with $j \in [\![2, n]\!]$
foreach $(k, v) \in$ *values* **do**
  $\texttt{emit}_{\mathcal{R}_i \to \mathcal{C}}(k, v)$

Reduce function:
// *key*: $F(k_1, t)$ such that $t \in \cup_{i=1}^{n} R_i$
// *values*: collection of $\mathcal{E}(pk, t) \oplus (\oplus_{j=2}^{n} F(k_j, t))$ or $F(k_j, t)$ with $j \in [\![2, n]\!]$
$L := \{v : v \in values\}$
if $|L| = n$ **then**
  $\mathcal{E}(pk, t) := \mathcal{E}(pk, t) \oplus (\oplus_{j=2}^{n} F(k_j, t)) \oplus (\oplus_{j=2}^{n} F(k_j, t))$
  $\texttt{emit}_{\mathcal{C} \to \mathcal{U}}(-, \mathcal{E}(pk, t))$

Figure 5.6: Map and Reduce functions of our secure approach SI.

Finally, the Reduce function produces the key-value pair $(-, \mathcal{E}(pk, t))$ and sends it to the user. The output of the Reduce function is in a key-value form to be consistent with the MapReduce paradigm since at the end of the SI protocol keys are irrelevant. All key-value pairs outputted by the Reduce function constitute the intersection of the n relations. The user has only to decrypt each value of key-value pair using her secret key in order to obtain the intersection in plain form. Our protocol SI is described in Figure 5.6.

Example. We illustrate our SI protocol following the example presented in Section 5.1. First, we perform the preprocessing on relations: NSA, GCHQ, and Mossad. We consider relation NSA as the main relation. The three data owners share the secret key k_1, data owners of relations NSA and GCHQ share a secret key k_2, and data owners of relations NSA and Mossad share a secret key k_3. Hence, after the preprocessing phase, we obtain three encrypted relations denoted NSA*, GCHQ*, and Mossad* as illustrated in Figure 5.7.

We now illustrate the execution of intersection computation with MapReduce using our secure protocol SI in Figure 5.7.

Before giving the experimental results of the original protocol [LRU14] and of our secure protocol SI, we give a proof of correctness of SI.

5.2.3 Proof of Correctness

We say that the protocol SI is *correct* if for $n \geq 2$ relations $R_1, R_2, \ldots R_n$, SI returns the correct intersection of the $n \geq 2$ relations, i.e., the encrypted relation composed of pairs $(-, \mathcal{E}(pk, t))$ such that $t \in R$, where $R := \cap_{i=1}^{n} R_i$.

Lemma 25. *Assume that the pseudo-random function family F perfectly emulates a random oracle, then protocol SI is correct.*

Relation NSA^* (Owner knows secret keys k_1, k_2, and k_3.)

Suspect's Identity
$(F(k_1, \texttt{F654}), \mathcal{E}(pk, \texttt{F654}) \oplus F(k_2, \texttt{F654}) \oplus F(k_3, \texttt{F654}))$
$(F(k_1, \texttt{U840}), \mathcal{E}(pk, \texttt{U840}) \oplus F(k_2, \texttt{U840}) \oplus F(k_3, \texttt{U840}))$
$(F(k_1, \texttt{X098}), \mathcal{E}(pk, \texttt{X098}) \oplus F(k_2, \texttt{X098}) \oplus F(k_3, \texttt{X098}))$

Relation GCHQ^* (Owner knows secret keys k_1 and k_2.)

Suspect's Identity
$(F(k_1, \texttt{F654}), F(k_2, \texttt{F654}))$
$(F(k_1, \texttt{M349}), F(k_2, \texttt{M349}))$
$(F(k_1, \texttt{P027}), F(k_2, \texttt{P027}))$

Relation Mossad^* (Owner knows secret keys k_1 and k_3.)

Suspect's Identity
$(F(k_1, \texttt{F654}), F(k_3, \texttt{F654}))$
$(F(k_1, \texttt{M349}), F(k_3, \texttt{M349}))$
$(F(k_1, \texttt{U840}), F(k_3, \texttt{U840}))$

Figure 5.7: Encrypted relations NSA^*, GCHQ^*, and Mossad^* after the preprocessing phase of our secure protocol SI.

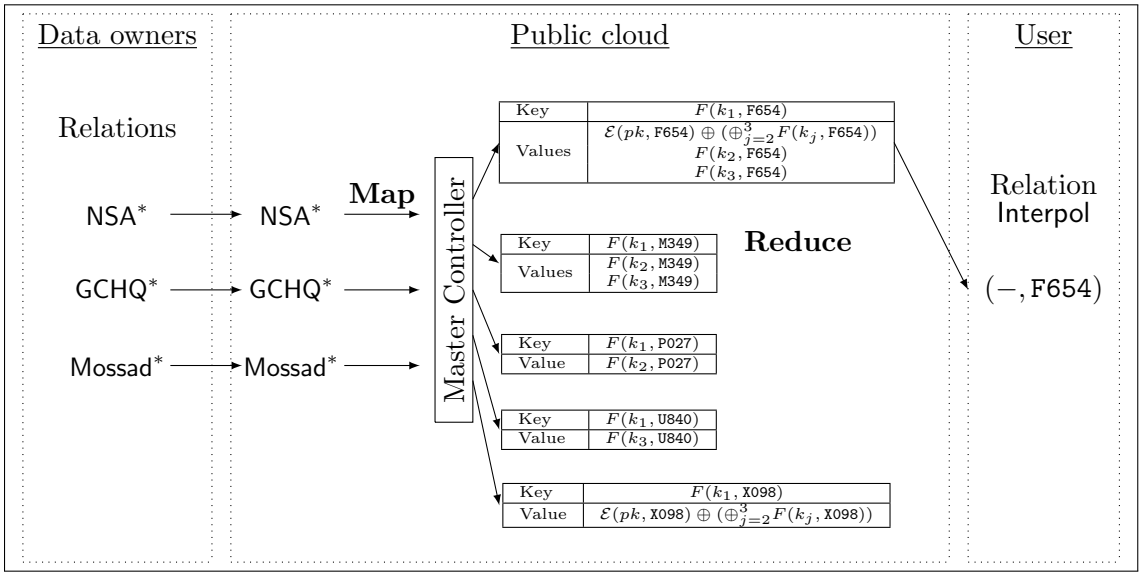


Figure 5.8: Example of intersection with MapReduce between three relations using our secure protocol SI. First, data owners outsource their respective encrypted relation on the public cloud. The public cloud runs the Map function, then the Reduce function verifies if keys are associated to a list of three values. In that case, the public cloud produces and sends encrypted tuples corresponding to the intersection to the user. In the example, the user obtains the tuples $(-, \mathcal{E}(pk, \texttt{F654}))$ equals to $(-, (\mathcal{E}(pk, \texttt{F654}) \oplus (\oplus_{j=2}^{3} F(k_j, \texttt{F654})) \oplus (\oplus_{j=2}^{3} F(k_j, \texttt{F654}))))$.

Proof. Let $R_1, R_2, \ldots R_n$ be n relations. Let $R_1^*, R_2^*, \ldots R_n^*$ be the corresponding encrypted relations computed by the preprocessing phase of SI. We set $R := \cap_{i=1}^{n} R_i$.

For each tuple $t \in R$, there exists a key-value pair in the main relation R_1^* of the form $(F(k_1, t), \mathcal{E}(pk, t) \oplus (\oplus_{i=2}^{n} F(k_i, t)))$, and a key-value pair in relation R_j^*, with $2 \leq j \leq n$, of the form $(F(k_1, t), F(k_j, t))$. Following the MapReduce paradigm, the n values are sent

to the same reducer that sums the corresponding values. Thus, for each key $F(k_1, t)$, with $t \in R$, we obtain

$$\mathcal{E}(pk, t) \oplus (\oplus_{i=2}^{n} F(k_i, t)) \oplus (\oplus_{i=2}^{n} F(k_i, t)) = \mathcal{E}(pk, t) \ .$$

Hence, for each $t \in R$, reducer associated to the key $F(k_1, t)$ emits the pair $(-, \mathcal{E}(pk, t))$ to the user. Moreover, for each $t \in \left(\cup_{i=1}^{n} R_i \right) \setminus R$, the reducer associated to the key $F(k_1, t)$ does not output the pair $(-, \mathcal{E}(pk, t))$ since it is associated to less than n values. Finally, SI produces pairs $(-, \mathcal{E}(pk, t))$ such that $t \in R$ corresponding to the intersection of relations $R_1, R_2, \ldots, R_n$ which concludes the proof. □

5.2.4 Complexity of Original and Secure Protocols

Our secure protocol SI is efficient from both computation and communication points of view. The overhead for the computation complexity is linear in the number of tuples by relation while the communication complexity is the same as in the standard protocol [LRU14]. We summarize in Figure 5.1 the trade-offs between computation and communication costs for our secure protocol SI vs the standard MapReduce protocol computing the intersection of $n \geq 2$ relations. We consider that the cost to access to one tuple in the relation is equal to 1. In our communication cost analysis, we measure the total size of the data that is emitted from a map or reduce node.

Table 5.1: Summary of results. Let $N = \max(|R_1|, \ldots, |R_n|)$ be the biggest cardinal of relations R_i with $i \in [\![1, n]\!]$. Let $C_\oplus$ be the computation cost of a bitwise exclusive OR operation).

Protocol	*Computation cost (big-O)*	*Communication cost (big-O)*
Standard	$2 \cdot n \cdot N$	$(n+1) \cdot N$
SI	$2 \cdot n \cdot N + (n-1) \cdot N \cdot C_\oplus$	$(n+1) \cdot N$

5.3 Experimental Results

We present the experimental results for the MapReduce protocol [LRU14], and for our secure protocol SI computing the intersection of relations with MapReduce. We run two types of experiments: the *experiment on the number of tuples*, and the *experiment on the number of intersected relations*.

5.3.1 Settings

According to SI protocol, we need a pseudo-random function and an asymmetric encryption scheme.

For the asymmetric encryption scheme, we use the RSA-OAEP asymmetric encryption scheme [BR94, KS98] implemented in Go package `rsa`[†] with a 1024-bit RSA modulus. Note that we can also use an other asymmetric encryption scheme such as ElGamal [Gam85], however ElGamal ciphertexts are twice longer for the same security and we do not use it for communication cost reason.

Concerning the pseudo-random function, we use the HMAC-SHA256 keyed-hash message authentication code [BCK96] implemented in Go package `hmac`[‡] to compute key of

[†] https://golang.org/pkg/crypto/rsa/
[‡] https://golang.org/pkg/crypto/hmac/

key-value pairs, i.e., for the pseudo-random function that uses key k_1. For the pseudo-random function used in values evaluations, we use AES [DR98] with *Cipher Block Chaining* (CBC) mode of operation [EMST78]. Indeed, we need the length of the pseudo-random evaluations to be equal to the length of the RSA-OAEP ciphertexts; which is possible with CBC mode. Note that we must consider a unique *initial vector* (IV) for each secret key; moreover, these IV are shared between the corresponding data owner and the data owner of the main relation.

5.3.2 Varying the Number of Tuples

In the experiment on the number of tuples, we consider two relations of the same schema composed of only one attribute whose values are integers. These two relations have the same cardinal C and share $C/2$ elements. Tuples of the first relation consist in all integers from 1 to C, while tuples of the second relation consist in all integers from $C/2$ to $C+C/2$. We run the original protocol [LRU14] and our secure protocol SI on couples of relations of cardinal $500,000$ to $3,000,000$, by step of $250,000$.

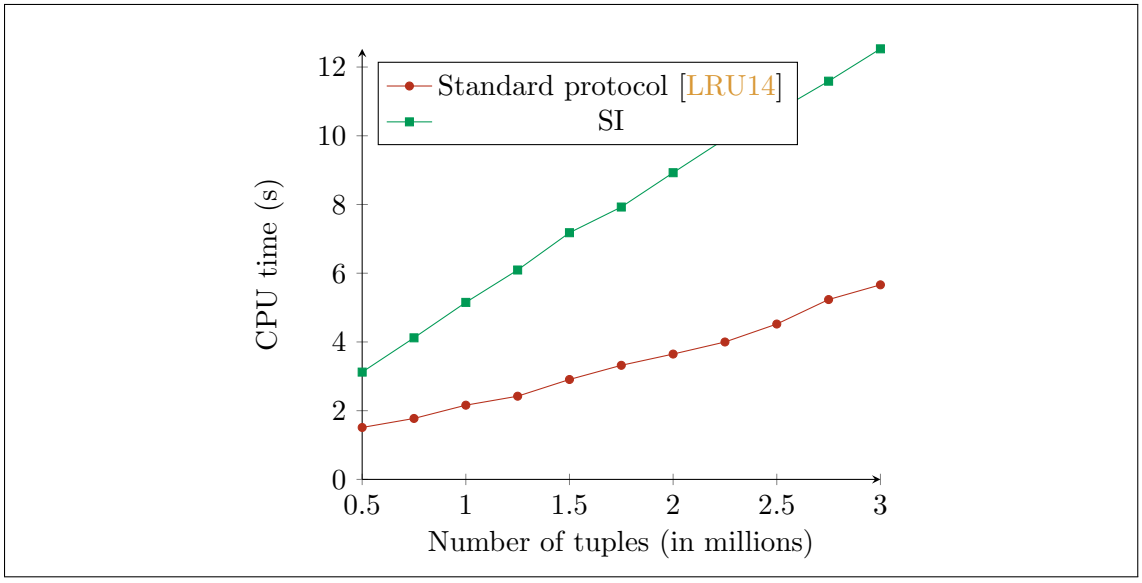


Figure 5.9: CPU time vs the number of tuples for the standard MapReduce protocol [LRU14] and our secure approach SI computing the intersection between two relations.

We run this experiment with the original protocol and our secure protocol SI, and show its results in Figure 5.9. We observe that the computation complexity of our secure protocol is linear as determined in the complexity study (cf. Figure 5.1)

5.3.3 Varying the Number of Intersected Relations

In this experiment, we consider intersection between different number of relations of the same schema composed of only one attribute whose values are integers. We start by computing the intersection between 2 relations to finish with the intersection between 10 relations. In each case, relations have $500,000$ tuples and shares $250,000$ tuples. In practice, for the intersection of $n \in [\![2, 10]\!]$ relations, tuples of the first relation consist in integers from 1 to $500,000$, and tuples of the i-th relation with $i \in [\![2, n]\!]$ consist in integers from 1 to $250,000$ and integers from $i \cdot 250,000 + 1$ to $(i+1) \cdot 250,000$.

We run the standard protocol [LRU14] and our secure protocol SI for the experiment on the number of intersected relations. We remark in Figure 5.10 that the computation complexity of our secure protocol is linear as determined in the complexity study (cf.

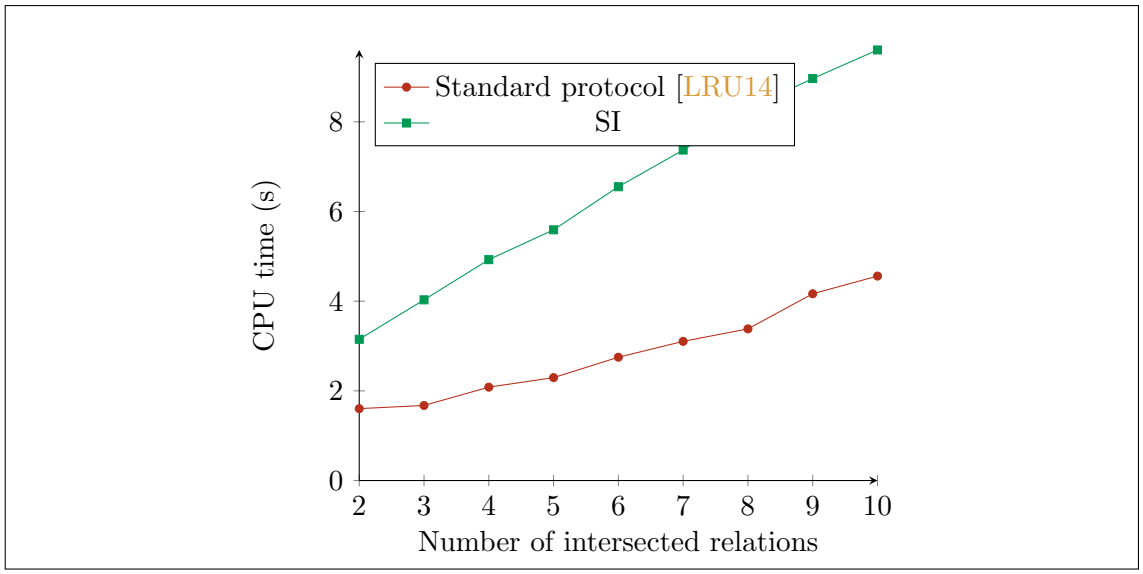


Figure 5.10: CPU time vs the number of intersected relations for the standard MapReduce protocol [LRU14] and our secure approach SI computing the intersection between two relations.

Figure 5.1). We observe that the computation complexity is less compared to the experiment on the number of tuples. Indeed, when we run the SI protocols with 10 relations of $500,000$ tuples (i.e., a total of $5,000,000$ tuples), the CPU time is approximately equal to 550 seconds while the CPU time for the intersection of 2 relations of 2 millions (i.e., a total of $4,000,000$ tuples) is approximately equal to $1,500$ seconds. This is due to number of common elements of each relation. In the case of the intersection of 10 relations (each composed of $500,000$ tuples), relations share $250,000$ while in the case of the intersection of the 2 relations (each composed of $2,000,000$ tuples), relations share $1,000,000$ tuples. Hence, the Reduce function has to perform a large number of exclusive or on 2048-bits strings.

5.4 Security Proof

In this section, we provide a formal security proof of our SI protocol by considering a semi-honest public cloud that may collude with the MapReduce's user. We assume $n \geq 2$ data owners respectively possessing a relation R_i, with $i[\![1, n]\!]$, such that R_i is constituted of N_i unique tuples. The public cloud computes the intersection of the n relation denoted R, such that $|R| = N$. In other terms, the n relations share N tuples, i.e., $R := \cap_{i=1}^{n} R_i$ with $|R| = N$.

5.4.1 Modeling of our SI Protocol

We start by modeling our SI protocol that computes the intersection of n relations by $n+2$ parties. The owner of the *main* relation R_1 is denoted $P_{\mathcal{R}_1}$, while other owners are denoted $P_{\mathcal{R}_i}$ for $i \in [\![2, n]\!]$ respectively. The public cloud is denoted $P_{\mathcal{C}}$, and the MapReduce's user is denoted by $P_{\mathcal{U}}$. Parties use respective inputs $I := (I_{\mathcal{R}_1}, \ldots, I_{\mathcal{R}_n}, I_{\mathcal{C}}, I_{\mathcal{U}}) \in \mathcal{I}$ and a function $g = (g_{\mathcal{R}_1}, \ldots, g_{\mathcal{R}_n}, g_{\mathcal{C}}, g_{\mathcal{U}})$ such that:

- $P_{\mathcal{R}_1}$ has the input $I_{\mathcal{R}_1} := (pk, k_1, \ldots, k_n, R_1)$ where pk is a public key of the cryptosystem Π , $k_i \in \mathcal{K}$ for $i \in [\![1, n]\!]$ are secret keys for the pseudo-random function F,

and R_1 is the relation owned by $P_{\mathcal{R}_1}$. The party $P_{\mathcal{R}_1}$ outputs $g_{\mathcal{R}_1}(I) = \perp$ (where $\perp$ means that the function returns nothing) since it does not learn any information.

- $P_{\mathcal{R}_i}$ for $i \in [\![2, n]\!]$ has the input $I_{\mathcal{R}_1} := (pk, k_1, k_i, R_i)$ where pk is a public key of the cryptosystem Π, $k_1 \in \mathcal{K}$ and $k_i \in \mathcal{K}$ are secret keys for the pseudo-random function F, and R_i is the relation owned by $P_{\mathcal{R}_i}$. The party $P_{\mathcal{R}_i}$ outputs $g_{\mathcal{R}_i}(I) = \perp$ since it does not learn any information.

- $P_{\mathcal{C}}$ has the input $I_{\mathcal{C}} := pk$ where pk is a public key the cryptosystem Π. It returns $g_{\mathcal{C}}(I) = (N_1, \ldots, N_n, N)$ because the party $P_{\mathcal{C}}$ learns the number of tuples in each relation R_i for $i \in [\![1, n]\!]$ and the number of tuples common to these n relations.

- $P_{\mathcal{U}}$ has the input $I_{\mathcal{U}} := (pk, sk)$ where (pk, sk) is a key pair of the cryptosystem Π. It returns $g_{\mathcal{U}}(I) = R$ where $R = \cap_{i=1}^{n} R_i$, i.e., the result of intersection between the n relations computed by the public cloud.

5.4.2 Proof

We prove that our protocol SI securely computes the intersection of n relations in the presence of a semi-honest adversaries even if the two parties $P_{\mathcal{C}}$ and $P_{\mathcal{U}}$ collude, i.e., if the public cloud and the MapReduce's user share all their respective public and private information. In other terms, we prove that the public cloud and the MapReduce's user do not learn extra information than they have even if the MapReduce's user shares her secret key sk with the public cloud. The security proof is given in Theorem 8.

Theorem 8. *Assume F is a secure pseudo-random function and that Π is an IND-CPA asymmetric encryption scheme, then the SI protocol securely computes the intersection of n relations in the presence of semi-honest adversaries even if parties $P_{\mathcal{C}}$ and $P_{\mathcal{U}}$ collude.*

The security proof for Theorem 8 is decomposed in Lemma 26 for parties $P_{\mathcal{R}_i}$ (with $i \in [\![1, n]\!]$), and in Lemma 27 for the collusion between parties $P_{\mathcal{C}}$ and $P_{\mathcal{U}}$.

Lemma 26. *There exist probabilistic polynomial-time simulators and $\mathcal{S}^{SI}_{\mathcal{R}_i}$ for $i \in [\![1, n]\!]$ such that*

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

Proof. We assume that $P_{\mathcal{R}_i}$, with $i \in [\![1, n]\!]$, is corrupted. We observe that $\mathcal{R}_i$ receives no output and no incoming message from other parties. Thus, we merely need to show that a simulator can generate the view of party $P_{\mathcal{R}_i}$ from its inputs.

In the protocol, $P_{\mathcal{R}_1}$ receives the public key pk of the MapReduce's user, the n secret keys $k_1, \ldots, k_n$ used with the pseudo-random function F, and the relation R_1. Parties $P_{\mathcal{R}_i}$, with $i \in [\![2, n]\!]$, receives the public key pk of the user, the secret keys k_1 and k_i for the pseudo-random function F, and the corresponding relation R_i Simulators $\mathcal{S}^{\text{SI}}_{\mathcal{R}_i}$, for $i \in [\![1, n]\!]$, work exactly as the preprocessing phase presented in Figure 5.5.

Hence, we remark that $\mathcal{S}_{\mathcal{R}_i}$, for all $i \in [\![1, n]\!]$, uses exactly the same algorithm as the real protocol SI, then it describes the same distribution as $\mathsf{view}^{\text{SI}}_{\mathcal{R}_i}(I, \lambda)$, which concludes the proof. □

Lemma 27. *Assume F is a secure pseudo-random function and Π is an IND-CPA asymmetric encryption scheme, then there exists a probabilistic polynomial-time simulator $\mathcal{S}^{SI}_{\mathcal{C},\mathcal{U}}$ such that*

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

Proof. Let $\lambda \in \mathbb{N}$ be a security parameter. Before building $\mathcal{S}^{\text{SI}}_{\mathcal{C},\mathcal{U}}$ that computes a distribution that can be simulated perfectly, we use the hybrid argument to build hybrid simulators denoted $\mathcal{S}^{H_i\text{-SI}}_{\mathcal{C},\mathcal{U}}$ for each $i \in [\![1, n]\!]$. The simulator, $\mathcal{S}^{H_i\text{-SI}}_{\mathcal{C},\mathcal{U}}$ works as SI but each evaluation of the pseudo-random function performed by parties $\mathcal{R}_j$ for $j \in [\![1, i]\!]$ are substituted using the random oracle OPRF presented in Figure 5.11.

$\mathsf{OPRF}(j, x)$:
if $T[j, x] = \emptyset$ **then**
  $T[j, x] \xleftarrow{\$} \{0,1\}^{|\mathcal{Y}|}$
return $T[j, x]$

Figure 5.11: Random oracle OPRF.

We start by presenting the simulator $\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}$ in Figure 5.12.

Simulator: $\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}(1^\lambda, pk, (N_1, \ldots, N_n, N))$
$R := \emptyset; R^* := \emptyset$
// Generate intersection result.
while $|R| < N$ **do**
  $r \xleftarrow{\$} \mathbb{N}$
  if $r \notin R$ **then**
    $R := R \cup \{r\}$
// Generate the n relations sharing only elements of R.
for $i \in [\![1, n]\!]$ **do**
  $k_i \xleftarrow{\$} \mathcal{K}$
  $R_i := R$
  while $|R_i| < N_i$ **do**
    $r \xleftarrow{\$} \mathbb{N}$
    $R_i := R_i \cup \{r\}$
    $R_i := R_i \setminus \left(\cup_{j=1}^{i-1} R_j \setminus R \right)$
// Generate protected relation R_1^* using oracle OPRF evaluations with k_1.
$R_1^* := \emptyset$
foreach $r \in R_1$ **do**
  $r^* := \mathcal{E}(pk, r)$
  $R_1^* := R_1^* \cup \{(\mathsf{OPRF}(k_1, r), (r^* \oplus F(k_2, r) \oplus \cdots \oplus F(k_n, r)))\}$
  if $r \in R$ **then**
    $R^* := R^* \cup \{r^*\}$
// Generate protected relations R_2^* to R_n^* using oracle OPRF evaluations
// with k_1.
for $i \in [\![2, n]\!]$ **do**
  $R_i^* := \emptyset$
  foreach $r \in R_i$ **do**
    $R_i^* := R_i^* \cup \{(\mathsf{OPRF}(k_1, r), F(k_i, r))\}$
$\mathsf{view} := (R_1^*, \ldots, R_n^*, R, R^*)$

Figure 5.12: Simulator $\mathcal{S}^{H_1-\text{SI}}_{\mathcal{C},\mathcal{U}}$ for the proof of Lemma 27.

Assume by contradiction that there exists a distinguisher $D \in \text{PPT}(\lambda)$ such that for all inputs I, we have

$$\left| \Pr[D(\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)), (1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I)))) = 1] - \Pr[D(\mathsf{view}^{\text{SI}}_{\mathcal{C},\mathcal{U}}(I, \lambda)) = 1] \right| = \mu(\lambda) \ ,$$

where μ is a non-negligible function in λ.

We construct a guessing adversary $\mathcal{A} \in \text{PPT}(\lambda)$ that uses D to win the PRF experiment against the pseudo-random function family F defined in Figure 2.1 on page 19. Adversary $\mathcal{A}$ is presented in Figure 5.13.

Adversary: $\mathcal{A}(pk)$

$R := \emptyset; R^* := \emptyset$
while $|R| < N$ **do**
  $r \xleftarrow{\$} \mathbb{N}$
  if $r \notin R$ **then**
    $R := R \cup \{r\}$
for $i \in [\![1, n]\!]$ **do**
  $k_i \xleftarrow{\$} \mathcal{K}$
  $R_i := R$
  while $|R_i| < N_i$ **do**
    $r \xleftarrow{\$} \mathbb{N}$
    $R_i := R_i \cup \{r\}$
    $R_i := R_i \setminus \left(\cup_{j=1}^{i-1} R_j \setminus R \right)$
$R_1^* := \emptyset$
foreach $r \in R_1$ **do**
  $r^* := \mathcal{E}(pk, r)$
  $R_1^* := R_1^* \cup \{(f_b(r), (r^* \oplus F(k_2, r) \oplus \cdots \oplus F(k_n, r)))\}$
  if $r \in R$ **then**
    $R^* := R^* \cup \{r^*\}$
for $i \in [\![2, n]\!]$ **do**
  $R_i^* := \emptyset$
  foreach $r \in R_i$ **do**
    $R_i^* := R_i^* \cup \{(f_b(r), F(k_i, r))\}$
$\mathsf{view} := (R_1^*, \ldots, R_n^*, R, R^*)$
$b_* := D(\mathsf{view})$
return b_*

Figure 5.13: Adversary $\mathcal{A}(pk)$.

First, when $b = 0$, we remark that

$$\Pr[\mathsf{Exp}^{\text{prf-0}}_{F,\mathcal{A}}(\lambda) = 1] = \Pr[D(\mathsf{view}^{\text{SI}}_{\mathcal{C},\mathcal{U}}(I, \lambda)) = 1] \ .$$

Indeed, when $b = 0$ the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol SI. Then the probability that the experiment $\mathsf{Exp}^{\text{prf-0}}_{F,\mathcal{A}}$ returns 1 is equal to the probability that the distinguisher D returns 1 on input computed by the real protocol.

On the other hand, when $b = 1$, we have

$$\Pr[\mathsf{Exp}^{\text{prf-1}}_{F,\mathcal{A}}(\lambda) = 1] = \Pr[D(\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_\mathcal{C}(I)), (1^\lambda, I_\mathcal{U}, g_\mathcal{U}(I)))) = 1] \ .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}$. Then the probability that the experiment $\mathsf{Exp}^{\text{prf-1}}_{F,\mathcal{A}}$ returns 1 is equal to the probability that the distinguisher D returns 1 on input computed by the simulator $\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}$.

It therefore follows that

$$\begin{aligned}\mathsf{Adv}^{\mathrm{prf}}_{F,\mathcal{A}}(\lambda) &= \big|\Pr[\mathsf{Exp}^{\text{prf-1}}_{F,\mathcal{A}}(\lambda)=1] - \Pr[\mathsf{Exp}^{\text{prf-0}}_{F,\mathcal{A}}(\lambda)=1]\big| \\ &= \big|\Pr[D(\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)),(1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I))))=1] \\ &\quad - \Pr[D(\mathsf{view}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}(I,\lambda))=1]\big| \\ &= \mu(\lambda)\ ,\end{aligned}$$

which is non-negligible. However, we assume that F is a secure pseudo-random function, hence, it does not exist D such that

$$\big|\Pr[D(\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)),(1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I))))=1] - \Pr[D(\mathsf{view}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}(I,\lambda))=1]\big|\ ,$$

is non-negligible. Hence, we have

$$\left\{\mathcal{S}^{H_1\text{-SI}}_{\mathcal{C},\mathcal{U}}((I_\mathcal{C}, g_{I_\mathcal{C}}(I,\lambda)),(I_\mathcal{U}, g_{I_\mathcal{U}}(I,\lambda)))\right\}_{\lambda\in\mathbb{N},I\in\mathcal{I}} \overset{\mathrm{c}}{\equiv} \left\{\mathsf{view}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}(I,\lambda)\right\}_{\lambda\in\mathbb{N},I\in\mathcal{I}}\ .$$

The construction of simulators $\mathcal{S}^{H_i\text{-SI}}_{\mathcal{C},\mathcal{U}}$ for $i\in[\![2,n]\!]$ is very similar. The only difference is that evaluations of the pseudo-random function F are replaced using the oracle OPRF presented in Figure 5.11. We prove as previous we have for $\lambda\in\mathbb{N}$, $I\in\mathcal{I}$, and $i\in[\![1,n-1]\!]$

$$\mathcal{S}^{H_i\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_\mathcal{C}(I)),(1^\lambda, I_\mathcal{U}, g_\mathcal{U}(I))) \overset{\mathrm{c}}{\equiv} \mathcal{S}^{H_{i+1}\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_\mathcal{C}(I)),(1^\lambda, I_\mathcal{U}, g_\mathcal{U}(I)))\ .$$

Finally, we show how to build the simulator $\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}$. The difference between $\mathcal{S}^{H_n\text{-SI}}_{\mathcal{C},\mathcal{U}}$ and $\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}$ is that $\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}$ substitutes Π encryption of real values by Π encryption of random values of the same size. More formally, $\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}$ is presented in Figure 5.14.

Now we show that for $\lambda\in\mathbb{N}$ and $I\in\mathcal{I}$, we have

$$\mathcal{S}^{H_n\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)),(1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I))) \overset{\mathrm{c}}{\equiv} \mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)),(1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I)))\ .$$

Assume there exists a distinguisher $D\in\mathrm{PPT}(\lambda)$ such that for all inputs I, we have

$$\begin{aligned}\big|\Pr[D(\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)),(1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I))))=1] \\ - \Pr[D(\mathcal{S}^{H_n\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)),(1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I))))=1]\big| = \mu(\lambda)\ ,\end{aligned}$$

where $\mu(\cdot)$ is a non-negligible function in λ.

We construct a guessing adversary $\mathcal{B}\in\mathrm{PPT}(\lambda)$ that uses D to win the IND-CPA experiment. Adversary $\mathcal{B}$ is presented in Figure 5.15.

First, we remark that

$$\Pr[\mathsf{Exp}^{\text{indcpa-0}}_{\Pi,\mathcal{B}}(\lambda)=1] = \Pr[D(\mathcal{S}^{H_n\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)),(1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I))))=1]\ .$$

When $b=0$, the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}^{H_n\text{-SI}}_{\mathcal{C},\mathcal{U}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}^{H_n\text{-SI}}_{\mathcal{C},\mathcal{U}}$. On the other hand, we have

$$\Pr[\mathsf{Exp}^{\text{indcpa-1}}_{\Pi,\mathcal{B}}(\lambda)=1] = \Pr[D(\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_\mathcal{C}, g_{I_\mathcal{C}}(I)),(1^\lambda, I_\mathcal{U}, g_{I_\mathcal{U}}(I))))=1]\ .$$

Indeed, when $b=1$ the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}$.

Simulator: $\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}(1^\lambda, pk, (N_1, \ldots, N_n, N))$

$R := \emptyset; R^* := \emptyset$
while $|R| < N$ **do**
  $r \xleftarrow{\$} \mathbb{N}$
  if $r \notin R$ **then**
    $R := R \cup \{r\}$
for $i \in [\![1, n]\!]$ **do**
  $k_i \xleftarrow{\$} \mathcal{K}$
  $R_i := R$
  while $|R_i| < N_i$ **do**
    $r \xleftarrow{\$} \mathbb{N}$
    $R_i := R_i \cup \{r\}$
    $R_i := R_i \setminus \left(\cup_{j=1}^{i-1} R_j \setminus R \right)$
$R_1^* := \emptyset$
foreach $r \in R_1$ **do**
  if $r \in R$ **then**
    $r^* := \mathcal{E}(pk, r)$
    $R^* := R^* \cup \{r^*\}$
    $R_1^* := R_1^* \cup \{(\mathsf{OPRF}(k_1, r), (r^* \oplus \mathsf{OPRF}(k_2, r) \oplus \cdots \oplus \mathsf{OPRF}(k_n, r)))\}$
  else
    $r' \xleftarrow{\$} \mathbb{N}$
    $r^* := \mathcal{E}(pk, r')$
    $R_1^* := R_1^* \cup \{(\mathsf{OPRF}(k_1, r), (r^* \oplus \mathsf{OPRF}(k_2, r) \oplus \cdots \oplus \mathsf{OPRF}(k_n, r)))\}$
for $i \in [\![2, n]\!]$ **do**
  $R_i^* := \emptyset$
  foreach $r \in R_i$ **do**
    $R_i^* := R_i^* \cup \{(\mathsf{OPRF}(k_1, r), \mathsf{OPRF}(k_i, r))\}$
$\mathsf{view} := (R_1^*, \ldots, R_n^*, R, R^*)$

Figure 5.14: Simulator $\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}$ for the proof of Lemma 27.

Finally, we evaluate the probability that $\mathcal{B}$ wins the experiment

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{indcpa}}_{\Pi,\mathcal{B}}(\lambda) &= \left| \Pr[\mathsf{Exp}^{\text{indcpa-1}}_{\Pi,\mathcal{B}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{indcpa-0}}_{\Pi,\mathcal{B}}(\lambda) = 1] \right| \\
&= \big| \Pr[D(\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)), (1^\lambda, I_{\mathcal{U}}, g_{I_{\mathcal{U}}}(I)))) = 1] \\
&\quad - \Pr[D(\mathcal{S}^{H_n\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)), (1^\lambda, I_{\mathcal{U}}, g_{I_{\mathcal{U}}}(I)))) = 1] \big| \\
&= \mu(\lambda) \ ,
\end{aligned}$$

which is non-negligible. However, we assume that Π is IND-CPA. Hence, for $\lambda \in \mathbb{N}$ and $I \in \mathcal{I}$, we have

$$\mathcal{S}^{H_n\text{-SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)), (1^\lambda, I_{\mathcal{U}}, g_{I_{\mathcal{U}}}(I))) \stackrel{\mathrm{c}}{\equiv} \mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)), (1^\lambda, I_{\mathcal{U}}, g_{I_{\mathcal{U}}}(I))) \ .$$

By transitivity, we have

$$\left\{\mathcal{S}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}((1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)), (1^\lambda, I_{\mathcal{U}}, g_{I_{\mathcal{U}}}(I)))\right\}_{\lambda\in\mathbb{N}, I\in\mathcal{I}} \stackrel{\mathrm{c}}{\equiv} \left\{\mathsf{view}^{\mathrm{SI}}_{\mathcal{C},\mathcal{U}}(I, \lambda)\right\}_{\lambda\in\mathbb{N}, I\in\mathcal{I}} \ ,$$

which concludes the proof. □

Adversary: $\mathcal{B}(pk)$

$R := \emptyset; R^* := \emptyset$
while $|R| < N$ **do**
  $r \xleftarrow{\$} \mathbb{N}$
  if $r \notin R$ **then**
    $R := R \cup \{r\}$
for $i \in [\![1, n]\!]$ **do**
  $k_i \xleftarrow{\$} \mathcal{K}$
  $R_i := R$
  while $|R_i| < N_i$ **do**
    $r \xleftarrow{\$} \mathbb{N}$
    $R_i := R_i \cup \{r\}$
    $R_i := R_i \setminus \left(\cup_{j=1}^{i-1} R_j \setminus R \right)$
$R_1^* := \emptyset$
foreach $r \in R_1$ **do**
  if $r \in R$ **then**
    $r^* := \mathcal{E}(pk, r)$
    $R^* := R^* \cup \{r^*\}$
    $R_1^* := R_1^* \cup \{(\mathsf{OPRF}(k_1, r), (r^* \oplus \mathsf{OPRF}(k_2, r) \oplus \cdots \oplus \mathsf{OPRF}(k_n, r)))\}$
  else
    $r' \xleftarrow{\$} \mathbb{N}$
    $r^* := \mathcal{E}(pk, \mathsf{LoR}_b(r, r'))$
    $R_1^* := R_1^* \cup \{(\mathsf{OPRF}(k_1, r), (r^* \oplus \mathsf{OPRF}(k_2, r) \oplus \cdots \oplus \mathsf{OPRF}(k_n, r)))\}$
for $i \in [\![2, n]\!]$ **do**
  $R_i^* := \emptyset$
  foreach $r \in R_i$ **do**
    $R_i^* := R_i^* \cup \{(\mathsf{OPRF}(k_1, r), \mathsf{OPRF}(k_i, r))\}$
$\mathsf{view} := (R_1^*, \ldots, R_n^*, R, R^*)$
$b_* := D(\mathsf{view})$
return b_*

Figure 5.15: Adversary $\mathcal{B}$ for the proof of Lemma 27.

5.5 Conclusion

We have presented an efficient privacy-preserving protocol called SI (for *Secure Intersection*) using the MapReduce paradigm to compute the intersection between an arbitrary number of relations. In fact, in the protocol proposed by Leskovec et al. [LRU14], the public cloud performing the computation learns all tuples of the data owners along the intersection result that it sends to the MapReduce's user. In our protocol SI, the public cloud cannot learn information on the input sets. Moreover, if the cloud and the user collude, i.e., the cloud knows the secret key of the asymmetric encryption scheme used by the user, then they cannot learn more than the final result of the intersection. If no such a collusion exists, then the public cloud only learns cardinals of the relations sent by the data owner, the cardinal of their intersection, and relations sharing a common element (but not the element itself).

To achieve our goal, we have relied on pseudo-random functions, asymmetric encryption and bitwise exclusive OR operations. We have compared the standard and our secure approach SI with respect to three fundamental criteria: computation cost, communication cost, and privacy guarantees. We also implemented SI protocol with the Apache Hadoop® framework [Fou19b] showing the scalability of our secure approach.

Looking forward to future work, we plan to study secure intersection with MapReduce considering a malicious public cloud, i.e., the public cloud can perform any operations on data that it process. We would like also to hide to the public cloud the information about common elements that are shared between relations. Additionally, we aim to investigate the computation of a secure relation difference using the MapReduce paradigm.

CHAPTER 6

Secure Grouping and Aggregation with MapReduce

We propose secure multiparty protocols computing grouping and aggregation operations on a relation using the MapReduce paradigm. We focus on a scenario where the data owner outsources her relation on an *semi-honest* public cloud. More specifically, we evaluate grouping and aggregation with COUNT, SUM, AVG, MIN, and MAX operations. At the end of the computation, the public cloud sends the final result to an external user. We assume that the public cloud does not know user's private information, i.e., the public cloud and the user do not collude. This work has been conducted in collaboration with Radu Ciucanu, Pascal Lafourcade, and Lihua Ye, and has been published in the paper "*Secure Grouping and Aggregation with MapReduce*" [CGLY18] at the 15th *International Conference on Security and Cryptography* (SECRYPT, 2018).

Contents

6.1 Introduction . . . 107
6.1.1 Grouping and Aggregation with MapReduce . . . 107
6.1.2 Problem Statement . . . 108
6.1.3 Contributions . . . 109
6.1.4 Outline . . . 109
6.2 Grouping and Aggregation with MapReduce . . . 109
6.2.1 MapReduce Grouping and Aggregation with COUNT Operation 109
6.2.2 MapReduce Grouping and Aggregation with SUM Operation . . 110
6.2.3 MapReduce Grouping and Aggregation with AVG Operation . . 110
6.2.4 MapReduce Grouping and Aggregation with MIN Operation . . 111
6.3 Secure Grouping and Aggregation with MapReduce . . . 112
6.3.1 Preprocessing for SGA Protocols . . . 112
6.3.2 Secure MapReduce Grouping and Aggregation with COUNT Operation . . . 113
6.3.3 Secure MapReduce Grouping and Aggregation with SUM Operation 114
6.3.4 Secure MapReduce Grouping and Aggregation with AVG Operation 114
6.3.5 Secure MapReduce Grouping and Aggregation with MIN Operation 116
6.3.6 Proof of Correctness . . . 116
6.3.7 Complexity of Originals and SGA Protocols . . . 117
6.4 Experimental Results . . . 118
6.4.1 Dataset and Settings . . . 118

6.4.2 Results 118
6.5 Security Proof 119
6.5.1 Modeling of SGA_{SUM} Protocol 119
6.5.2 Security Proof of SGA_{SUM} Protocol 120
6.5.3 Modeling of SGA_{MIN} Protocol 124
6.5.4 Security Proof of SGA_{MIN} Protocol 124
6.6 Conclusion 129

6.1 Introduction

We address the fundamental problem of how to group and aggregate data from a relation in a privacy-preserving manner using the MapReduce paradigm [DG04]. We assume that the data is externalized in a public cloud by the data owner and there is a user, called MapReduce's user, that queries it. We consider the following five aggregation operations, which are precisely those included in the *Structured Query Language* (SQL) standard: COUNT, SUM, AVG, MIN, and MAX. We recall that COUNT operation counts the number of rows in a relation that matches a specified criteria, SUM operations computes the total sum of a numeric column, AVG operation computes the average value of a numeric column, while MIN (resp. MAX) computes the smallest (resp. the largest) value of the selected column.

We start by a running example to present the concepts of grouping and aggregation and of MapReduce computations. Then, we present our problem statement and illustrate with the same example the privacy issues related to grouping and aggregation with MapReduce.

Example 1. *Assume there is a university storing a relation R corresponding to the list of professors with their associated department and salary. The grouping and aggregation operation on the relation R, in the case where we assume one group attribute and one aggregate function, is denoted by $\gamma_{A,\theta(B)}(R)$, where A is the grouping attribute and θ is one of the five aggregation operations applied on the attribute B different from the grouping attribute. In this example (Figure 6.1), we consider the attribute "Department" as the grouping attribute and SUM is the aggregation operation applied on attribute "Salary". Hence, for each department we sum all the associated salaries. Since Alice and Bob are in the Computer Science department, the sum of salaries associated to the Computer Science department is $1,900 + 1,800 = 3,700$. In the same way, we sum the salaries of Mallory and Oscar from the Mathematics department. Since Eve is the only one in the Physics department, the sum corresponds to the salary of Eve which is equal to $2,000$. For the query $\gamma_{Department,SUM(Salary)}(R)$, we obtain the relation presented in Figure 6.2. Aggregation operations COUNT, AVG, MIN, or MAX work similarly.*

Name	Department	Salary
Alice	Computer Science	1,900
Mallory	Mathematics	1,750
Bob	Computer Science	1,800
Eve	Physics	2,000
Oscar	Mathematics	1,600

Figure 6.1: Relation R.

Department	SUM (salary)
Computer Science	3,700
Physics	2,000
Mathematics	3,350

Figure 6.2: Result of $\gamma_{\text{Department,SUM(Salary)}}(R)$.

6.1.1 Grouping and Aggregation with MapReduce

A protocol to perform grouping and aggregation with MapReduce is presented by Leskovec et al. [LRU14, Chapter 2]. First, a set of nodes of the public cloud has chunks of the

relation. The Map function creates for each tuple of the relation a *key-value* pair where *key* is equal to the value of the grouping attribute in the considered tuple, and *value* is equal to the value of the aggregation attribute of the considered tuple. Then, key-value pairs are grouped by key, i.e., key-value pairs output by the Map phase which have the same key are sent to the same reducer. For each key, the Reduce function applies the aggregate function on the associated values of the considered key.

Example 2. *Following Example 6.1, we perform grouping and aggregation with MapReduce on the relation R where the grouping attribute is the attribute "Department", the aggregation attribute is the attribute "Salary", and the operation is the SUM operation. We start grouping and aggregation with MapReduce by applying the map function. Since the grouping attribute is the attribute "Department" and that the aggregation attribute is the attribute "Salary", the map function emits the pairs:* $(Computer\ Science, 1, 900)$, $(Mathematics, 1, 750)$, $(Computer\ Science, 1, 800)$, $(Physics, 2, 000)$, $(Mathematics, 1, 600)$. *Pairs sharing the same key (i.e., same value of the grouping attribute) are sent on the same reducer via the master controller. Then, the reduce function performs on each reducer the aggregation, consisting here of the sum, and we obtain the pairs* $(Computer\ Science, 3, 700)$ *since* $1, 900 + 1, 800 = 3, 700$, $(Mathematics, 3, 350)$ *since* $1, 750 + 1, 600 = 3, 350$, *and* $(Physics, 2, 000)$. *The final result is presented in Figure 6.2.*

6.1.2 Problem Statement

We assume three entities: a data owner of a relation R, a public cloud, and a user that we call the MapReduce's user. This user that does not know the schema of R is authorized to query a grouping and aggregation operation on R.

First, R is outsourced by the data owner to the distributed file system of some public cloud provider. We assume that the relation R is initially spread over a set $\mathcal{R}_1$ of nodes of the public cloud, each of them storing a chunk of R, i.e., a set of tuples of R. Then, the result of the query, denoted $\gamma_{A,\theta(B)}(R)$, is computed over a set $\mathcal{R}_2$ of nodes before it is sent to the user's nodes $\mathcal{U}$.

We assume that the data owner and the MapReduce's user are trustworthy while the cloud service provider is semi-honest [Lin17], i.e., it dutifully executes the protocol but tries to deduce as much as possible information on the relation. Moreover, we assume the public cloud and the MapReduce's user do not collude, but that public cloud sets of nodes may collude. We illustrate the architecture of the MapReduce grouping and aggregation in Figure 6.3.

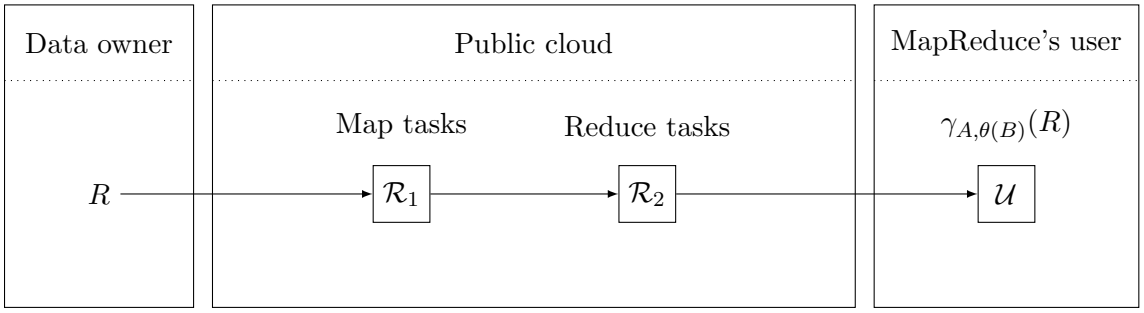


Figure 6.3: The system architecture.

In order to preserve the privacy of the data owner, the public cloud should not learn any plain input data, contrary to what happens for original protocols [LRU14, Chapter 2] and exemplified above. Indeed, the public cloud receives the original relation R from the data owner, hence it learns the content of all tuples of the relation R and the result of the query that it sends to the user.

We expect the public cloud sets of nodes $\mathcal{R}_1$ and $\mathcal{R}_2$ cannot learn any information about the relation R apart from repetitions of tuples within relation.

6.1.3 Contributions

We propose the following contributions.

- We revisit the original protocols for MapReduce grouping and aggregation [LRU14, Chapter 2] to guarantee the privacy of the data owner by designing a secure approach, called SGA (for *Secure Grouping Aggregation*), for each of the considered five aggregation operations, i.e., COUNT, SUM, AVG, MIN, and MAX. We denote by $\text{SGA}_{\text{COUNT}}$, SGA_{SUM}, SGA_{AVG}, SGA_{MIN}, and SGA_{MAX} the corresponding protocols in this secure approach. In each case, the secure approach is efficient from both computational and communication points of view, in the sense that the overhead is linear for each of the two complexity measures. Our technique is essentially based on two types of encryption schemes: (i) the well-known additive-homomorphic Paillier's cryptosystem [Pai99] used for COUNT, SUM, and AVG operations, and (ii) the order-preserving symmetric encryption scheme [AKSX04] for MIN and MAX operations.

- For each aggregation operation, we give experimental results for the original protocol and our secure approach. Experiments are run using the open-source MapReduce implementation Apache Hadoop® [Fou19b].

- We give security proofs of our secure approach in the random oracle model.

6.1.4 Outline

We first recall the original protocols to perform grouping and aggregation with operations COUNT, SUM, AVG, MIN, and MAX [LRU14]. Then, we present our secure protocols in Section 6.2. In Section 6.4, we show an experimental evaluation of original and our secure protocols using Apache Hadoop® [Fou19b] which provides a MapReduce implementation. Before to conclude this chapter, we prove in Section 6.5 the security properties of our secure protocols in the random oracle model.

6.2 Grouping and Aggregation with MapReduce

We recall original protocols proposed by Leskovec et al. [LRU14, Chapter 2] that compute the grouping and aggregation with COUNT, SUM, AVG, MIN, and MAX aggregation operations.

For each operation, we denote by A the grouping attribute, and by B the aggregation attribute. Note that MIN operation requires that the aggregated attribute have a type that can be compared, e.g., numbers or strings, while SUM and AVG require that the types allow arithmetic operations. We recall that we denote a grouping and aggregation operation on a relation R by $\gamma_{A,\theta(B)}(R)$, where A is the grouping attribute, θ is one of the five aggregation operations such as SUM, and B is the aggregation attribute. Moreover, we denote by $\pi_A(t)$ the component for the attribute A of the tuple t.

6.2.1 MapReduce Grouping and Aggregation with COUNT Operation

The Map and Reduce functions for the grouping and aggregation with COUNT operation are presented in Figure 6.4.

- *The Map Function.* For each tuple of a relation R, it produces a key-value pair where the key is equal to the grouping attribute value and the value is equal to 1.
- *The Reduce Function.* For a specified key, the Reduce function sums all associated values 1 to obtain the number of tuples in R sharing the same grouping attribute value.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R
// *value*: collection of $t \in R$
foreach $t \in R$ **do**
| $\texttt{emit}_{\mathcal{R}_1 \to \mathcal{R}_2}(\pi_A(t), 1)$

Reduce function:

Input: $(key, values)$
// *key*: $\pi_A(t)$ for $t \in R$
// *values*: collection of 1
count := 0
foreach $1 \in values$ **do**
| count := count + 1
$\texttt{emit}_{\mathcal{R}_2 \to \mathcal{U}}(\pi_A(t), \text{count})$

Figure 6.4: Map and Reduce functions for the COUNT protocol.

6.2.2 MapReduce Grouping and Aggregation with SUM Operation

The Map and Reduce function for the MapReduce grouping and aggregation with SUM operation are presented in Figure 6.5.

- *The Map Function.* For each tuple of a relation R, it produces a key-value pair where the key is equal to the grouping attribute value and the value is equal the aggregation attribute value.
- *The Reduce Function.* For a specified key, the Reduce function sums all associated values to obtain the sum of aggregation attribute values in R sharing the same grouping attribute value.

6.2.3 MapReduce Grouping and Aggregation with AVG Operation

The Map and Reduce functions for the MapReduce grouping and aggregation with AVG operation are presented in Figure 6.7.

- *The Map Function.* It works as the Map function for SUM operation. For each tuple of a relation R, it produces a key-value pair where the key is equal to the grouping attribute value and the value is equal the aggregation attribute value.
- *The Reduce Function.* It combines the Reduce function of COUNT and SUM operations. For a specified key, the Reduce function sums all associated values. Moreover, it counts the numbers of summed values. Hence, it can compute the average of aggregation attribute values in R sharing the same grouping attribute value.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R
// *value*: collection of $t \in R$
foreach $t \in R$ **do**
  | $\texttt{emit}_{\mathcal{R}_1 \to \mathcal{R}_2}(\pi_A(t), \pi_B(t))$

Reduce function:

Input: $(key, values)$
// *key*: $\pi_A(t)$ for $t \in R$
// *values*: collection of $\pi_B(t)$ with $t \in R$
$\mathsf{sum} \leftarrow 0$
foreach $\pi_B(t) \in values$ **do**
  | $\mathsf{sum} := \mathsf{sum} + \pi_B(t)$
$\texttt{emit}_{\mathcal{R}_2 \to \mathcal{U}}(\pi_A(t), \mathsf{sum})$

Figure 6.5: Map and Reduce functions for the SUM protocol.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R
// *value*: collection of $t \in R$
foreach $t \in R$ **do**
  | $\texttt{emit}_{\mathcal{R}_1 \to \mathcal{R}_2}(\pi_A(t), \pi_B(t))$

Reduce function:

Input: $(key, values)$
// *key*: $\pi_A(t)$ for $t \in R$
// *values*: collection of $\pi_B(t)$ with $t \in R$
$\mathsf{cpt} := 0$
$\mathsf{sum} := 0$
foreach $\pi_B(t) \in values$ **do**
  | $\mathsf{cpt} := \mathsf{cpt} + 1$
  | $\mathsf{sum} := \mathsf{sum} + \pi_B(t)$
$\texttt{emit}_{\mathcal{R}_2 \to \mathcal{U}}(\pi_A(t), \mathsf{sum}/\mathsf{cpt})$

Figure 6.6: Map and Reduce functions for the AVG protocol.

6.2.4 MapReduce Grouping and Aggregation with MIN Operation

The Map and Reduce functions for the MapReduce grouping and aggregation with MIN operation are presented in Figure 6.7.

- *The Map Function.* It works as the Map function for SUM and AVG operation. For each tuple of a relation R, it produces a key-value pair where the key is equal to the grouping attribute value and the value is equal the aggregation attribute value.
- *The Reduce Function.* For the considered key, the Reduce function determines the minimum of associated values. Hence, it can compute the minimum of aggregation attribute values in R sharing the same grouping attribute value.

The Map and Reduce functions for the MAX operation are very similar to the Map and

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R
// *value*: collection of $t \in R$
foreach $t \in R$ **do**
  $\texttt{emit}_{\mathcal{R}_1 \to \mathcal{R}_2}(\pi_A(t), \pi_B(t))$

Reduce function:

Input: $(key, values)$
// *key*: $\pi_A(t)$ for $t \in R$
// *values*: collection of $\pi_B(t)$ with $t \in R$
$\mathsf{min} \xleftarrow{\$} values$
foreach $\pi_B(t) \in values$ **do**
  if $\pi_B(t) < \mathsf{min}$ **then**
    $\mathsf{min} := \pi_B(t)$
$\texttt{emit}_{\mathcal{R}_2 \to \mathcal{U}}(\pi_A(t), \mathsf{min})$

Figure 6.7: Map and Reduce functions for the MIN protocol.

Reduce functions of the MIN operation so we do not present them. The only difference is in the Reduce function which checks the maximum of values associated to the considered key, instead of the minimum.

6.3 Secure Grouping and Aggregation with MapReduce

We present our secure protocols that computes the MapReduce grouping and aggregation for COUNT, SUM, AVG, MIN, and MAX operations. We call these protocols $\text{SGA}_{\text{COUNT}}$, SGA_{SUM}, SGA_{AVG}, SGA_{MIN}, and SGA_{MAX} respectively, where SGA stands for *Secure Grouping Aggregation*.

In order to compute MapReduce grouping and aggregation with these five operations in a privacy-preserving way, we need a pseudo-random function denoted F and defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, an additive-homomorphic asymmetric encryption scheme $\Pi := (\mathcal{G}, \mathcal{E}, \mathcal{D})$, and an order-preserving encryption scheme $\mathsf{\Pi} := (\mathsf{G}, \mathsf{E}, \mathsf{D})$, defined in Definition 16 on page 20.

We start by presenting the preprocessing algorithm of our secure MapReduce protocols.

6.3.1 Preprocessing for SGA Protocols

Before the data owner outsources its relation R to the public cloud, it performs a preprocessing on R to obtain a *protected* relation denoted R^*. In order to build this protected relation, the data owner picks randomly a secret $k \in \mathcal{K}$ for the pseudo-random function F, and a secret K for the order-preserving encryption scheme $\mathsf{\Pi}$. Moreover, the data owner has to get user's public key pk back for the additive-homomorphic asymmetric encryption scheme Π.

This preprocessing has two goals: (i) it protects owner's data in order to avoid the public cloud to learn tuples of the relation R, (ii) it allows the public cloud to perform the grouping and aggregation operations without revealing the content of the result to the public cloud. We present the preprocessing algorithm in Figure 6.8.

For each tuple $t \in R$, the preprocessing generates a key-value pair in the encrypted relation R^*. We denote by A the grouping attribute, and we denote by B the aggregation

Preprocessing:

Input: R
$R^* := \emptyset$
foreach $t \in R$ **do**
  $R^* := R^* \cup \{(F(k, \pi_A(t)), (\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)), \mathsf{E}^{\mathsf{ope}}(K, \pi_B(t))))\}$
return R^*

Figure 6.8: Preprocessing algorithm of SGA protocols.

attribute. For a tuple $t \in R$, value of the grouping attribute, i.e., $\pi_A(t)$, is evaluated using $F(k, \cdot)$. These deterministic pseudo-random function evaluations allow the public cloud to perform equality checks between values of grouping attributes. Hence, tuples having same values of grouping attribute are sent to the same reducer. In addition to these pseudo-random evaluations, the preprocessing algorithm also encrypts, for each tuple $t \in R$, the value of the grouping attribute with the public key pk of the user using the asymmetric encryption scheme Π. These encryptions allow the user to retrieve the value of grouping attributes associated to the aggregation value. Indeed, pseudo-random evaluations are non-inversible.

Moreover, for COUNT, SUM and AVG operations, the data owner encrypts each value of the aggregation attribute with the additive-homomorphic asymmetric encryption scheme Π using the user public key pk in order to allow the public cloud to perform the aggregation computation in encrypted values. For the MIN and MAX operations, the data owner also encrypts value of aggregation attribute of each tuple with an order-preserving encryption scheme with the secret key K.

If the relation R is made of N tuples, then the complexity of the preprocessing algorithm is equal to $\mathcal{O}((\mathcal{C}_F + 2 \cdot \mathcal{C}_\mathcal{E} + \mathcal{C}_\mathsf{E}) \cdot N)$, where $\mathcal{C}_F$ (resp. $\mathcal{C}_\mathcal{E}$, $\mathcal{C}_\mathsf{E}$) is the cost of pseudo-random evaluation (resp. additive-homomorphic encryption, order-preserving encryption). In order to illustrate computation over encrypted data, we consider in the following the Paillier's cryptosystem (cf. Section 2.4.4 on page 23) for the additive-homomorphic encryption scheme Π.

In the preprocessing algorithm presented in Figure 6.8, we compute all encryptions that are required for the five aggregation operations. Indeed, we can assume that the MapReduce's user is allowed to perform these five aggregation operations on the considered relation. However, if the data owner knows the aggregation operation that will be queried, then the preprocessing algorithm can be adapted. For the COUNT operation, only pseudo-random evaluations and encryption of the grouping attribute values are required. For the SUM and AVG operations, we also need to compute the encryption of the aggregation attribute values using an additive-homomorphic encryption scheme. Finally, the MIN and MAX operations require all encryptions and the preprocessing corresponds to the one presented in Figure 6.8.

6.3.2 Secure MapReduce Grouping and Aggregation with COUNT Operation

We denote by $\text{SGA}_{\text{COUNT}}$ our protocol that computes the secure MapReduce grouping and aggregation with COUNT operation. The Map and Reduce functions for the $\text{SGA}_{\text{COUNT}}$ protocol are presented in Figure 6.9.

- *The Map Function.* For each tuple of the relation R^* sent by the data owner to the public cloud, the Map function keeps the key and sends only the Paillier encryption of the grouping attribute as value of the key-value pair.

- *The Reduce Function.* The Reduce function works as for the no-secure protocol and computes the count of tuples sharing the same value of the grouping attribute. An inherent limitation of the $\text{SGA}_{\text{COUNT}}$ protocol is that the public cloud always can count the number of loop applied in the Reduce function, hence we do not encrypt values 1. Moreover, the preprocessing algorithm keeps the repetitions of the relation. Since we assume that sets of nodes of the public cloud collude, the count value can also be determined. However, we stress that value of the grouping attribute associated to a count remains unkown to the public cloud. Indeed, it remains protected through the Paillier's cryptosystem.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R^*
// *value*: collection of $(F(k, \pi_A(t)), (\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)), \mathsf{E}(K, \pi_B(t)))) \in R^*$
// with $t \in R$
foreach $(\texttt{key}, (\texttt{val}_1, \texttt{val}_2, \texttt{val}_3)) \in R^*$ **do**
 | $\texttt{emit}_{\mathcal{R}_1 \to \mathcal{R}2}(\texttt{key}, \texttt{val}_1)$

Reduce function:

Input: $(key, values)$
// *key*: $F(k, \pi_A(t))$ with $t \in R$
// *values*: collection of $\mathcal{E}(pk, \pi_A(t))$ with $t \in R$
$\mathsf{count} := 0$
foreach $\mathcal{E}(pk, \pi_A(t)) \in values$ **do**
 | $\mathsf{count} := \mathsf{count} + 1$
$\texttt{emit}_{\mathcal{R}_2 \to \mathcal{U}}(\mathcal{E}(pk, \pi_A(t)), \mathsf{count})$

Figure 6.9: Map and Reduce functions for the $\text{SGA}_{\text{COUNT}}$ protocol.

6.3.3 Secure MapReduce Grouping and Aggregation with SUM Operation

We present the Map and Reduce functions of our secure MapReduce grouping and aggregation protocol for the SUM operation in Figure 6.10. This protocol is called SGA_{SUM} for *Secure Grouping Aggregation*.

- *The Map Function.* For each tuple of the relation R^* sent by the data owner to the public cloud, the Map function produces the key-value pair where the key is the pseudo-random evaluation of the grouping attribute and value is the couple made of the Paillier encryption of the grouping attribute value and the Paillier encryption of aggregation attribute value allowing to compute the sum.
- *The Reduce Function.* Due to the homomorphic property of the Paillier's cryptosystem, the Reduce function uses all Paillier encryptions of the aggregation attribute values to compute the sum associated to the grouping attribute value.

6.3.4 Secure MapReduce Grouping and Aggregation with AVG Operation

We present the Map and Reduce functions of our secure MapReduce grouping and aggregation protocol for the AVG operation in Figure 6.11. This protocol is called SGA_{AVG} for *Secure Grouping Aggregation*.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R^*
// *value*: collection of $(F(k, \pi_A(t)), (\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)), \mathsf{E}(K, \pi_B(t)))) \in R^*$
// with $t \in R$
foreach $(\texttt{key}, (\texttt{val}_1, \texttt{val}_2, \texttt{val}_3)) \in R^*$ **do**
| $\texttt{emit}_{\mathcal{R}_1 \to \mathcal{R}2}(\texttt{key}, (\texttt{val}_1, \texttt{val}_2))$

Reduce function:

Input: $(key, values)$
// *key*: $F(k, \pi_A(t))$ with $t \in R$
// *values*: collection of $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)))$ with $t \in R$
$\mathsf{sum} := \mathcal{E}(pk, 0)$
foreach $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)) \in values$ **do**
| $\mathsf{sum} := \mathsf{sum} \cdot \mathcal{E}(pk, \pi_B(t))$
$\texttt{emit}_{\mathcal{R}_2 \to \mathcal{U}}(\mathcal{E}(pk, \pi_A(t)), \mathsf{sum})$

Figure 6.10: Map and Reduce functions for the SGA_{SUM} protocol.

- *The Map Function.* It works exactly as the Map function of the SGA_{SUM} protocol.

- *The Reduce Function.* It combines the Reduce function of the $\text{SGA}_{\text{COUNT}}$ and the SGA_{SUM} protocols. For each considered key, the Reduce function emits a triplet with the asymmmetric encryption of the grouping attribute (corresponding to the current key value), the asymmetric encryption of the sum of the aggregation attribute values, and the count. In this way, the MapReduce's user is allowed to compute the average.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R^*
// *value*: collection of $(F(k, \pi_A(t)), (\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)), \mathsf{E}(K, \pi_B(t)))) \in R^*$
// with $t \in R$
foreach $(\texttt{key}, (\texttt{val}_1, \texttt{val}_2, \texttt{val}_3)) \in R^*$ **do**
| $\texttt{emit}_{\mathcal{R}_1 \to \mathcal{R}2}(\texttt{key}, (\texttt{val}_1, \texttt{val}_2))$

Reduce function:

Input: $(key, values)$
// *key*: $F(k, \pi_A(t))$ with $t \in R$
// *values*: collection of $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)))$ with $t \in R$
$\mathsf{count} := 0$
$\mathsf{sum} := \mathcal{E}(pk, 0)$
foreach $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)) \in values$ **do**
| $\mathsf{count} := \mathsf{count} + 1$
| $\mathsf{sum} := \mathsf{sum} \cdot \mathcal{E}(pk, \pi_B(t))$
$\texttt{emit}_{\mathcal{R}_2 \to \mathcal{U}}(\mathcal{E}(pk, \pi_A(t)), \mathsf{sum}, \mathsf{count})$

Figure 6.11: Map and Reduce functions for the SGA_{AVG} protocol.

6.3.5 Secure MapReduce Grouping and Aggregation with MIN Operation

We present the Map and Reduce functions of our secure MapReduce grouping and aggregation protocol for the MIN operation in Figure 6.12. This protocol is called SGA_{MIN} for *Secure Grouping Aggregation*.

- *The Map Function.* It produces key-value pairs such that value of pairs contains the asymmetric encryption of the grouping attribute value, the asymmetric encryption and the order-preserving symmetric encryption of the aggregation attribute value.

- *The Reduce Function.* Using the property of order-preserving symmetric encryption, each reducer computes the minimum value associated to the considered value of the grouping attribute value. Since the MapReduce's user does not know the secret key K used with the order-preserving symmetric encryption scheme, she uses asymmetric encryption in order to know the value of the minimum.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R^*
// *value*: collection of $(F(k, \pi_A(t)), (\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)), \mathsf{E}(K, \pi_B(t)))) \in R^*$
// with $t \in R$
foreach $(\texttt{key}, (\texttt{val}_1, \texttt{val}_2, \texttt{val}_3)) \in R^*$ **do**
  $\texttt{emit}_{\mathcal{R}_1 \to \mathcal{R}2}(\texttt{key}, (\texttt{val}_1, \texttt{val}_2, \texttt{val}_3))$

Reduce function:

Input: $(key, values)$
// *key*: $F(k, \pi_A(t))$ with $t \in R$
// *values*: collection of $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)), \mathsf{E}(K, \pi_B(t)))$ with $t \in R$
$\texttt{min} \xleftarrow{\$} \{\texttt{val}_3 : (\texttt{val}_1, \texttt{val}_2, \texttt{val}_3) \in values\}$
foreach $(\texttt{val}_1, \texttt{val}_2, \texttt{val}_3) \in values$ **do**
  if $\texttt{val}_3 < \texttt{min}$ **then**
    $\texttt{encA} := \texttt{val}_1$
    $\texttt{encB} := \texttt{val}_2$
    $\texttt{min} := \texttt{val}_3$
$\texttt{emit}_{\mathcal{R}_2 \to \mathcal{U}}(\texttt{encA}, \texttt{encB}, \texttt{min})$

Figure 6.12: Map and Reduce functions for the SGA_{MIN} protocol.

6.3.6 Proof of Correctness

We say that the protocol $\text{SGA}_{\text{COUNT}}$ (resp. $\text{SGA}_{\text{SUM}}, \text{SGA}_{\text{AVG}}$, MIN, MAX) is *correct* if it returns the correct count (resp. sum, average, minimum, maximum) for each value of the grouping attribute.

Lemma 28. *Assume that the pseudo-random function family F perfectly emulates a random oracle, then protocol SGA_{COUNT} (resp. SGA_{SUM}, SGA_{AVG}, SGA_{MIN}, SGA_{MAX}) is correct.*

Proof. Let R be a relation, and R^* be the corresponding protected relation computed by the preprocessing algorithm presented in Figure 6.8.

For each $t \in R$, there exists a key-value pair in relation R^* of the form

$$(F(k, \pi_A(t)), (\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)), \mathsf{E}(K, \pi_B(t)))) \ .$$

Since the pseudo-random function F perfectly emulates a random oracle, tuples sharing the same value of grouping attribute are sent to the same reducer. Indeed, we recall that F is a deterministic algorithm. Hence, each reducer associated to a key $F(k, \pi_A(t))$ with $t \in R$ has a collection of values of the form

- $(\mathcal{E}(pk, \pi_A(t)), 1)$ for the COUNT operation,
- $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)))$ for the SUM and AVG operations,
- $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)), \mathsf{E}(K, \pi_B(t)))$ for the MIN and MAX operations.

For the COUNT operation, the cardinal number of the collection corresponds to the result of the COUNT operation of the considered key. For the SUM operation, the reducer uses the additive-homomorphic property of the Paillier's cryptosystem to compute the result of the SUM operation of the considered key. Concerning the AVG operation, the reducer computes the cardinal number of the collection, and uses the additive-homomorphic property of the Paillier's cryptosystem to compute the sum of aggregation attribute values associated to the considered key allowing the MapReduce's user to compute the average. Finally for the MIN and MAX operations, the reducer associated to the considered key uses the property of the order-preserving encryption scheme to determine the minimum or the maximum of the aggregation attribute value.

This concludes the proof. □

6.3.7 Complexity of Originals and SGA Protocols

We summarize in Table 6.1 the computation cost and the communication cost for original protocols [LRU14, Chapter 2] and our SGA protocols. In our communication cost analysis, we measure the total size of the data that is emitted from a map or reduce node, and quantify by 1 the transmission of an attribute value of a tuple.

We remark that in each case, our SGA protocols are efficient from both computational and communication points of view, in the sense that the overhead is linear for each of the two complexity measures.

Table 6.1: Complexity of original MapReduce grouping and aggregation protocols and of our SGA protocols. Let N_1 be the number of tuples of the relation R. Let $\mathcal{C}_+$ (resp. $\mathcal{C}_{\times_{\mathsf{mod}}}$, $\mathcal{C}_{\mathcal{E}}$, $\mathcal{C}_{\div}$, $\mathcal{C}_{\mathsf{comp}}$) be the cost of addition (resp. modular multiplication, asymmetric encryption, division, comparison).

Operation	Protocol	Computation cost (big-$\mathcal{O}$)	Communication cost (big-$\mathcal{O}$)
COUNT	Original	$\mathcal{C}_+ \cdot N_1$	$4 \cdot N_1$
	SGA	$\mathcal{C}_+ \cdot N_1$	$4 \cdot N_1$
SUM	Original	$\mathcal{C}_+ \cdot N_1$	$4 \cdot N_1$
	SGA	$(\mathcal{C}_{\times_{\mathsf{mod}}} + \mathcal{C}_{\mathcal{E}}) \cdot N_1$	$5 \cdot N_1$
AVG	Original	$(2 \cdot \mathcal{C}_+ + \mathcal{C}_{\div}) \cdot N_1$	$4 \cdot N_1$
	SGA	$(\mathcal{C}_+ + \mathcal{C}_{\mathcal{E}}) \cdot N_1$	$6 \cdot N_1$
MIN / MAX	Original	$\mathcal{C}_{\mathsf{comp}} \cdot N_1$	$4 \cdot N_1$
	SGA	$\mathcal{C}_{\mathsf{comp}} \cdot N_1$	$7 \cdot N_1$

6.4 Experimental Results

We present experimental results for the original MapReduce protocols that compute the grouping and aggregation with the COUNT, SUM, AVG, MIN operations [LRU14, Chapter 2] and for our corresponding SGA protocols. We do not give results for the MAX operation since they are the same as the MIN operation.

6.4.1 Dataset and Settings

We use the real-world *MovieLens 10M Dataset* [HK16] which contains $10,000,054$ of anonymous ratings of movies made by MovieLens users. There are $10,681$ different movies, and $71,567$ different users. All users had rated at least 20 movies. We denote the dataset $R(A, B)$ where attribute A is the *movie ID* and attribute B is the *rating*. A movie ID is an integer, and the ratings are made on a 5-star scale (whole-star ratings only). To perform grouping and aggregation operations, we first randomly pick samples of $R(A, B)$, ranging from $1,000,000$ to $10,000,000$ tuples in steps of $1,000,000$. We run all the protocols on these 10 relations.

According to our secure protocols, we use as pseudo-random function the HMAC-SHA256 keyed-hash message authentication code [BCK96] implemented in Go package `hmac`[†]. Moreover, we use the RSA-OAEP asymmetric encryption scheme [BR94, KS98] implemented in Go package `rsa`[‡] with a 1024-bit RSA modulus for encryption of grouping attribute values. Indeed, this encryption does not require additive-homomorphic property. However, encryption of aggregation attribute values needs the additive-homomorphic property for $\text{SGA}_{\text{COUNT}}$, SGA_{SUM}, and SGA_{AVG} protocols, hence we use a Go implementation of the Paillier's cryptosystem called Gaillier[§]. Note that Gaillier is not an optimized implementation. Hence, we use it with a 64-bit RSA modulus as proof of concept. Finally, SGA_{MIN} and SGA_{MAX} protocols are based on OPE schemes which are not yet implemented in Go language. Hence, we implemented a prototype of the *summation of random numbers* scheme [AKSX04].

We can note that, instead of using RSA-OAEP cryptosystem, we could use another asymmetric encryption scheme such as the ElGamal cryptosystem [Gam85]. However for the same security, ElGamal ciphertexts are twice longer compared to RSA-OAEP ciphertexts, hence we do not use it for communication considerations.

6.4.2 Results

We first report wall clock times of the preprocessing algorithm for the 10 relations we used in Table 6.2.

CPU times are presented in Figure 6.13. First, as expected by the complexity study 6.1, we observe that the CPU time for the four operations of original protocols are the same.

We remark that complexity of our secure protocols remain linear. The overhead compared to the original protocols is due to the size of the encrypted data. Moreover, we observe two tendencies for secure protocols. Indeed, our secure protocols for the SUM and AVG operations require an encryption for each grouping attribute value in order to compute the encrypted sum of the aggregation attribute values, while operations COUNT and MIN do not need such encryption.

[†] https://golang.org/pkg/crypto/hmac/
[‡] https://golang.org/pkg/crypto/rsa/
[§] https://github.com/actuallyachraf/gomorph

Table 6.2: Wall clock times of the secure protocols preprocessing.

Number of tuples / relation	Wall clock time
1,000,000	63.526 s
2,000,000	130.162 s
3,000,000	196.391 s
4,000,000	263.213 s
5,000,000	326.961 s
6,000,000	390.763 s
7,000,000	449.211 s
8,000,000	524.712 s
9,000,000	575.996 s
10,000,000	644.523 s

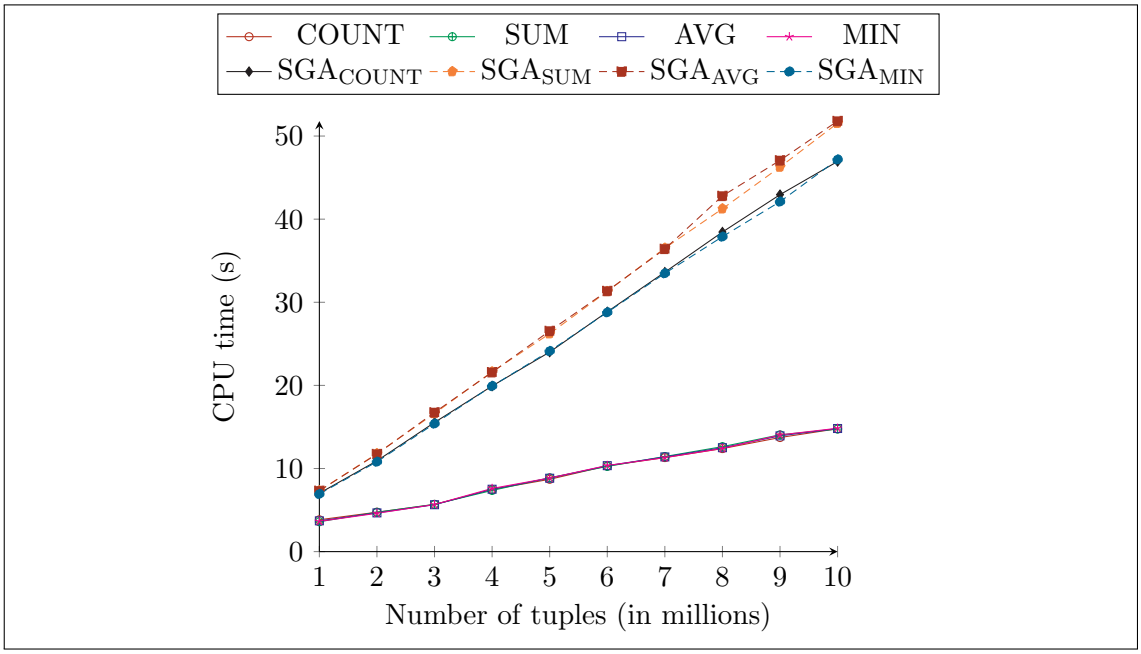


Figure 6.13: CPU time vs the number of tuples for no-secure and secure protocols which perform grouping and aggregation with the COUNT, SUM, AVG, and MIN operations.

6.5 Security Proof

We now give the security proof of our protocol SGA_{SUM} (respectively SGA_{MIN}) that securely computes the grouping and aggregation with the SUM operation (respectively with the MIN operation). Security proofs for $\text{SGA}_{\text{COUNT}}$ and SGA_{AVG} protocols are similar to the security proof of SGA_{SUM} protocol, and security proof for SGA_{MAX} protocols is similar to the security proof of SGA_{MIN} protocol so we omit them. In the following, we recall that we denote by A the grouping attribute, and we denote by B the aggregation attribute.

We first give the proof for the SGA_{SUM} protocol in Theorem 9. Then, we give the security proof for the SGA_{MIN} protocol in Theorem 10.

6.5.1 Modeling of SGA_{SUM} Protocol

We start by modeling our SGA_{SUM} protocol. The data owner owning the relation R is denoted $P_{\mathcal{R}}$. The public cloud is denoted $P_{\mathcal{C}}$, and the user is denoted $P_{\mathcal{U}}$. Parties use

respective inputs $I := (I_{\mathcal{R}}, I_{\mathcal{C}}, I_{\mathcal{U}})$ and a function $g := (g_{\mathcal{R}}, g_{\mathcal{C}}, g_{\mathcal{U}})$ that computes the grouping and aggregation with the SUM operation. We describe these three parties as follows

- $P_{\mathcal{R}}$ has the input $I_{\mathcal{R}} := (pk, k, R)$ where pk is a public key of the Paillier's cryptosystem, $k \in \mathcal{K}$ is a secret key for the pseudo-random function family F, and R is a relation such that $|R| = N_1$. The party $\mathcal{R}$ outputs $g_{\mathcal{R}}(I) := \perp$ (where $\perp$ denotes that the function returns nothing) since it does not learn any information.

- $P_{\mathcal{C}}$ has the input $I_{\mathcal{C}} := pk$ where pk is a public key of the Paillier's cryptosystem. It returns $g_{\mathcal{C}}(I) := (N_1, N_2, \tau)$ where N_1 is the number of tuples in relation R, and N_2 is the number of tuples in the result of the query $\gamma_{A,\mathrm{SUM}(B)}(R)$, i.e., the number of distinct values for the grouping attribute. Moreover, since a pseudo-random function is deterministic, the public cloud learns the occurrence for each value of the grouping attribute, i.e., $\tau := \{(i, \rho_i)\}_{i \in [\![1, N_2]\!]}$, where ρ_i is the occurrence of the $\pi_A(t_i)$ grouping attribute value such that $t_i \in R$.

- $P_{\mathcal{U}}$ has the input $I_{\mathcal{U}} := (pk, sk)$ where (pk, sk) is a key pair of the Paillier's cryptosystem. It returns $g_{\mathcal{U}}(I) := \gamma_{A,\mathrm{SUM}(B)}(R)$, i.e., the result of grouping and aggregation on relation R with the SUM operation computed by the public cloud.

6.5.2 Security Proof of $\mathrm{SGA}_{\mathrm{SUM}}$ Protocol

We give the security proof of our $\mathrm{SGA}_{\mathrm{SUM}}$ protocol in Theorem 9.

Theorem 9. *Assume F is a secure pseudo-random function family and that Paillier's cryptosystem is an IND-CPA asymmetric encryption scheme, then the SGA_{SUM} protocol securely computes the grouping and aggregation with the SUM operation in the presence of semi-honest adversaries if parties $P_{\mathcal{C}}$ and $P_{\mathcal{U}}$ do not collude.*

The security proof for Theorem 9 is decomposed in Lemma 29 for the party $P_{\mathcal{R}}$, in Lemma 30 for the party $P_{\mathcal{C}}$, and in Lemma 31 for the party $P_{\mathcal{U}}$.

Lemma 29. *There exists probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{R}}^{SGA_{SUM}}$ such that*

$$\left\{\mathcal{S}_{\mathcal{R}}^{SGA_{SUM}}(1^{\lambda}, I_{\mathcal{R}}, g_{\mathcal{R}}(I))\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}} \overset{\mathrm{c}}{\equiv} \left\{\mathsf{view}_{\mathcal{R}}^{SGA_{SUM}}(I, \lambda)\right\}_{I \in \mathcal{I}, \lambda \in \mathbb{N}}.$$

Proof. Consider that $\mathcal{R}$ is corrupted, we observe that $P_{\mathcal{R}}$ receives no output and no incoming message from other parties. Thus, we merely need to show that a simulator can generate the view of party $P_{\mathcal{R}}$ from its inputs. For the sake of simplicity, we assume that $P_{\mathcal{R}}$ runs the preprocessing algorithm without considering encryption of the aggregation attribute values and the order-preserving encryption since they are useless.

Hence, $P_{\mathcal{R}}$ receives the public key pk of the user, the secret key k used for the pseudo-random function family F, and the relation R. Simulator $\mathcal{S}_{\mathcal{R}}^{\mathrm{SGA}_{\mathrm{SUM}}}$ runs the preprocessing algorithm 6.8 to obtain the encrypted relation R^* associated to the relation R.

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

Lemma 30. *If F is a secure pseudo-random function family and Paillier's cryptosystem is an IND-CPA asymmetric encryption scheme, then there exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{C}}^{SGA_{SUM}}$ such that*

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

Proof. Let $\lambda \in \mathbb{N}$ be a security parameter. Before to build $\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{SUM}}}$ that computes a distribution that can be simulated perfectly, we use the hybrid argument to build an hybrid simulator denoted $\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}$. Simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}$ works as SGA_{SUM} protocol but each evaluation of the pseudo-random function is substituted using the random oracle OPRF presented in Figure 6.14.

$\mathsf{OPRF}(x)$:
if $T[x] = \emptyset$ **then**
  $T[x] \xleftarrow{\$} \{0,1\}^{|\mathcal{Y}|}$
return $T[x]$.

Figure 6.14: Random oracle OPRF.

We start by presenting the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}$ in Figure 6.15.

Simulator: $\mathcal{S}_{\mathcal{C}}^{H-\text{SGA}_{\text{SUM}}}(1^{\lambda}, pk, (N_1, N_2, \tau))$
$R^* := \emptyset$
$R^*_{\gamma} := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t'_j \xleftarrow{\$} B$
    $R^* := R^* \cup \{(\mathsf{OPRF}(t), \mathcal{E}(pk, t'_j))\}$
  $R^*_{\gamma} := R^*_{\gamma} \cup \{(\mathsf{OPRF}(t), \mathcal{E}(pk, \sum_{j=1}^{\rho_i} t'_j))\}$
$\mathsf{view} := (R^*, R^*_{\gamma})$

Figure 6.15: Simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}$ for the proof of Lemma 30.

Assume by contradiction that there exists a non-uniform probabilistic-polynomial time distinguisher D such that for all inputs I, we have

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

where μ is a non-negligible function in λ.

We construct a non-uniform probabilistic-polynomial time guessing algorithm $\mathcal{A}$ that uses D to win PRF experiment $\mathsf{Exp}_{F,\mathcal{A}}^{\text{prf-}b}$ with $b \in \{0,1\}$ against the pseudo-random function family F. Adversary $\mathcal{A}$ is presented in Figure 6.16.

First, we remark that

$$\Pr[\mathsf{Exp}_{F,\mathcal{A}}^{\text{prf-0}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}(1^{\lambda}, I_{\mathcal{C}}, g_{\mathcal{C}}(I))) = 1] \ .$$

Indeed, when $b = 0$ the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}$. Then the probability that the experiment PRF experiment returns 1 is equal to the probability that the distinguisher D returns 1 on input computed by the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}$. On the other hand, we have

$$\Pr[\mathsf{Exp}_{F,\mathcal{A}}^{\text{prf-1}}(\lambda) = 1] = \Pr[D(\mathsf{view}_{\mathcal{C}}^{\text{SGA}_{\text{SUM}}}(I, \lambda)) = 1] \ .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol SGA_{SUM}. Then the probability that the PRF experiment returns 1 is equal to the probability that the distinguisher D returns 1 on input computed as in the real protocol.

Adversary: $\mathcal{A}(pk)$

$R^* := \emptyset$
$R^*_\gamma := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t'_j \xleftarrow{\$} B$
    $R^* := R^* \cup \{(f_b(t), \mathcal{E}(pk, t'_j))\}$
  $R^*_\gamma := R^*_\gamma \cup \{(\mathsf{OPRF}(t), \mathcal{E}(pk, \sum_{j=1}^{\rho_i} t'_j))\}$
$\mathsf{view} := (R^*, R^*_\gamma)$
$b_* := D(\mathsf{view})$
return b_*

Figure 6.16: Adversary $\mathcal{A}$ for the proof of Lemma 30.

It therefore follows that

$$\begin{aligned}\mathsf{Adv}^{\mathrm{prf}}_{F,\mathcal{A}}(\lambda) &= \left|\Pr[\mathsf{Exp}^{\text{prf-1}}_{F,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{prf-0}}_{F,\mathcal{A}}(\lambda) = 1]\right| \\ &= \left|\Pr[D(\mathsf{view}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}(I, \lambda)) = 1] - \Pr[D(\mathcal{S}^{H\text{-}\mathrm{SGA_{SUM}}}_{\mathcal{C}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1]\right| \\ &= \mu(\lambda)\ ,\end{aligned}$$

which is non-negligible. However, we assume that F is a secure pseudo-random function, hence, it does not exist D such that

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

is non-negligible. Hence, we have

$$\mathsf{view}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}(I, \lambda) \stackrel{\mathrm{c}}{\equiv} \mathcal{S}^{H\text{-}\mathrm{SGA_{SUM}}}_{\mathcal{C}}(I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I, \lambda))\ .$$

Finally, we show of to build the simulator $\mathcal{S}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}$. The difference between $\mathcal{S}^{H\text{-}\mathrm{SGA_{SUM}}}_{\mathcal{C}}$ and $\mathcal{S}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}$ is that $\mathcal{S}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}$ substitutes Paillier encryption of real values by Paillier encryption of random values of the same size. More formally, $\mathcal{S}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}$ is presented in Figure 6.17.

Simulator: $\mathcal{S}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}(1^\lambda, pk, (N_1, N_2, \tau))$

$R^* := \emptyset$
$R^*_\gamma := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t' \xleftarrow{\$} B$
    $R^* := R^* \cup \{(\mathsf{OPRF}(t), \mathcal{E}(pk, t'))\}$
  $t' \xleftarrow{\$} B$
  $R^*_\gamma := R^*_\gamma \cup \{(\mathsf{OPRF}(t), \mathcal{E}(pk, t'))\}$
$\mathsf{view} := (R^*, R^*_\gamma)$

Figure 6.17: Simulator $\mathcal{S}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}$ for the proof of Lemma 30.

Now we show that we have

$$\mathcal{S}^{H\text{-}\mathrm{SGA_{SUM}}}_{\mathcal{C}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \stackrel{\mathrm{c}}{\equiv} \mathcal{S}^{\mathrm{SGA_{SUM}}}_{\mathcal{C}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))\ .$$

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

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

where μ is a non-negligible function in λ.

We construct a non-uniform probabilistic-polynomial time guessing algorithm $\mathcal{B}$ that uses D to win the IND-CPA experiment $\mathsf{Exp}^{\text{indcpa-}b}_{\text{Paillier},\mathcal{B}}$ for $b \in \{0,1\}$ against Paillier cryptosystem. Adversary $\mathcal{B}$ is presented in Figure 6.18.

Adversary: $\mathcal{B}(pk)$

$R^* := \emptyset$
$R^*_\gamma := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t'_j \xleftarrow{\$} B$
    $R^* := R^* \cup \{(\mathsf{OPRF}(t), \mathcal{E}(pk, t'_j))\}$
  $t' \xleftarrow{\$} B$
  $R^*_\gamma := R^*_\gamma \cup \{(\mathsf{OPRF}(t), \mathcal{E}(pk, \mathsf{LoR}_b(\sum_{j=1}^{\rho_i} t'_j, t')))\}$
$\mathsf{view} := (R^*, R^*_\gamma)$
$b_* := D(\mathsf{view})$
return b_*

Figure 6.18: Adversary $\mathcal{B}$ for the proof of Lemma 30.

First, we remark that

$$\Pr[\mathsf{Exp}^{\text{ind-cpa-0}}_{\text{Paillier},\mathcal{B}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{SUM}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \ .$$

When $b = 0$, the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{SUM}}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{SUM}}}$. On the other hand, we have

$$\Pr[\mathsf{Exp}^{\text{ind-cpa-1}}_{\text{Paillier},\mathcal{B}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \ .$$

Indeed, when $b = 1$ the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}$.

Finally, we evaluate the probability that $\mathcal{B}$ wins the experiment

$$\begin{aligned}
\mathsf{Adv}^{\text{indcpa}}_{\text{Paillier},\mathcal{B}}(\lambda) &= \left| \Pr[\mathsf{Exp}^{\text{ind-cpa-1}}_{\text{Paillier},\mathcal{B}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{ind-cpa-0}}_{\text{Paillier},\mathcal{B}}(\lambda) = 1] \right| \\
&= \Big| \Pr[D(\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \\
&\quad - \Pr[D(\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{SUM}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \Big| \\
&= \mu(\lambda) \ ,
\end{aligned}$$

which is non-negligible. However, we assume that Paillier is IND-CPA. Hence, we have

$$\mathcal{S}_{\mathcal{C}}^{H\text{-SGA}_{\text{SUM}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \overset{\text{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{SUM}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \ .$$

By transitivity, we have

$$\mathsf{view}_{\mathcal{C}}^{\mathrm{SGA_{SUM}}}(I,\lambda) \stackrel{\mathrm{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{\mathrm{SGA_{SUM}}}(1^{\lambda}, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \ .$$

which concludes the proof. □

Lemma 31. *There exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{U}}^{SGA_{SUM}}$ *such that*

$$\left\{\mathcal{S}_{\mathcal{U}}^{SGA_{SUM}}(1^{\lambda}, I_{\mathcal{R}}, g_{\mathcal{R}}(I))\right\}_{\lambda\in\mathbb{N}, I\in\mathcal{I}} \stackrel{\mathrm{c}}{\equiv} \left\{\mathsf{view}_{\mathcal{U}}^{SGA_{SUM}}(I,\lambda)\right\}_{\lambda\in\mathbb{N}, I\in\mathcal{I}} .$$

Proof. The view of $\mathcal{U}$ contains the relation $\gamma_{A,\mathrm{SUM}(B)}(R)$. For each pair $t \in \gamma_{A,\mathrm{SUM}(B)}(R)$, $\mathcal{S}_{\mathcal{U}}$ uses the public key pk and produces the pair $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)))$ constituting the result sent by the public cloud.

Hence, $\mathcal{S}_{\mathcal{U}}((sk, pk), \gamma_{A,\mathrm{SUM}(B)}(R))$ describes exactly the same distribution as, which concludes the proof. □

We now give the security proof of our secure protocol $\mathrm{SGA_{MIN}}$ in Theorem 10.

6.5.3 Modeling of $\mathrm{SGA_{MIN}}$ Protocol

We start by modeling our $\mathrm{SGA_{MIN}}$ protocol that computes the grouping and aggregation with the MIN operation. The data owner, the public cloud, and the user use respective inputs $I := (I_{\mathcal{R}}, I_{\mathcal{C}}, I_{\mathcal{U}})$ and a function $g := (g_{\mathcal{R}}, g_{\mathcal{C}}, g_{\mathcal{U}})$. We describe the three parties as follows:

- $\mathcal{R}$ has the input $I_{\mathcal{R}} := (pk, k, K, R)$ where pk is a public key of an asymmetric encryption scheme Π, $k \in \mathcal{K}$ is a secret key for the pseudo-random function family F, K is a secret key for the order-preserving symmetric encryption scheme Π^{ope}, and R is a relation owned by $\mathcal{R}$. The party $\mathcal{R}$ outputs $g_{\mathcal{R}}(I) = \perp$ (where $\perp$ denotes that the function returns nothing) since $\mathcal{R}$ does not learn any information.

- $\mathcal{C}$ has the input $I_{\mathcal{C}} := pk$ where pk is a public key of a asymmetric encryption scheme Π. It returns $g_{\mathcal{C}}(I) = (N_1, N_2, \tau)$ where N_1 is the number of tuples in relation R, and N_2 is the number of tuples in $\gamma_{A,\mathrm{MIN}(B)}(R)$, i.e., the number of distinct values for the grouping attribute. Moreover, since a pseudo-random function is deterministic, the public cloud learns the occurrence for each value of the grouping attribute, i.e., $\tau := \{(i, \rho_i)\}_{i\in[\![1,N_2]\!]}$, where ρ_i is the occurrence of the $\pi_A(t_i)$ grouping attribute value such that $t_i \in R$.

- $\mathcal{U}$ has the input $I_{\mathcal{U}} := (pk, sk)$ where (pk, sk) is a key pair of an asymmetric encryption scheme Π. It returns $g_{\mathcal{U}}(I) := \gamma_{A,\mathrm{MIN}(B)}(R)$, i.e., the result of grouping and aggregation on relation R with the MIN operation computed by the public cloud.

6.5.4 Security Proof of $\mathrm{SGA_{MIN}}$ Protocol

We give the security proof of our $\mathrm{SGA_{MIN}}$ protocol in Theorem 10.

Theorem 10. *Assume F is a secure pseudo-random function family, Π is an IND-CPA asymmetric encryption scheme, and Π^{ope} is an IND-OCPA order-preserving symmetric encryption scheme, then the SGA_{MIN} protocol securely computes the grouping and aggregation with the MIN operation in the presence of semi-honest adversaries if parties $\mathcal{C}$ and $\mathcal{U}$ do not collude.*

The security proof for Theorem 10 is decomposed in Lemma 32 for the party $\mathcal{R}$, in Lemma 33 for the party $\mathcal{C}$, and in Lemma 34 for the party $\mathcal{U}$.

Lemma 32. *Let $\lambda \in \mathbb{N}$ be a security parameter. There exists probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{R}}^{SGA_{MIN}}$ such for all inputs $I = (I_{\mathcal{R}}, I_{\mathcal{C}}, I_{\mathcal{U}})$, we have*

$$\mathcal{S}_{\mathcal{R}}^{SGA_{MIN}}(1^{\lambda}, I_{\mathcal{R}}, g_{\mathcal{R}}(I)) \stackrel{c}{\equiv} \mathsf{view}_{\mathcal{R}}^{SGA_{MIN}}(I, \lambda) \ .$$

Proof. Consider that $\mathcal{R}$ is corrupted, we observe that $\mathcal{R}$ receives no output and no incoming message from other parties. Thus, we merely need to show that a simulator can generate the view of party $\mathcal{R}$ from its inputs.

In the protocol $\mathrm{SGA}_{\mathrm{MIN}}$, $\mathcal{R}$ receives the public key pk of the user, the secret key k used with the pseudo-random function family F, the secret key K used with the order-preserving symmetric encryption scheme Π^{ope}, and the relation R. Formally, $\mathcal{S}_{\mathcal{R}}^{\mathrm{SGA}_{\mathrm{MIN}}}$ is given (pk, k, K, R), and runs the preprocessing phase 6.8 on input (pk, k, K, R). $\mathcal{S}_{\mathcal{R}}^{\mathrm{SGA}_{\mathrm{MIN}}}$ obtains the encrypted relation R^* associated to the relation R.

We remark that $\mathcal{S}_{\mathcal{R}}^{\mathrm{SGA}_{\mathrm{MIN}}}$ uses exactly the same algorithm as the real protocol $\mathrm{SGA}_{\mathrm{MIN}}$, then it describes the same distribution as $\mathsf{view}_{\mathcal{R}}^{\mathrm{SGA}_{\mathrm{MIN}}}(I, \lambda)$, i.e., we have

$$\mathcal{S}_{\mathcal{R}}^{\mathrm{SGA}_{\mathrm{MIN}}}(1^{\lambda}, I_{\mathcal{R}}, g_{\mathcal{R}}(I)) \stackrel{c}{\equiv} \mathsf{view}_{\mathcal{R}}^{\mathrm{SGA}_{\mathrm{MIN}}}(I, \lambda) \ ,$$

which concludes the proof. □

Lemma 33. *If F is a secure pseudo-random function family, Π is an IND-CPA asymmetric encryption scheme, and Π^{ope} is an IND-OCPA order-preserving symmetric encryption scheme, then there exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{C}}^{SGA_{MIN}}$ such for all inputs $I = (I_{\mathcal{R}}, I_{\mathcal{C}}, I_{\mathcal{U}})$, we have*

$$\mathcal{S}_{\mathcal{C}}^{SGA_{MIN}}(1^{\lambda}, I_{\mathcal{C}}, g_{\mathcal{C}}(I)) \stackrel{c}{\equiv} \mathsf{view}_{\mathcal{C}}^{SGA_{MIN}}(I, \lambda) \ .$$

Proof. Let $\lambda \in \mathbb{N}$ be a security parameter. Before to build $\mathcal{S}_{\mathcal{C}}^{\mathrm{SGA}_{\mathrm{MIN}}}$ that computes a distribution that can be simulated perfectly, we use the hybrid argument to build hybrid simulators denoted $\mathcal{S}_{\mathcal{C}}^{H_1\text{-}\mathrm{SGA}_{\mathrm{MIN}}}$ and $\mathcal{S}_{\mathcal{C}}^{H_2\text{-}\mathrm{SGA}_{\mathrm{MIN}}}$. The simulator, $\mathcal{S}_{\mathcal{C}}^{H_1\text{-}\mathrm{SGA}_{\mathrm{MIN}}}$ works as $\mathrm{SGA}_{\mathrm{MIN}}$ but each evaluation of the pseudo-random function performed by the party $\mathcal{R}$ is substituted using the random oracle OPRF presented in Figure 6.14. We start by presenting the simulator $\mathcal{S}_{\mathcal{C}}^{H_1\text{-}\mathrm{SGA}_{\mathrm{MIN}}}$ in Figure 6.19.

Simulator: $\mathcal{S}_{\mathcal{C}}^{H_1\text{-}\mathrm{SGA}_{\mathrm{MIN}}}(1^{\lambda}, pk, (N_1, N_2, \tau))$

$R^* := \emptyset$
$R^*_{\gamma} := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t'^{(i)}_j \xleftarrow{\$} B$
    $R^* := R^* \cup \{(\mathsf{OPRF}(t), (\mathcal{E}(pk, t), \mathcal{E}(pk, t'^{(i)}_j), \mathsf{E}^{\mathsf{ope}}(K, t'^{(i)}_j)))\}$
  $R^*_{\gamma} := R^*_{\gamma} \cup \{(\mathsf{OPRF}(t), (\mathcal{E}(pk, t), \mathcal{E}(pk, \min_j t'^{(i)}_j), \mathsf{E}^{\mathsf{ope}}(K, \min_j t'^{(i)}_j)))\}$
$\mathsf{view} := (R^*, R^*_{\gamma})$

Figure 6.19: Simulator $\mathcal{S}_{\mathcal{C}}^{H_1\text{-}\mathrm{SGA}_{\mathrm{MIN}}}$ for the proof of Lemma 33.

Assume by contradiction that there exists a non-uniform probabilistic-polynomial time distinguisher D such that for all inputs I, we have

$$\Pr[D(\mathcal{S}_{\mathcal{C}}^{H_1\text{-}\mathrm{SGA}_{\mathrm{MIN}}}(1^{\lambda}, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] - \left|\Pr[D(\mathsf{view}_{\mathcal{C}}^{\mathrm{SGA}_{\mathrm{MIN}}}(I, \lambda)) = 1]\right| = \mu(\lambda) \ ,$$

where μ is a non-negligible function in λ.

We construct a non-uniform probabilistic-polynomial time guessing algorithm $\mathcal{A}$ that uses D to win the pseudo-random function distinguishing experiment $\mathsf{Exp}^{\text{prf-}b}_{F,\mathcal{A}}$ (with $b \in \{0,1\}$) against the pseudo-random function family F. Adversary $\mathcal{A}$ is presented in Figure 6.20.

Adversary: $\mathcal{A}(pk)$

$R^* := \emptyset$
$R^*_\gamma := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t'^{(i)}_j \xleftarrow{\$} B$
    $R^* := R^* \cup \{(f_b(t), (\mathcal{E}(pk, t), \mathcal{E}(pk, t'^{(i)}_j, \mathsf{E}^{\mathsf{ope}}(K, t'^{(i)}_j))))\}$
  $R^*_\gamma := R^*_\gamma \cup \{(\mathsf{OPRF}(t), (\mathcal{E}(pk, t), \mathcal{E}(pk, \min_j t'^{(i)}_j, \mathsf{E}^{\mathsf{ope}}(K, \min_j t'^{(i)}_j))))\}$
$\mathsf{view} := (R^*, R^*_\gamma)$
$b_* := D(\mathsf{view})$
return b_*

Figure 6.20: Adversary $\mathcal{A}$ for the proof of Lemma 33.

First, we remark that

$$\Pr[\mathsf{Exp}^{\text{prf-0}}_{F,\mathcal{A}}(\lambda) = 1] = \Pr[D(\mathsf{view}^{\text{SGA}_{\text{MIN}}}_{\mathcal{C}}(I, \lambda)) = 1] \ .$$

Indeed, when $b = 0$ the view that $\mathcal{A}$ uses as input for D is computed as in the real protocol SGA_{MIN}. Then the probability that the experiment $\mathsf{Exp}^{\text{prf-0}}_{F,\mathcal{A}}$ returns 1 is equal to the probability that the distinguisher D returns 1 on input computed by the real protocol. On the other hand, we have

$$\Pr[\mathsf{Exp}^{\text{prf-1}}_{F,\mathcal{A}}(\lambda) = 1] = \Pr[D(\mathcal{S}^{H_1\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}(1^\lambda, I_{\mathcal{C}}, g_{\mathcal{C}}(I))) = 1] \ .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}^{H_1\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}$. Then the probability that the experiment $\mathsf{Exp}^{\text{prf-1}}_{F,\mathcal{A}}$ returns 1 is equal to the probability that the distinguisher D returns 1 on input computed by the simulator $\mathcal{S}^{H_1\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}$. It therefore follows that

$$\begin{aligned}
\mathsf{Adv}^{\text{prf}}_{F,\mathcal{A}}(\lambda) &= \left|\Pr[\mathsf{Exp}^{\text{prf-1}}_{F,\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{prf-0}}_{F,\mathcal{A}}(\lambda) = 1]\right| \\
&= \left|\Pr[D(\mathcal{S}^{H_1\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] - \Pr[D(\mathsf{view}^{\text{SGA}_{\text{MIN}}}_{\mathcal{C}}(I, \lambda)) = 1]\right| \\
&= \mu(\lambda) \ ,
\end{aligned}$$

which is non-negligible. However, we assume that F is a secure pseudo-random function family, hence, it does not exist D such that

$$\left|\Pr[D(\mathsf{view}^{\text{SGA}_{\text{MIN}}}_{\mathcal{C}}(I, \lambda)) = 1] - \Pr[D(\mathcal{S}^{H_1\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1]\right|$$

is non-negligible. Hence, we have

$$\mathsf{view}^{\text{SGA}_{\text{MIN}}}_{\mathcal{C}}(I, \lambda) \overset{\text{c}}{\equiv} \mathcal{S}^{H_1\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}(I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I, \lambda)) \ .$$

We now build the simulator $\mathcal{S}^{H_2\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}$. The difference between $\mathcal{S}^{H_1\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}$ and $\mathcal{S}^{H_2\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}$ is that $\mathcal{S}^{H_2\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}$ substitutes asymmetric encryption of real values by asymmetric encryption of random values of the same size. We present more formally $\mathcal{S}^{H_2\text{-SGA}_{\text{MIN}}}_{\mathcal{C}}$ in Figure 6.21.

Simulator: $\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, pk, (N_1, N_2, \tau))$

$R^* := \emptyset$
$R^*_\gamma := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t'^{(i)}_j \xleftarrow{\$} B$
    $R^* := R^* \cup \{(\mathsf{OPRF}(t), (\mathcal{E}(pk, t), \mathcal{E}(pk, t'^{(i)}_j), \mathsf{E}^{\mathsf{ope}}(K, t'^{(i)}_j)))\}$
  $t_1 \xleftarrow{\$} B$
  $t_2 \xleftarrow{\$} B$
  $R^*_\gamma := R^*_\gamma \cup \{(\mathsf{OPRF}(t), (\mathcal{E}(pk, t_1), \mathcal{E}(pk, t_2), \mathsf{E}^{\mathsf{ope}}(K, \min_j t'^{(i)}_j)))\}$
$\mathsf{view} := (R^*, R^*_\gamma)$

Figure 6.21: Simulator $\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}$ for the proof of Lemma 33.

Now we show that we have

$$\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \overset{\text{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{H_1\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \ .$$

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

$$\left| \Pr[D(\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] - \Pr[D(\mathcal{S}_{\mathcal{C}}^{H_1\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \right| = \mu(\lambda) \ ,$$

where μ is a non-negligible function in λ.

We construct a non-uniform probabilistic-polynomial time guessing algorithm $\mathcal{B}$ that uses D to win the experiment $\mathsf{Exp}^{\text{indcpa-}b}_{\Pi,\mathcal{B}}$ for $b \in \{0, 1\}$ against the asymmetric encryption scheme Π. Algorithm $\mathcal{B}$ is presented in Figure 6.22.

Adversary: $\mathcal{B}(pk)$

$R^* := \emptyset$
$R^*_\gamma := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t'^{(i)}_j \xleftarrow{\$} B$
    $R^* := R^* \cup \{(\mathsf{OPRF}(t), (\mathcal{E}(pk, t), \mathcal{E}(pk, t'^{(i)}_j, \mathsf{E}^{\mathsf{ope}}(K, t'^{(i)}_j)))\}$
  $t_1 \xleftarrow{\$} B$
  $t_2 \xleftarrow{\$} B$
  $x_1 := \mathcal{E}(pk, \mathsf{LoR}_b(t, t_1))$
  $x_2 := \mathcal{E}(pk, \mathsf{LoR}_b(\min_j t'^{(i)}_j, t_2))$
  $R^*_\gamma := R^*_\gamma \cup \{(\mathsf{OPRF}(t), (x_1, x_2, \mathsf{E}^{\mathsf{ope}}(K, \min_j t'^{(i)}_j)))\}$
$\mathsf{view} := (R^*, R^*_\gamma)$
$b_* := D(\mathsf{view})$
return b_*

Figure 6.22: Adversary $\mathcal{B}$ for the proof of Lemma 33.

First, we remark that

$$\Pr[\mathsf{Exp}^{\text{indcpa-0}}_{\Pi,\mathcal{B}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{H_1\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \ .$$

When $b = 0$, the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H_1\text{-SGA}_{\text{MIN}}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H_1\text{-SGA}_{\text{MIN}}}$. On the other hand, we have

$$\Pr[\mathsf{Exp}_{\Pi,\mathcal{B}}^{\text{indcpa-1}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \ .$$

Indeed, when $b = 1$ the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}$.

Finally, we evaluate the probability that $\mathcal{B}$ wins the experiment

$$\begin{aligned}
\mathsf{Adv}_{\Pi,\mathcal{B}}^{\text{indcpa}}(\lambda) &= \big|\Pr[\mathsf{Exp}_{\Pi,\mathcal{B}}^{\text{indcpa-1}}(\lambda) = 1] - \Pr[\mathsf{Exp}_{\Pi,\mathcal{B}}^{\text{indcpa-0}}(\lambda) = 1]\big| \\
&= \big|\Pr[D(\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \\
&\quad - \Pr[D(\mathcal{S}_{\mathcal{C}}^{H_1\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1]\big| \\
&= \mu(\lambda) \ ,
\end{aligned}$$

which is non-negligible. However, we assume that Π is IND-CPA. Hence, we have:

$$\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \stackrel{\text{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{H_1\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \ .$$

Finally, we show how to build the simulator $\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{MIN}}}$ that perfectly simulates our SGA_{MIN} protocol. The difference between $\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}$ and $\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{MIN}}}$ is that $\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{MIN}}}$ substitutes order-preserving encryption of real values by order-preserving encryption of random values having the same distribution. However, values of $\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}$ are still picked randomly so the simulator $\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{MIN}}}$ is the same.

Now we show that we have

$$\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \stackrel{\text{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \ .$$

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

$$\big|\Pr[D(\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] - \Pr[D(\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1]\big| = \mu(\lambda) \ ,$$

where μ is a non-negligible function in λ.

We construct a non-uniform probabilistic-polynomial time guessing algorithm $\mathcal{D}$ that uses D to win the experiment $\mathsf{Exp}_{\Pi,\mathcal{D}}^{\text{indocpa-}b}$ for $b \in \{0, 1\}$ against the order-preserving symmetric encryption scheme Π^{ope}. Algorithm $\mathcal{D}$ works is presented in Figure 6.23.

First, we remark that

$$\Pr[\mathsf{Exp}_{\Pi^{\mathsf{ope}},\mathcal{D}}^{\text{indocpa-0}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \ .$$

When $b = 0$, the view that $\mathcal{D}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}$. Then the probability that the IND-OCPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H_2\text{-SGA}_{\text{MIN}}}$. On the other hand, we have

$$\Pr[\mathsf{Exp}_{\Pi^{\mathsf{ope}},\mathcal{D}}^{\text{indocpa-1}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \ .$$

Indeed, when $b = 1$ the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{\text{SGA}_{\text{MIN}}}$. Then the probability that the IND-OCPA experiment returns 1 is equal to the

Adversary: $\mathcal{D}(pk)$

$R^* := \emptyset$
$R^*_\gamma := \emptyset$
for $i \in [\![1, N_2]\!]$ **do**
  $t \xleftarrow{\$} A$
  $(t'^{(i)}_1, \ldots, t'^{(i)}_{\rho_i}) \xleftarrow{\$} B^{\rho_i}$ such that $t'^{(i)}_v \leq t'^{(i)}_w$ if $v \leq w$
  $(u'^{(i)}_1, \ldots, u'^{(i)}_{\rho_i}) \xleftarrow{\$} B^{\rho_i}$ such that $u'^{(i)}_v \leq u'^{(i)}_w$ if $v \leq w$
  for $j \in [\![1, \rho_i]\!]$ **do**
    $t^{(i)}_1 \xleftarrow{\$} A$
    $t^{(i)}_2 \xleftarrow{\$} B$
    $R^* := R^* \cup \{(\mathsf{OPRF}(t), (\mathcal{E}(pk, t^{(i)}_1), \mathcal{E}(pk, t^{(i)}_2), \mathsf{E}^{\mathsf{ope}}(K, \mathsf{LoR}_b(t'^{(i)}_j, u'^{(i)}_j))))\}$
  $R^*_\gamma := R^*_\gamma \cup \{R^*[1 + \sum_{j=1}^{i-1} \rho_j]\}$
$\mathsf{view} := (R^*, R^*_\gamma)$
$b_* := D(\mathsf{view})$
return b_*

Figure 6.23: Adversary $\mathcal{D}$ for the proof of Lemma 33.

probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{\mathrm{SGA}_{\mathrm{MIN}}}$.

Finally, we evaluate the probability that $\mathcal{D}$ wins the experiment

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{indocpa}}_{\Pi^{\mathsf{ope}},\mathcal{D}}(\lambda) &= \big|\Pr[\mathsf{Exp}^{\text{indocpa-1}}_{\Pi^{\mathsf{ope}},\mathcal{D}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\text{indocpa-0}}_{\Pi^{\mathsf{ope}},\mathcal{D}}(\lambda) = 1]\big| \\
&= \big|\Pr[D(\mathcal{S}_{\mathcal{C}}^{\mathrm{SGA}_{\mathrm{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \\
&\quad - \Pr[D(\mathcal{S}_{\mathcal{C}}^{H_2\text{-}\mathrm{SGA}_{\mathrm{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1]\big| \\
&= \mu(\lambda)\ ,
\end{aligned}$$

which is non-negligible. However, we assume that Π^{ope} is IND-OCPA. Hence, we have

$$\mathcal{S}_{\mathcal{C}}^{H_2\text{-}\mathrm{SGA}_{\mathrm{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \overset{\mathrm{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{\mathrm{SGA}_{\mathrm{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))\ .$$

By transitivity, we have

$$\mathsf{view}_{\mathcal{C}}^{\mathrm{SGA}_{\mathrm{MIN}}}(I, \lambda) \overset{\mathrm{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{\mathrm{SGA}_{\mathrm{MIN}}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))\ .$$

□

Lemma 34. *Let $\lambda \in \mathbb{N}$ be a security parameter. There exists a probabilistic polynomial-time simulator $\mathcal{S}_{\mathcal{U}}^{SGA_{MIN}}$ such for all inputs $I = (I_{\mathcal{R}}, I_{\mathcal{C}}, I_{\mathcal{U}})$, we have*

$$\mathcal{S}_{\mathcal{U}}^{SGA_{MIN}}(1^\lambda, I_{\mathcal{R}}, g_{\mathcal{R}}(I)) \overset{\mathrm{c}}{\equiv} \mathsf{view}_{\mathcal{U}}^{SGA_{MIN}}(I, \lambda)\ .$$

Proof. The view of $\mathcal{U}$ contains the relation $\gamma_{A,\mathrm{MIN}(B)}(R)$. For each pair $t \in \gamma_{A,\mathrm{MIN}(B)}(R)$, $\mathcal{S}_{\mathcal{U}}$, simulator $\mathcal{S}_{\mathcal{U}}$ uses the public key pk and produces the pair $(\mathcal{E}(pk, \pi_A(t)), \mathcal{E}(pk, \pi_B(t)))$. Hence, $\mathcal{S}_{\mathcal{U}}((sk, pk), \gamma_{A,\mathrm{MIN}(B)}(R))$ describes exactly the same distribution as, which concludes the proof. □

6.6 Conclusion

We have presented efficient algorithms for grouping and aggregation operations with MapReduce that enjoy privacy guarantees such as none of the nodes of the public cloud

computing can learn the input or the output relation. To achieve our goal, we relied on Paillier's cryptosystem and on Order-Preserving encryption. We developed an efficient approach on the computation cost side as the communication cost side. We have compared this approach to the standard algorithm with respect to three fundamental criteria: computation cost, communication cost, and privacy guarantees.

Looking forward to future work, we plan to study the practical performance of our algorithms in an open-source system that implements the MapReduce paradigm as Hadoop[¶]. Additionally, we aim to investigate the grouping and aggregation computation with privacy guarantees in different big data systems (such as Spark or Flink) whose users also tend to outsource data and computations similarly to MapReduce.

[¶]Apache Hadoop: https://hadoop.apache.org/

CHAPTER 7

Secure Joins with MapReduce

In this chapter, we propose a secure approach for the two standard protocols of join computation using the MapReduce paradigm: the *cascade* protocol and the *hypercube* protocol. The secure approach relies on classic cryptographic primitives such that pseudo-random function and asymmetric encryption scheme. Both protocols allow an external user, called the MapReduce's user, to query the join of $n \geq 2$ relations owned by n different data owners. At the end of the protocol the MapReduce's user only learns the join of the n relations assuming that the public cloud and the MapReduce's user do not collude. This work has been conducted in collaboration with Xavier Bultel, Radu Ciucanu, Pascal Lafourcade, and Lihua Ye, and has been published in the paper "*Secure Joins with MapReduce*" at the 11$^{\text{th}}$ *International Symposium on Foundations & Practice of Security* (FPS, 2018) [BCG+18].

Contents

7.1 Introduction . . . 132
7.1.1 Joins with MapReduce . . . 133
7.1.2 Problem statement . . . 135
7.1.3 Contributions . . . 135
7.1.4 Outline . . . 136
7.2 n-ary Joins with MapReduce . . . 136
7.2.1 Cascade Protocol . . . 136
7.2.2 Hypercube Protocol . . . 136
7.3 Secure n-ary Joins with MapReduce . . . 137
7.3.1 Preprocessing for Secure Protocols . . . 137
7.3.2 Secure n-ary Joins with MapReduce and Cascade Protocol . . . 138
7.3.3 Secure n-ary Joins with MapReduce and Hypercube Protocol . . . 139
7.4 Experimental Results . . . 140
7.4.1 Dataset and Settings . . . 140
7.4.2 Results . . . 141
7.5 Security Proofs . . . 142
7.5.1 Security Proof for SCAS Protocol . . . 142
7.5.2 Security Proof for SHYP Protocol . . . 147
7.6 Conclusion . . . 151

7.1 Introduction

We address the fundamental problem of computing relational joins between an arbitrary number of relations in a privacy-preserving manner using the MapReduce programming model. We assume that the data, i.e., the content of relations, is externalized into the public cloud by the data owner and there is a MapReduce's user that is allowed to query it as shown in Figure 7.1. This standard model has been used recently by Dolev et al. [DLS16].

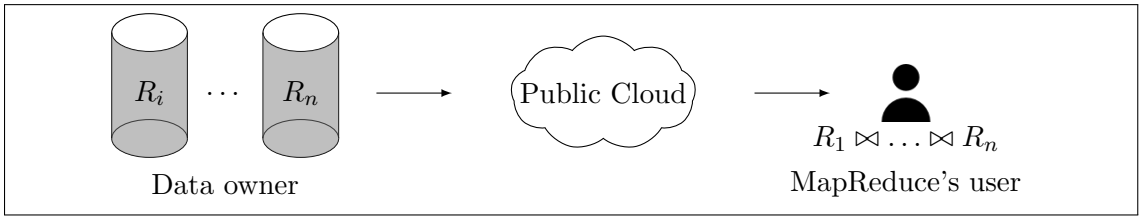
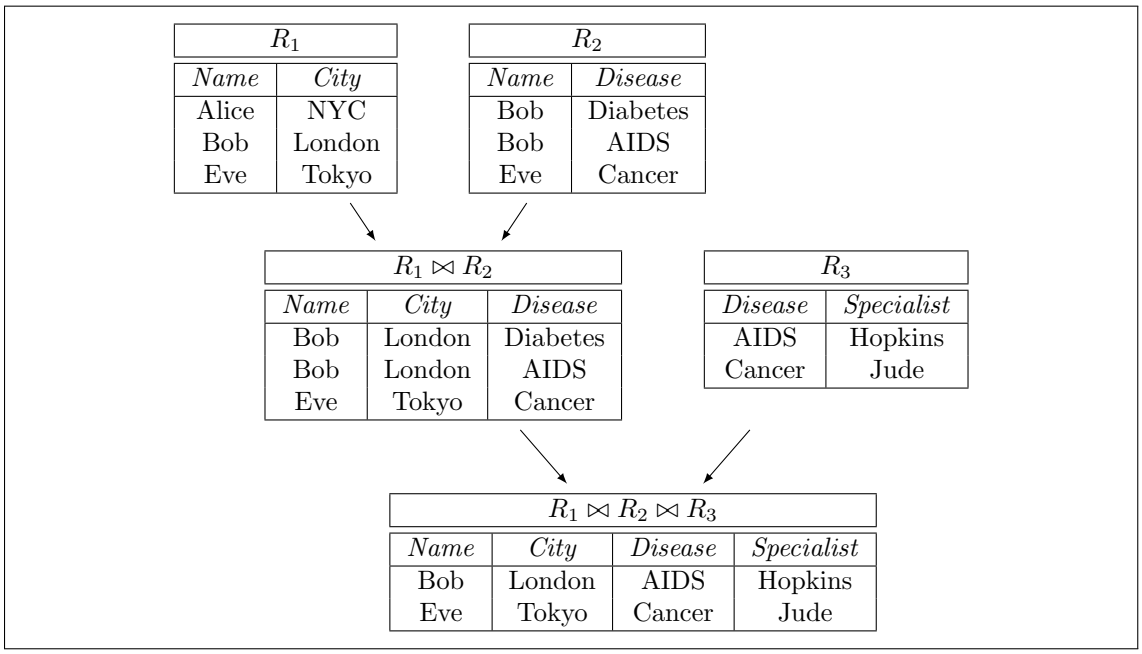


Figure 7.1: The system architecture.

We next present via a running example the concept of relational joins. Then, we present MapReduce computations, our problem statement, and illustrate the privacy issues related to joins computation with MapReduce.

Example 3. *The data owner is a hospital storing relations R_1, R_2, R_3 cf. Figure 7.2. The natural join of these relations, denoted $R_1 \bowtie R_2 \bowtie R_3$, is the relation whose tuples are composed of tuples of R_1, R_2 and R_3 that agree on shared attributes. In our case, the attribute* Name *is shared between R_1 and R_2. Moreover, the attribute* Disease *is shared between intermediate join result $(R_1 \bowtie R_2)$ and relation R_3. In Figure 7.2, we give both the intermediate result $(R_1 \bowtie R_2)$ and the final result $(R_1 \bowtie R_2) \bowtie R_3$. We observe that tuple $(Alice, NYC)$ from relation R_1, tuple $(Bob, Diabetes)$ from relation R_2, and tuple $(Bob, London, Diabetes)$ from relation $R_1 \bowtie R_2$ do not participate to the final result.*



R_1

Name	*City*
Alice	NYC
Bob	London
Eve	Tokyo

R_2

Name	*Disease*
Bob	Diabetes
Bob	AIDS
Eve	Cancer

$R_1 \bowtie R_2$

Name	*City*	*Disease*
Bob	London	Diabetes
Bob	London	AIDS
Eve	Tokyo	Cancer

R_3

Disease	*Specialist*
AIDS	Hopkins
Cancer	Jude

$R_1 \bowtie R_2 \bowtie R_3$

Name	*City*	*Disease*	*Specialist*
Bob	London	AIDS	Hopkins
Eve	Tokyo	Cancer	Jude

Figure 7.2: Joins between relations R_1, R_2 and R_3.

7.1.1 Joins with MapReduce

Two protocols for computing relational joins with MapReduce are presented in the literature: the *Cascade* protocol (i.e., a generalization of the binary join from Chapter 2 of [LRU14]) and the *Hypercube* protocol [AU10, CBS15].

Cascade Protocol

To compute an n-ary join ($n \geq 2$), the cascade protocol uses $n-1$ MapReduce rounds, i.e., a sequence of $n-1$ binary joins. A binary join works as follows: first, it applies the Map function on the first two relations R_1 and R_2 that are spread over sets of nodes $\mathcal{R}_1$ and $\mathcal{R}_2$ of the public cloud, respectively. The Map function creates for each tuple of each relation a key-value pair where key is equal to values of shared attributes between the two relations, and value is equal to non-shared values of the tuple as well as the name of the relation. Then, the key-value pairs are grouped by key, i.e., all key-value pairs output by the map phase which have the same key are sent to the same reducer. For each key and from the associated values coming from these two relations, the Reduce function creates all possible tuples corresponding to the joins of these two relations. We obtain as intermediate result a new relation denoted Q_2 that is spread over a set of nodes $\mathcal{Q}_2$. This first step defines the first round of the cascade protocol. We illustrate this process in Figure 7.3.

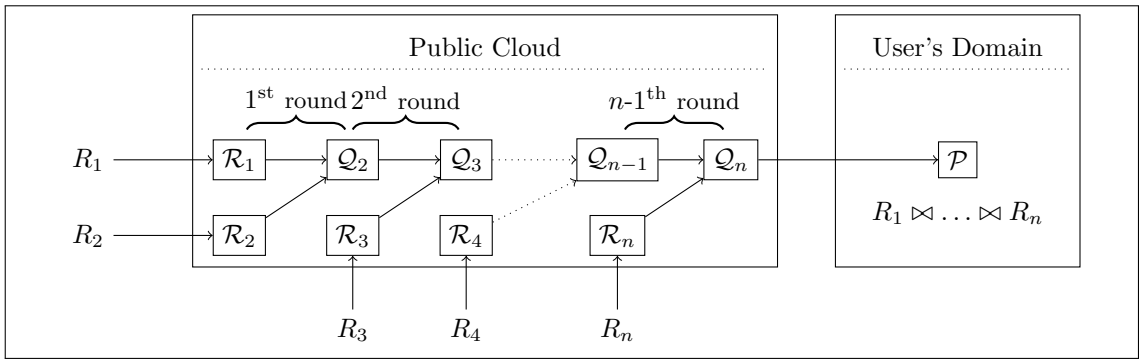


Figure 7.3: Cascade of joins with MapReduce between n relations.

Example 3 continued. To compute $(R_1 \bowtie R_2) \bowtie R_3$ with MapReduce following the cascade protocol, we start by joining R_1 and R_2. Relations R_1 and R_2 share attribute *Name*. Hence from R_1, the Map function produces the following key-value pairs: (Alice, (R_1, NYC)), (Bob, (R_1, London)), and (Eve, (R_1, Tokyo)). These key-value pairs are sent to three different reducers depending on the key value. From relation R_2, the Map function produces key-value pairs (Bob, (R_2, Diabetes)), (Bob, (R_2, AIDS)), and (Eve, (R_2, Cancer)). We stress that values of pairs (Bob, (R_2, Diabetes)) and (Bob, (R_2, AIDS)) are sent to the same reducer as the pair (Bob, (R_1, London)) since all these pairs have the same key. Similarly, (Eve, (R_2, Cancer)) and (Eve, (R_1, Tokyo)) are sent to the same reducer. The pair (Alice, (R_1, NYC)) does not participate in the join result since no other pair shares the same key. Then, from values (R_1, London), (R_2, Diabetes), and (R_2, AIDS) present on the reducer associated to the key Bob, the reduce creates all possible tuples with values coming from different relations i.e., (Bob, London, Diabetes) and (Bob, London, AIDS). Similarly, the reducer associated to the key Eve produces (Eve, Tokyo, Cancer). These tuples correspond to the relation $(R_1 \bowtie R_2)$ cf. Figure 7.2. We apply the Map and the Reduce functions on relations $(R_1 \bowtie R_2)$ and R_3 sharing the attribute *Disease*. From $(R_1 \bowtie R_2)$, the Map function produces key-value pairs: (Diabetes, ($R_1 \bowtie R_2$, Bob,

London)), (AIDS, ($R_1 \bowtie R_2$, Bob, London)), and (Cancer, ($R_1 \bowtie R_2$, Eve, Tokyo)). From R_3, the Map function produces: (AIDS, (R_3, Hopkins)), and (Cancer, (R_3, Jude)). Finally, the Reduce function step produces tuples (Bob, London, AIDS, Hopkins) and (Eve, Tokyo, Cancer, Jude) corresponding to relation $(R_1 \bowtie R_2) \bowtie R_3$ cf. Figure 7.2.

Hypercube Protocol

Contrarily to cascade, the hypercube computes the join of all n relations in only one MapReduce round. The hypercube has dimension d (where d is the number of join attributes). There are $p := \prod_{j \in [\![1,d]\!]} \alpha_j$ reducers denoted $\mathcal{H}_i$ (for $i \in [\![1, p]\!]$), where α_j is the number of buckets associated with the j^{th} attribute. Hence, each reducer $\mathcal{H}_i$ can be uniquely identified by a point in the hypercube. For each relation R_i spread over a set of nodes $\mathcal{R}_i$, the Map function computes the image of all tuples on the d dimensions of the hypercube to decide to which reducers $\mathcal{H}_i$ the tuple should be sent. Then, each reducer computes all possible combinations of input tuples that agree on shared attributes, only if all n relations are represented on the same reducer. All these combinations correspond to the final result of the n-ary join.

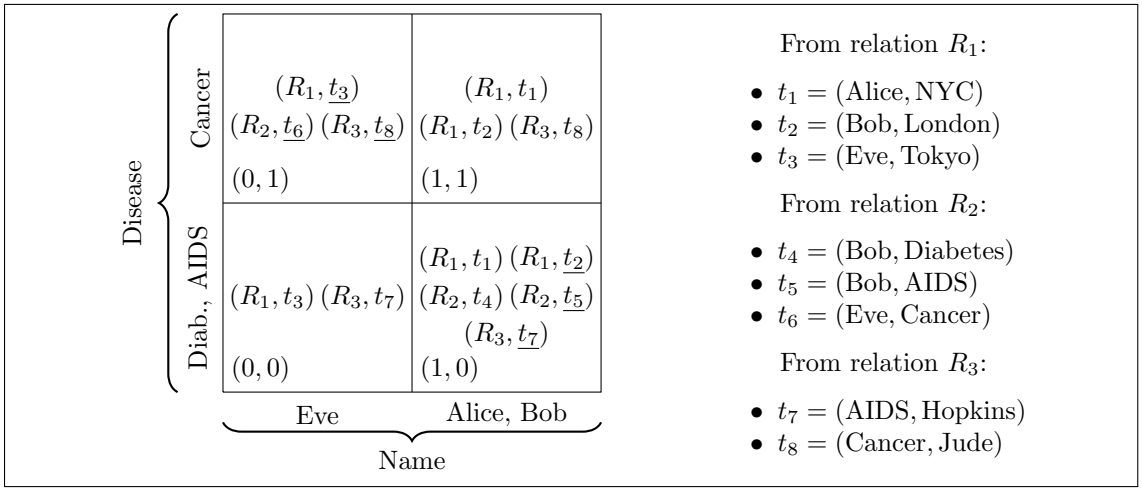


Figure 7.4: Running example with hypercube protocol. Underlined tuples correspond to tuples that participate to the final join result.

Example 3 continued. We have two join attributes (*Name* and *Disease*), hence two hash functions h_N and h_D for attributes *Name* and *Disease*, respectively. For instance, assume 4 reducers establishing a 2×2 square cf. Figure 7.4, where $h_N(\text{Eve}) = 0$, $h_N(\text{Alice}) = h_N(\text{Bob}) = 1$, $h_D(\text{Diabetes}) = h_D(\text{AIDS}) = 0$, and $h_D(\text{Cancer}) = 1$. For each tuple of each relation, we compute the value of the *Name* component (if there exists) with the hash function h_N and the value of the *Disease* component (if there exists) with the hash function h_D. For instance, the tuple $t_6 = (\text{Eve}, \text{Cancer})$ of the relation R_2 is sent to the reducer of coordinates $(0, 1)$ since $h_N(\text{Eve}) = 0$ and $h_D(\text{Cancer}) = 1$ (cf. Figure 7.4). If one of these two attributes is missing in a tuple, then the tuple is replicated over all reducers associated to the different values of the missing attributes of the tuple. For example, tuple $t_1 = (\text{Alice}, \text{NYC})$ of relation R_1 has no attribute *Disease*, and consequently, is sent to reducers $(1, 0)$ and $(1, 1)$. In such a situation we may write $(1, \star)$ to simplify presentation. Finally, each reducer performs all possible combinations over tuples that agree on join attributes of the three relations R_1, R_2, and R_3. We obviously obtain the same final result as for cascade protocol.

7.1.2 Problem statement

We assume three participants: the data owner, the public cloud, and the MapReduce's user (cf. Figure 7.1). The data owner externalizes n relations R_i with $i \in [\![1, n]\!]$ to the public cloud. We assume that the public cloud is semi-honest, i.e., it executes dutifully the computation tasks but tries to learn the maximum of information on tuples of each relation. In order to preserve privacy of data owner and to allow the join computation between relations, we want that the public cloud learns nothing about input data or join result. Moreover, we want that the user who queries the join result learns nothing else than the final join result, i.e., she does not learn information on tuples of relations that do not participate to the final result.

We denote by $\mathbb{R}_i$ the set of attributes of a relation R_i, for $i \in [\![1, n]\!]$. In the case of the cascade protocol, we denote by $\mathbb{Q}_i$ the set of attributes of relation Q_i for $i \in [\![1, n]\!]$, where $Q_1 := R_1$ and $Q_i := Q_{i-1} \bowtie R_i$ for $i \in [\![2, n]\!]$. Finally we denote by $\mathbb{X}$ the set of shared attributes between the n relations, i.e., $\mathbb{X} := |\cup_{i,j \in [\![1,n]\!]; i \neq j} \mathbb{R}_i \cap \mathbb{R}_j|$.

We expect the following security properties:

1. The MapReduce's user learns nothing else than result $R_1 \bowtie \ldots \bowtie R_n$, i.e., she does not learn tuples from the input relation that do not participate in the result.

2. Neither a set of nodes $\mathcal{R}_i$ nor data owner learn final result data.

3. A set of nodes $\mathcal{Q}_i$ (resp. $\mathcal{H}_i$) cannot learn owner's data and final result.

Example 3 continued. Looking at the three security properties of the problem statement, we see that the cascade and the hypercube protocols do not respect properties (1), (2), and (3). In fact, both protocols reveal to the public cloud all tuples of relations R_1, R_2 and R_3 since they are not encrypted. Moreover, if the MapReduce's user colludes with the intermediate set of nodes $\mathcal{R}_1 \bowtie \mathcal{R}_2$, then she learns tuples that she should not, in this case the tuple (Bob, London, Diabetes) as shown in Figure 7.2.

7.1.3 Contributions

We propose a secure approach that extend the two aforementioned join protocols while ensuring the desired security properties, and remaining efficient from both computational and communication points of view.

- The secure approach assumes that the public cloud and the user do not collude. We encrypt all values of each tuple using a public key encryption scheme with the MapReduce's user public key denoted pk. To be able to perform the equality joins between relations we rely on secure pseudo-random functions.

- We give experimental results of our secure approach for both cascade and hypercube protocols using Apache Hadoop® 3.2.0 [Fou19b] on the *Higgs Twitter Dataset* [DLMM] that has been built by Domenico et al. [DLMM13] after monitoring the spreading processes on Twitter before, during and after the announcement of the discovery of a new particle with the features of the elusive Higgs boson on 4^{th} July 2012.

- We prove that our secure approach for both protocols satisfies the security properties using the random oracle model. We also notice a limitation regarding learning repetitions between pseudo-random values which seems to us inherent because we need to perform equality checks.

7.1.4 Outline

We present in Section 7.2 the cascade and hypercube protocols for the n-ary join computation with the MapReduce paradigm. Then, we present our secure approach for both protocols in Section 7.3. In Section 7.4, we compare experimentally the performance of our secure approach vs the insecure original protocols. Then, we give security proofs for our both secure protocols in Section 7.5. Finally, we outline conclusion and future work in Section 7.6.

7.2 n-ary Joins with MapReduce

We formally present the standard protocols for computing n-ary joins $Q := R_1 \bowtie \cdots \bowtie R_n$ with MapReduce: *cascade* protocol denoted CAS, i.e., a sequence of $n-1$ rounds of binary joins [LRU14] and *hypercube* protocol denoted HYP, i.e., a single round doing all the $n-1$ joins [AU10]. We have already presented examples for both protocols in Section 7.1.1.

7.2.1 Cascade Protocol

We recall that the i^{th} round of the cascade protocol takes action between sets of nodes $\mathcal{Q}_i$ and $\mathcal{R}_{i+1}$ storing relations Q_i and R_{i+1} respectively, with $i \in [\![1, n-1]\!]$ such that $Q_1 := R_1$. Moreover, $\mathbb{R}$ denotes the schema of the relation R, i.e., the set of attributes of the relation R.

Map function:

Input: $(key, value)$
// *key*: id of chunk of Q_i or R_{i+1}
// *value*: collection of $t_q \in Q_i$ or $t_r \in R_{i+1}$
foreach $t_q \in Q_i$ **do**
  `emit`$(\pi_{\mathbb{Q}_i \cap \mathbb{R}_{i+1}}(t_q), (\mathtt{Q}_i, t_q))$
foreach $t_r \in R_{i+1}$ **do**
  `emit`$(\pi_{\mathbb{Q}_i \cap \mathbb{R}_{i+1}}(t_r), (\mathtt{R}_{i+1}, t_r))$

Reduce function:

Input: $(key, values)$
// *key*: $\pi_{\mathbb{Q}_i \cap \mathbb{R}_{i+1}}(t)$ with $t \in Q_i \cup R_{i+1}$
// *values*: collection of $(\mathtt{Q}_i, t_q)$ or $(\mathtt{R}_{i+1}, t_r)$
foreach $(\mathtt{Q}_i, t_q) \in values$ **do**
  foreach $(\mathtt{R}_{i+1}, t_r) \in values$ **do**
    `emit`$(-, t_q \bowtie t_r)$

Figure 7.5: Map and Reduce functions for CAS protocol.

To compute join between n relations $R_1, \ldots, R_n$, we apply $n-1$ times the Map and the Reduce functions of the cascade protocol presented in Figure 7.5. The final relation Q_n corresponds to $Q := R_1 \bowtie \cdots \bowtie R_n$.

7.2.2 Hypercube Protocol

We assume we have an hypercube of dimension

$$d := |\mathbb{X}| = |\{X_1, \ldots, X_d\}| = |\cup_{i,j \in [\![1,n]\!]; i \neq j} \mathbb{R}_i \cap \mathbb{R}_j| \,.$$

In other terms, d is the number of join attributes.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R_i, with $i \in [\![1, n]\!]$
// *value*: collection of $t \in R_i$
foreach $t \in R_i$ **do**
  for $\ell \in [\![1, d]\!]$ **do**
    if $X_\ell \in \mathbb{R}_i$ **then**
      $x_\ell := h_\ell(\pi_{X_\ell}(t))$
    else
      $x_\ell := \star$
  $\texttt{emit}((x_1, \ldots, x_d), (\texttt{R}_i, t))$

Reduce function:

Input: $(key, values)$
// *key*: $(x_1, \ldots, x_d)$, with $x_\ell \in [\![0, \alpha_\ell]\!]$ and $\ell \in [\![1, d]\!]$
// *values*: collection of $(\texttt{R}_i, t)$, with $i \in [\![1, n]\!]$ and $t \in R_i$
for $i \in [\![1, n]\!]$ **do**
  $R'_i := \bigcup_{(\texttt{R}_i, t) \in values} \{t\}$
for $t \in R'_1 \bowtie \cdots \bowtie R'_n$ **do**
  $\texttt{emit}(-, t)$

Figure 7.6: Map and Reduce functions for HYP protocol.

Moreover, we assume that we have d non-cryptographic hash functions, i.e., they do not necesseraly satisfy security properties given in Definition 12 on page 18. These d hash functions are denoted h_ℓ for $\ell \in [\![1, d]\!]$, such that $h_\ell \colon X_\ell \to [\![0, \alpha_\ell]\!]$ where $\alpha_\ell \in \mathbb{N}^*$ is the number of buckets for the attribute X_ℓ. Hence, the hypercube is composed of $\prod_{i=1}^{d} \alpha_i$ reducers where each reducer is uniquely identified by a d-tuple $(x_1, \ldots, x_d)$ with $x_\ell \in [\![0, \alpha_\ell]\!]$ for $\ell \in [\![1, d]\!]$.

We present in Figure 7.6 the Map and the Reduce functions of the hypercube protocol for the join computation with MapReduce between n relations $R_1, \ldots, R_n$. The Map function sends the pair to the corresponding reducer of the hypercube associated to the coordinates of the key-value pair's key where the star $\star$ in the ℓ-th coordinate means that we duplicate the tuple t on all the α_ℓ buckets of the ℓ-th dimension of the hypercube. Then, if the same reducer of the hypercube has at least one tuple coming from all the n relations and that these tuples agree on their shared attributes then the Reduce function produces all possible key-values pairs of the form $(-, t_1 \bowtie \cdots \bowtie t_n)$ where $t_i \in R_i$, with $i \in [\![1, n]\!]$, and where the $-$ symbol refers to the empty string since keys are irrelevant at the end of the computation.

7.3 Secure n-ary Joins with MapReduce

We present our secure approach for CAS and HYP protocols to compute joins between $n > 2$ relations with MapReduce, denoted SCAS and SHYP, respectively. We recall we assume that the public cloud and the MapReduce's user do not collude.

7.3.1 Preprocessing for Secure Protocols

To prevent the public cloud from learning the content of relations, the data owner protects each relation R_i for $i \in [\![1, n]\!]$ before outsourcing. The protected relation obtained from R_i is denoted R_i^* and is sent to the public cloud by the data owner.

The data owner protects relations in two ways. First, it uses a secure pseudo-random function $F(k, \cdot)$ where k is the data owner's secret key. The data owner applies $F(k, \cdot)$ on values of shared attributes of each tuples of relations R_i for $i \in [\![1, n]\!]$. Since a secure pseudo-random function is deterministic, it allows the cloud to perform equality checks between values of join attributes. On other hand, the data owner encrypts each component of tuples with an IND-CPA asymmetric encryption scheme (e.g., ElGamal [Gam85], RSA-OAEP [BR94]) using the public key pk of the MapReduce's user. Hence the encrypted values of non-shared attributes do not give any information to an adversary, e.g., to a curious public cloud provider. Values of shared attributes are also encrypted using the public scheme encryption since we want the MapReduce's user can decrypt them.

Preprocessing: **Input:** $(R_1, \ldots, R_N)$

$\mathsf{visited} := \emptyset$
for $i \in [\![1, N]\!]$ **do**
  $R_i^* := \emptyset$
  $\mathbb{R}_i^f := \{A^f | A \in \mathbb{R}_i \cap \mathbb{X}\}$
  $\mathbb{R}_i^{\mathcal{E}} := \{A^{\mathcal{E}} | A \in \mathbb{R}_i \setminus \mathsf{visited}\}$
  $\mathbb{R}_i^* := \mathbb{R}_i^f \cup \mathbb{R}_i^{\mathcal{E}}$
  foreach $t \in R_i$ **do**
    $t_f := \bigtimes_{A \in \mathbb{R}_i^f} F(k, \pi_A(t))$
    $t_{\mathcal{E}} := \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} \mathcal{E}(pk, \pi_A(t))$
    $R_i^* := R_i^* \cup \{t_f \times t_{\mathcal{E}}\}$
  $\mathsf{visited} := \mathsf{visited} \cup \mathbb{R}_i$

Figure 7.7: Preprocessing of relations.

We present the preprocessing algori thm in Figure 7.7. The set visited prevents the data owner from encrypting several times the same values. We note that A^f and $A^{\mathcal{E}}$ are just notations making explicit the correspondences between initial and outsourced data. For instance, if a relation R has one attribute "Name" that is shared with another relation, then this attribute in the protected relation will be denoted "Name^f"; we apply in the same way the notation $A^{\mathcal{E}}$. Moreover $\mathbb{R}_i^*$ is the schema of the protected relation R_i^*. We give an example for the cascade protocol in Figure 7.8 using the running example from the Introduction (cf. Section 7.1.1).

For both protocols, we remark that the public cloud knows when components of same attribute are equal since a secure pseudo-random function is deterministic. We see in Figure 7.8 that the public cloud knows that R_2^* and R_3^* share two same values of disease since values 18 and 99 are present in both relations. However, we notice that only the data owner knows the secret key k used for the pseudo-random function evaluations.

7.3.2 Secure n-ary Joins with MapReduce and Cascade Protocol

If a relation participating at the i-th round contains an attribute that will participate to the join in a following round, the protocol must anticipate the pseudo-random values of the shared attribute to perform joins. In the original cascade protocol presented in Section 7.2.1, tuples are not encrypted and the anticipation is not necessary since each tuple value is available. In the secure approach of the cascade protocol, we add in value of key-value pairs the pseudo-random evaluations of all needed pseudo-random evaluations allowing joins in other rounds. This is possible since the preprocessing performed by the data owner outsources protected relations containing pseudo-random evaluations of values of join attributes.

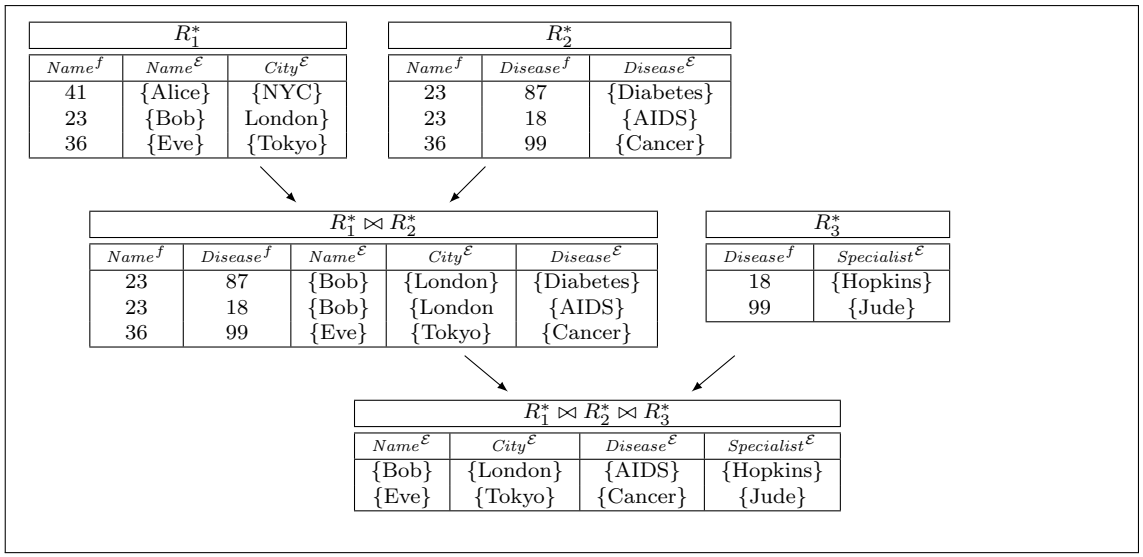


Figure 7.8: Intuition of the secure approach of the cascade protocol. We denote ciphertexts of an IND-CPA encryption scheme by $\{\cdot\}$, and pseudo-random evaluations by integers.

Map function:

Input: $(key, value)$
// *key*: id of chunk of Q_i^* or R_{i+1}^*
// *value*: collection of $t \in Q_i^*$ or $t \in R_{i+1}^*$
if $i = 1$ **then**
  foreach $t \in Q_1^*$ **do**
    `emit` $(\pi_{\mathbb{Q}_1^f \cap \mathbb{R}_2^f}(t), (\mathtt{Q}_1, t))$
foreach $t \in R_{i+1}^*$ **do**
  `emit` $(\pi_{\mathbb{Q}_i^f \cap \mathbb{R}_{i+1}^f}(t), (\mathtt{R}_{i+1}, t))$

Reduce function:

Input: $(key, values)$
// *key*: $\pi_{\mathbb{Q}_i \cap \mathbb{R}_{i+1}}(t)$ with $t \in Q_i$ or $t \in R_{i+1}$
// *values*: collection of $(\mathtt{Q}_i, t_q)$ or $(\mathtt{R}_{i+1}, t_r)$
for $(\mathtt{Q}, t_q) \in values$ **do**
  for $(\mathtt{R}, t_r) \in values$ **do**
    $t := t_q \times t_r$
    if $i \neq n - 1$ **then**
      `emit` $(\pi_{\mathbb{Q}_{i+1}^f \cap \mathbb{R}_{i+2}^f}(t), t)$
    else
      `emit` $(-, t)$

Figure 7.9: Map and Reduce functions for SCAS protocol.

Map and Reduce functions for the secure n-ary joins with MapReduce and cascade protocol are presented in Figure 7.9. For the i-th round, we recall that we have $Q_i^* := Q_{i-1}^* \bowtie R_i^*$ and $Q_1^* := R_1^*$.

7.3.3 Secure n-ary Joins with MapReduce and Hypercube Protocol

We present the Map and Reduce functions of the secure n-ary joins with MapReduce and hypercube protocol in Figure 7.10. The main difference compared to the insecure approach

is that the Map function receives encrypted tuples from the data owner. As for the secure approach with the cascade protocol, we add pseudo-random evaluations in value of each pair allowing the Reduce function to check correspondences of tuples on join attributes. We recall that the symbol $\star$ in the Map function means that coordinate x_ℓ takes all values of $[\![0, \alpha_\ell]\!]$.

Map function:

Input: $(key, value)$
// *key*: id of a chunk of R_i^*, with $i \in [\![1, n]\!]$
// *value*: collection of $t \in R_i^*$
foreach $t \in R_i^*$ **do**
  for $\ell \in [\![1, d]\!]$ **do**
    if $X_\ell^f \in \mathbb{R}_i^f$ **then**
      $x_\ell := h_\ell(\pi_{X_\ell^f}(t))$
    else
      $x_\ell := \star$
  $\texttt{emit}((x_1, \ldots, x_d), (\mathtt{R}_i, t))$

Reduce function:

Input: $(key, values)$
// *key*: $(x_1, \ldots, x_d)$, with $x_\ell \in [\![0, \alpha_\ell]\!]$, and $\ell \in [\![1, d]\!]$
// *values*: collection of $(\mathtt{R}_i, t)$, with $i \in [\![1, n]\!]$ and $t \in R_i^*$
for $i \in [\![1, n]\!]$ **do**
  $R'_i := \bigcup_{(\mathtt{R}_i, t) \in values} \{t\}$
$\mathbb{A} := \cup_{i=1}^n \mathbb{R}_i^{\mathcal{E}}$
for $t \in R'_1 \bowtie \cdots \bowtie R'_n$ **do**
  $t' := \bigtimes_{A \in \mathbb{A}} \pi_A(t)$
  $\texttt{emit}(-, t')$

Figure 7.10: Map and Reduce functions for SHYP protocol.

7.4 Experimental Results

We present experimental results for the original MapReduce approach computing the natural join with the cascade and the hypercube protocols, and we respectively compare them to our secure approach.

7.4.1 Dataset and Settings

We use the real-world *Higgs Twitter Dataset* [DLMM, DLMM13] that we denote by relation $R(A, B)$ and encoding followee-follower relation on Twitter. The relation $R(A, B)$ has $14,855,842$ tuples.

To perform natural joins with this dataset, we first randomly pick samples of $R(A, B)$, ranging from $500,000$ to $2,000,000$ tuples in steps of $250,000$. Then, for each sample, we generate two relations $S(B, C)$ and $T(C, A)$ that are copies of the sample. The join query used in our experiments is $R(A, B) \bowtie S(B, C) \bowtie T(C, A)$, consisting on all directed triangles of the *Higgs Twitter Dataset*. Using such a dataset and query is a standard practice in the database community literature to evaluate the performance of join query algorithms, as recently done, e.g., by Chu et al. [CBS15].

As mentioned in Section 7.3, our secure approach requires a pseudo-random function and an asymmetric encryption scheme. For the pseudo-random function, we use the

HMAC-SHA256 keyed-hash message authentication code [BCK96] implemented in Go package `hmac`[†]. Moreover, we use the RSA-OAEP asymmetric encryption scheme [BR94, KS98] implemented in Go package `rsa`[‡] with a 1024-bit RSA modulus. As mentioned in the previous chapter, we can also use an other asymmetric encryption scheme such as ElGamal [Gam85] instead of RSA-OAEP. However, for communication cost reason we do not use it since ElGamal ciphertexts are twice longer for the same security.

7.4.2 Results

We first report wall clock times of the secure protocols preprocessing in Table 7.1.

Table 7.1: Wall clock times of the secure protocols preprocessing.

Number of tuples / relation	Wall clock time
500,000	38.271 s
750,000	58.735 s
1,000,000	79.133 s
1,250,000	97.410 s
1,500,000	116.744 s
1,750,000	137.618 s
2,000,000	157.392 s

We present in Figure 7.11 running times for CAS and HYP protocols and their respective secure approach, namely SCAS and SHYP protocols. We report average times over eight runs. We consider size up to 2,000,000 tuples because after such a size, our machine gives out-of-memory errors hence we cannot compare meaningful results for all protocols. For HYP and SHYP protocols, we use four buckets for each of the three dimensions defined by attributes A, B, and C, hence a total number of $4^3 = 64$ reducers.

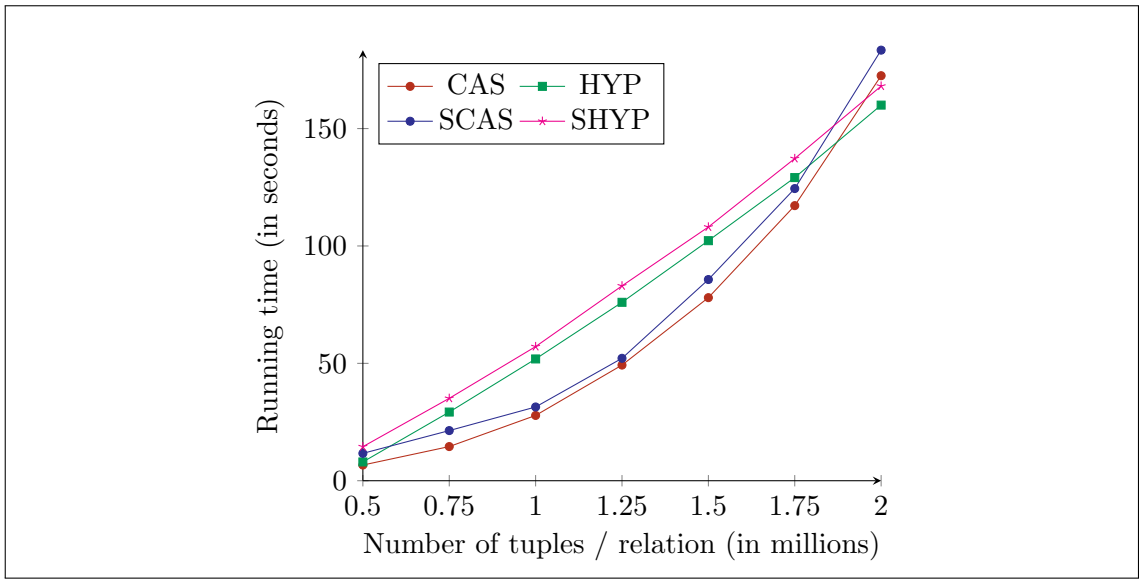


Figure 7.11: CPU time vs the number of tuples per relation for CAS and HYP protocols, and their respective secure approach.

We observe that HYP protocol seems better for the largest input data sizes. We also remark that CPU times of no-secure and secure approaches are very similar. Indeed,

[†] https://golang.org/pkg/crypto/hmac/
[‡] https://golang.org/pkg/crypto/rsa/

secure approaches deal with encrypted data but its encryption is done by data owners. However, wall clock times are different. For instance, it takes 191 s for the CAS protocol to compute the triangle join between three relations of 1,250,000 tuples while the SCAS protocol takes 768 s for the same querie. The difference comes from Apache Hadoop® which has to write on the disk the intermediate relation composed of larger encrypted data.

7.5 Security Proofs

We provide formal security proofs for SCAS and SHYP protocols. We use the standard multiparty computations definition of security against semi-honest adversaries [Lin17]. We assume that the join computation is performed between $n \geq 2$ relations. Each relation R_i, with $i \in [\![1, n]\!]$, is made of $\sigma_i \in \mathbb{N}^*$ tuples.

In the following, we denote by $T := [t_1, \ldots, t_{\sum_{i=1}^{n} \sigma_i}]$, the ordered list composed of tuples from the n relations, from R_1 to R_n. Hence, $t_1 \in T$ corresponds to the first tuple of R_1, t_{σ_1} corresponds to the last tuple of R_1, t_{σ_1+1} corresponds to the first tuple of R_2, and so forth. Using T, we build $\bar{T}$ as the ordered sublist of T made of the first occurence for each different tuple. We also define T_i, with $i \in [\![1, |\bar{T}|]\!]$, as the vector composed of indexes of t_i in T. Finally, we denote by $\gamma_i \in \mathbb{N}^*$, with $i \in [\![2, n]\!]$, the number of tuples of the intermediate relation $Q_i := Q_{i-1} \bowtie R_i$.

7.5.1 Security Proof for SCAS Protocol

Our secure approach assumes that the public cloud nodes may collude. Hence, in a security point of view, all sets of nodes of type $\mathcal{R}$ and $\mathcal{Q}$ are considered as a unique set of nodes when they collude, denoted $\mathcal{C}$.

We model the SCAS protocol with $n + 2$ parties $P_{R_1}, \ldots, P_{R_n}$, $P_{\mathcal{C}}$, and $P_{\mathcal{U}}$ using respective inputs $I := (I_{R_1}, \ldots, I_{R_n}, I_{\mathcal{C}}, I_{\mathcal{U}}) \in \mathcal{I}$, and a function $g := (g_{R_1}, \ldots, g_{R_n}, g_{\mathcal{C}}, g_{\mathcal{U}})$ such that:

- P_{R_i}, for $i \in [\![1, n]\!]$, is the data owner of relation R_i. It has the input $I_{R_i} := (R_i, \mathbb{X}, k, pk)$, where R_i is its private relation, $\mathbb{X}$ is the set of common attributes between relations, k is the PRF secret key shared between data owners, and pk is the public key of the MapReduce's user. P_{R_i} returns $g_{R_i}(I) := \perp$ because it does not learn anything.

- $P_{\mathcal{C}}$ is the public cloud nodes that represents the collusion between all its sets of nodes. It has the input $I_{\mathcal{C}} := pk$, where pk is the public key of the user. $P_{\mathcal{C}}$ returns

$$g_{\mathcal{C}}(I) := \left(\{\sigma_i\}_{i \in [\![1,n]\!]}, \{\gamma_i\}_{i \in [\![2,n]\!]}, |\bar{T}|, \{T_i\}_{i \in [\![1,|\bar{T}|]\!]} \right) ,$$

  because it learns the cardinal of relations $\{R_i\}_{i=1}^{n}$ and $\{Q_i\}_{i=2}^{n}$, and cross-column correlations.

- $P_{\mathcal{U}}$ is the set of nodes $\mathcal{U}$ of the MapReduce's user. It has the input $I_{\mathcal{U}} := (pk, sk)$, where (pk, sk) is the key pair of the MapReduce's user. $P_{\mathcal{U}}$ returns $g_{\mathcal{U}}(I) := R_1 \bowtie \cdots \bowtie R_n$ because the MapReduce's user obtains the result of the join at the end of the protocol.

The security of SCAS protocol is given in Theorem 11.

Theorem 11. *Assume F is a secure pseudo-random function and that Π is an IND-CPA asymmetric encryption scheme, then* SCAS *securely performs the join computation in the presence of semi-honest adversaries.*

The security proof for Theorem 11 is decomposed in Lemma 35 for parties $P_{\mathcal{R}_i}$ (with $i \in [\![1, n]\!]$), in Lemma 36 for the party $P_{\mathcal{C}}$, and in Lemma 37 for the party $\mathcal{U}$.

Lemma 35. *There exists probabilistic polynomial-time simulators $\mathcal{S}^{\text{SCAS}}_{\mathcal{R}_i}$ for $i \in [\![1, n]\!]$ such that*

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

Proof. We assume that $\mathcal{R}_i$ (with $i \in [\![1, n]\!]$) is corrupted. We observe that $P_{\mathcal{R}_i}$ receives no output and no incoming message from other parties. Thus, we merely need to show that a simulator can generate the view of party $P_{\mathcal{R}_i}$ from its inputs. Simulator $\mathcal{S}^{\text{SCAS}}_{\mathcal{R}_i}$ has input $(R_i, \mathbb{X}, k, pk)$. It encrypts each tuple of the relation following the preprocessing algorithm presented in Figure 7.7 on page 138 to build R^*_i. Hence, $\mathcal{S}^{\text{SCAS}}_{\mathcal{R}_i}$ performs exactly the same computation as SCAS protocol and describes exactly the same distribution as $\mathsf{view}^{\text{SCAS}}_{\mathcal{R}_i}(I, \lambda)$. □

Lemma 36. *Assume F is a secure pseudo-random function and that Π is an IND-CPA asymmetric encryption scheme, then there exists probabilistic polynomial-time simulator $\mathcal{S}^{\text{SCAS}}_{\mathcal{C}}$ such that*

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

Proof. Let $\lambda \in \mathbb{N}$ be a security parameter. Before to build $\mathcal{S}^{\text{SCAS}}_{\mathcal{C}}$ that computes a distribution that can be simulated perfectly, we use the hybrid argument to build hybrid simulators denoted $\mathcal{S}^{H\text{-SCAS}}_{\mathcal{C}}$. The simulator, $\mathcal{S}^{H-\text{SCAS}}_{\mathcal{C}}$ works as SCAS but each evaluation of the pseudo-random function performed by parties $\mathcal{R}_i$ for $i \in [\![1, n]\!]$ are replaced using the random oracle OPRF presented in Figure 7.12. Simulator $\mathcal{S}^{H\text{-SCAS}}_{\mathcal{C}}$ is, as for it, presented in Figure 7.13. The view contains all encrypted relations sent by data owners. It also contains intermediate encrypted relations computed by the public cloud, and the join of all encrypted relations sent by the public cloud to the MapReduce's user.

$\mathsf{OPRF}(x)$:
if $T[x] = \emptyset$ **then**
  $T[x] \xleftarrow{\$} \{0,1\}^{|\mathcal{Y}|}$
return $T[x]$.

Figure 7.12: Random oracle OPRF.

Assume there exists a polynomial-time distinguisher D such that for all inputs $I \in \mathcal{I}$, we have

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

where μ is a non-negligible function in λ. We show how to build a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}$ has a non-negligible advantage to win the pseudo-random function distinguishing experiment $\mathsf{E}^{\text{prf-}b}_{F,\mathcal{A}}$ (with $b \in \{0, 1\}$) against the pseudo-random function family F. Then we conclude the proof by contraposition. Adversary $\mathcal{A}$ is presented in Figure 7.14. At the end of its execution, $\mathcal{A}$ uses the distinguisher D to compute the bit b_* before returning it. First, we remark that

$$\Pr\left[\mathsf{Exp}^{\text{prf-0}}_{F,\mathcal{A}}(\lambda) = 1\right] = \Pr\left[D(\mathcal{S}^{H\text{-SCAS}}_{\mathcal{C}}(1^\lambda, I_{\mathcal{C}}, g_{\mathcal{C}}(I))) = 1\right] .$$

Indeed, when $b = 0$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}^{H\text{-SCAS}}_{\mathcal{C}}$. Then the probability that the experiment returns 1 is equal to the probability

Simulator: $\mathcal{S}_{\mathcal{C}}^{H\text{-SCAS}}(1^{\lambda}, pk, (\mathbb{X}, \{\sigma_i\}_{i=1}^{*}, \{\gamma_i\}_{i=2}^{n}, |\bar{T}|, \{T_i\}_{i=1}^{|\bar{T}|}))$

for $i \in [\![1, |\bar{T}|]\!]$ **do**
  $t_i \xleftarrow{\$} \bigtimes_{A \in \mathbb{X}} A$
$\mathsf{visited} := \emptyset$
for $i \in [\![1, n]\!]$ **do**
  $R_i^* := \emptyset$
  $\mathbb{R}_i^{\mathcal{E}} := \{A \colon A \in \mathbb{R}_i \setminus \mathsf{visited}\}$
  for $j \in [\![1, |\bar{T}|]\!]$ **do**
    $t_f := \bigtimes_{A \in \mathbb{R}_i \cap \mathbb{X}} \mathsf{OPRF}(\pi_A(t_j))$
    for $l \in [\![1, T_j[i-1]]\!]$ **do**
      $t \xleftarrow{\$} \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} A$
      $t_{\mathcal{E}} := \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} \mathcal{E}(pk, \pi_A(t))$
      $R_i^* := R_i^* \{t_f \times t_{\mathcal{E}}\}$
    $\mathsf{visited} := \mathsf{visited} \cup \mathbb{R}_i$
$Q_1^* := R_1^*$
for $i \in [\![1, n-1]\!]$ **do**
  $Q_{i+1}^* := Q_i^* \bowtie R_{i+1}^*$
$\mathsf{view} := \{(R_1^*, \ldots, R_n^*), (Q_2^*, \ldots, Q_n^*)\}$

Figure 7.13: Simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SCAS}}$ for the proof of Lemma 36.

Adversary: $\mathcal{A}(pk)$

for $i \in [\![1, |\bar{T}|]\!]$ **do**
  $t_i \xleftarrow{\$} \bigtimes_{A \in \mathbb{X}} A$
$\mathsf{visited} := \emptyset$
for $i \in [\![1, n]\!]$ **do**
  $R_i^* := \emptyset$
  $\mathbb{R}_i^{\mathcal{E}} := \{A \colon A \in \mathbb{R}_i \setminus \mathsf{visited}\}$
  for $j \in [\![1, |\bar{T}|]\!]$ **do**
    $t_f := \bigtimes_{A \in \mathbb{R}_i \cap \mathbb{X}} f_b(\pi_A(t_j))$
    for $l \in [\![1, T_j[i-1]]\!]$ **do**
      $t \xleftarrow{\$} \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} A$
      $t_{\mathcal{E}} := \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} \mathcal{E}(pk, \pi_A(t))$
      $R_i^* := R_i^* \{t_f \times t_{\mathcal{E}}\}$
    $\mathsf{visited} := \mathsf{visited} \cup \mathbb{R}_i$
$Q_1^* := R_1^*$
for $i \in [\![1, n-1]\!]$ **do**
  $Q_{i+1}^* := Q_i^* \bowtie R_{i+1}^*$
$\mathsf{view} := \{(R_1^*, \ldots, R_n^*), (Q_2^*, \ldots, Q_n^*)\}$
$b_* := D(\mathsf{view})$
return b_*

Figure 7.14: Adversary $\mathcal{A}$ for the proof of Lemma 36.

that the distinguisher returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SCAS}}$. On the other hand, we have

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

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in real protocol SCAS. Then the probability that the experiment returns 1 is equal to the probability that the

distinguisher returns 1 on inputs computed as in real protocol. Finally, we evaluate the probability that $\mathcal{A}$ wins the PRF experiment

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

which is non-negligible. However, we assume that F is a secure pseudo-random function, hence, it does not exist D such that

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

is non-negligible. Hence, we have

$$\mathcal{S}^{H\text{-}\mathrm{SCAS}}_{\mathcal{C}}(I_{\mathcal{C}},g_{I_{\mathcal{C}}}(I,\lambda))\stackrel{\mathrm{c}}{\equiv}\mathsf{view}^{\mathrm{SCAS}}_{\mathcal{C}}(I,\lambda)\ .$$

Finally, we show how to build the simulator $\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}$. The difference between $\mathcal{S}^{H\text{-}\mathrm{SCAS}}_{\mathcal{C}}$ and $\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}$ is that $\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}$ substitutes Π encryption of real values by Π encryption of random values of the same size. However, simulator $\mathcal{S}^{H\text{-}\mathrm{SCAS}}_{\mathcal{C}}$ still picks random values, hence we do not formally present $\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}$ and assume that random values are rewritten by other random values.

Now we show that we have

$$\mathcal{S}^{H\text{-}\mathrm{SCAS}}_{\mathcal{C},\mathcal{U}}(1^{\lambda},I_{\mathcal{C}},g_{I_{\mathcal{C}}}(I))\stackrel{\mathrm{c}}{\equiv}\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}(1^{\lambda},I_{\mathcal{C}},g_{I_{\mathcal{C}}}(I))\ .$$

Let λ be the security parameter. Assume there exists a distinguisher $D\in\mathrm{PPT}(\lambda)$ such that for all inputs I, we have

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

where $\mu(\cdot)$ is a non-negligible function in λ.

We construct a guessing adversary $\mathcal{B}\in\mathrm{PPT}(\lambda)$ that uses D to win the IND-CPA experiment. Adversary $\mathcal{B}$ is presented in Figure 7.15.

First, we remark that

$$\Pr[\mathsf{Exp}^{\text{indcpa-0}}_{\Pi,\mathcal{B}}(\lambda)=1]=\Pr[D(\mathcal{S}^{H\text{-}\mathrm{SCAS}}_{\mathcal{C}}(1^{\lambda},I_{\mathcal{C}},g_{I_{\mathcal{C}}}(I)))=1]\ .$$

When $b=0$, the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}^{H\text{-}\mathrm{SCAS}}_{\mathcal{C}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}^{H\text{-}\mathrm{SCAS}}_{\mathcal{C}}$. On the other hand, we have

$$\Pr[\mathsf{Exp}^{\text{indcpa-1}}_{\Pi,\mathcal{B}}(\lambda)=1]=\Pr[D(\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}(1^{\lambda},I_{\mathcal{C}},g_{I_{\mathcal{C}}}(I)))=1]\ .$$

Indeed, when $b=1$ the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}$.

Finally, we evaluate the probability that $\mathcal{B}$ wins the experiment

$$\begin{aligned}\mathsf{Adv}^{\mathrm{indcpa}}_{\Pi,\mathcal{B}}(\lambda) &= \left|\Pr[\mathsf{Exp}^{\text{indcpa-1}}_{\Pi,\mathcal{B}}(\lambda)=1]-\Pr[\mathsf{Exp}^{\text{indcpa-0}}_{\Pi,\mathcal{B}}(\lambda)=1]\right| \\ &= \left|\Pr[D(\mathcal{S}^{\mathrm{SCAS}}_{\mathcal{C}}(1^{\lambda},I_{\mathcal{C}},g_{I_{\mathcal{C}}}(I)))=1]-\Pr[D(\mathcal{S}^{H\text{-}\mathrm{SCAS}}_{\mathcal{C}}(1^{\lambda},I_{\mathcal{C}},g_{I_{\mathcal{C}}}(I)))=1]\right| \\ &= \mu(\lambda)\ ,\end{aligned}$$

Adversary: $\mathcal{B}(pk)$

for $i \in [\![1, |\bar{T}|]\!]$ **do**
  $t_i \xleftarrow{\$} \bigtimes_{A \in \mathbb{X}} A$
$\mathsf{visited} := \emptyset$
for $i \in [\![1, n]\!]$ **do**
  $R_i^* := \emptyset$
  $\mathbb{R}_i^{\mathcal{E}} := \{A \colon A \in \mathbb{R}_i \setminus \mathsf{visited}\}$
  for $j \in [\![1, |\bar{T}|]\!]$ **do**
    $t_f := \bigtimes_{A \in \mathbb{R}_i \cap \mathbb{X}} f_b(\pi_A(t_j))$
    for $l \in [\![1, T_j[i-1]]\!]$ **do**
      $t_1 \xleftarrow{\$} \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} A$
      $t_2 \xleftarrow{\$} \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} A$
      $t_{\mathcal{E}} := \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} \mathcal{E}(pk, \mathsf{LoR}_b(\pi_A(t_1), \pi_A(t_2)))$
      $R_i^* := R_i^* \{t_f \times t_{\mathcal{E}}\}$
  $\mathsf{visited} := \mathsf{visited} \cup \mathbb{R}_i$
$Q_1^* := R_1^*$
for $i \in [\![1, n-1]\!]$ **do**
  $Q_{i+1}^* := Q_i^* \bowtie R_{i+1}^*$
$\mathsf{view} := \{(R_1^*, \ldots, R_n^*), (Q_2^*, \ldots, Q_n^*)\}$
$b_* := D(\mathsf{view})$
return b_*

Figure 7.15: Adversary $\mathcal{B}$ for the proof of Lemma 36.

which is non-negligible. However, we assume that Π is IND-CPA. Hence, we have

$$\mathcal{S}_{\mathcal{C}}^{\text{SCAS}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \overset{\text{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{H\text{-SCAS}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \ .$$

By transitivity, we have

$$\mathcal{S}_{\mathcal{C}}^{\text{SCAS}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \overset{\text{c}}{\equiv} \mathsf{view}_{\mathcal{C}}^{\text{SCAS}}(I, \lambda) \ ,$$

which concludes the proof. □

Lemma 37. *There exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{U}}^{\text{SCAS}}$ *such that*

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

Simulator: $\mathcal{S}_{\mathcal{U}}^{\text{SCAS}}(1^\lambda, (sk, pk), R)$

$R^* := \emptyset$
foreach $t \in R$ **do**
  $R^* := R^* \cup \left\{\bigtimes_{A \in \mathbb{R}} \mathcal{E}(pk, \pi_A(t))\right\}$
$\mathsf{view} := R^*$

Figure 7.16: Simulator $\mathcal{S}_{\mathcal{U}}^{\text{SCAS}}$ for the proof of Lemma 37.

Proof. Simulator $\mathcal{S}_{\mathcal{U}}^{\text{SCAS}}$ is presented in Figure 7.16. The view of $\mathcal{U}$ only contains the encryption of the join query that is sent by the public cloud to the MapReduce's user. Each tuple value of the result is encrypted using the IND-CPA cryptosystem Π and the user's public key. We remark that $\mathcal{S}_{\mathcal{U}}^{\text{SCAS}}$ describes exactly the same distribution as $\mathsf{view}_{\mathcal{U}}^{\text{SCAS}}$, which concludes the proof. □

7.5.2 Security Proof for SHYP Protocol

The SHYP protocol runs in only one MapReduce round. Our secure approach assumes that the public cloud nodes (i.e., the set of nodes executing the Map function and the set of nodes executing the Reduce function) may collude. Hence, in a security point of view, all sets of nodes are considered as a unique set of nodes when they collude, denoted $\mathcal{C}$.

We model SHYP protocol with $n+2$ parties $P_{R_1}, \ldots, P_{R_n}$, $P_{\mathcal{C}}$, and $P_{\mathcal{U}}$ using respective inputs $I := (I_{R_1}, \ldots, I_{R_n}, I_{\mathcal{C}}, I_{\mathcal{U}}) \in \mathcal{I}$, and a function $g := (g_{R_1}, \ldots, g_{R_n}, g_{\mathcal{C}}, g_{\mathcal{U}})$ such that:

- P_{R_i}, for $i \in [\![1, n]\!]$, is the data owner of relation R_i. It has the input $I_{R_i} := (R_i, \mathbb{X}, k, pk)$, where R_i is its private relation, $\mathbb{X}$ is the set of shared attributes between data owners, k is the PRF secret key shared between data owners, and pk is the public key of the MapReduce's user. P_{R_i} returns $g_{R_i}(I) := \perp$ because it does not learn anything.

- $P_{\mathcal{C}}$ is the public cloud nodes. It has the input $I_{\mathcal{C}} := pk$, where pk is the public key of the user. $P_{\mathcal{C}}$ returns

$$g_{\mathcal{C}}(I) := (\{\sigma_i\}_{i \in [\![1,n]\!]}, |\bar{T}|, \{T_i\}_{i \in [\![1,|\bar{T}|]\!]}) \,,$$

  because it learns the cardinal of each relation $\{R_i\}_{i \in [\![1,n]\!]}$ and cross-column correlations.

- $P_{\mathcal{U}}$ is the set of nodes $\mathcal{U}$ of the MapReduce's user. It has the input $I_{\mathcal{U}} := (pk, sk)$, where (pk, sk) is the key pair of the MapReduce's user. $P_{\mathcal{U}}$ returns $g_{\mathcal{U}}(I) := R_1 \bowtie \cdots \bowtie R_n$ because the user obtains the result of the join at the end of the protocol.

The security of SHYP protocol is given in Theorem 12.

Theorem 12. *Assume F is a secure pseudo-random function and that Π is an IND-CPA asymmetric encryption scheme, then* SHYP *securely performs the join computation in the presence of semi-honest adversaries.*

The security proof for Theorem 12 is decomposed in Lemma 38 for parties $P_{\mathcal{R}_i}$ (with $i \in [\![1, n]\!]$), in Lemma 39 for the party $P_{\mathcal{C}}$, and in Lemma 40 for the party $\mathcal{U}$.

Lemma 38. *There exists probabilistic polynomial-time simulators $\mathcal{S}^{\text{SHYP}}_{\mathcal{R}_i}$ for $i \in [\![1, n]\!]$ such that*

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

The proof of Lemma 38 is the same than the proof of Lemma 35 on page 143.

Lemma 39. *Assume F is a secure pseudo-random function and that Π is an IND-CPA asymmetric encryption scheme, then there exists probabilistic polynomial-time simulator $\mathcal{S}^{\text{SHYP}}_{\mathcal{C}}$ such that*

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

Proof. Let $\lambda \in \mathbb{N}$ be a security parameter. Before to build $\mathcal{S}^{\text{SHYP}}_{\mathcal{C}}$ that computes a distribution that can be simulated perfectly, we use the hybrid argument to build hybrid simulators denoted $\mathcal{S}^{H\text{-SHYP}}_{\mathcal{C}}$. The simulator, $\mathcal{S}^{H-\text{SHYP}}_{\mathcal{C}}$ works as SHYP but each evaluation of the pseudo-random function performed by parties $\mathcal{R}_i$ for $i \in [\![1, n]\!]$ are replaced using the random oracle OPRF presented in Figure 7.12. Simulator $\mathcal{S}^{H\text{-SHYP}}_{\mathcal{C}}$ is, as for it, presented in Figure 7.17. The view contains all encrypted relations sent by data owners. It also contains all encrypted relations computed by each reducer of the public cloud forming the join of the relations that is sent to the MapReduce's user.

Simulator: $\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}\big(1^{\lambda}, pk, (\{\sigma_i\}_{i\in[\![1,n]\!]}, |\bar{T}|, \{T_i\}_{i\in[\![1,|\bar{T}|]\!]}\big)$

for $i \in [\![1, |\bar{T}|]\!]$ **do**
  $t_i \xleftarrow{\$} \bigtimes_{A\in\mathbb{X}} A$
$\mathsf{visited} := \emptyset$
for $i \in [\![1, n]\!]$ **do**
  $R_i^* := \emptyset$
  $R_{H_i}^* := \emptyset$
  $\mathbb{R}_i^{\mathcal{E}} := \{A\colon A \in \mathbb{R}_i \setminus \mathsf{visited}\}$
  for $j \in [\![1, |\bar{T}|]\!]$ **do**
    $t_f := \bigtimes_{A\in\mathbb{R}_i^f} \mathsf{OPRF}(\pi_A(t_j))$
    for $\ell \in [\![1, d]\!]$ **do**
      if $X_\ell \in \mathbb{R}_i^f$ **then**
        $x_\ell := h_\ell(\pi_{X_\ell}(t_f))$
      else
        $x_\ell := \star$
    $\mathcal{K} := \mathcal{K} \cup \{(x_1, \ldots, x_d)\}$
    for $l \in [\![1, T_j[i-1]]\!]$ **do**
      $t \xleftarrow{\$} \bigtimes_{A\in\mathbb{R}_i^{\mathcal{E}}} A$
      $t_{\mathcal{E}} := \bigtimes_{A\in\mathbb{R}_i^{\mathcal{E}}} \mathcal{E}(pk, \pi_A(t))$
      $R_i^* := R_i^* \cup \{t_f \times t_{\mathcal{E}}\}$
      $R_{H_i}^* := R_{H_i}^* \cup \{((x_1, \ldots, x_\ell), (\mathtt{R}_i, t_f \times t_{\mathcal{E}}))\}$,
  $\mathsf{visited} := \mathsf{visited} \cup \mathbb{R}_i$
foreach $\kappa \in \mathcal{K}$ **do**
  for $i \in [\![1, n]\!]$ **do**
    $\bar{R}_i^* := \bigcup_{(\kappa,(\mathtt{R}_i,t))\in\cup_{j\in[\![1,n]\!]} R_i^*} \{t\}$
  $Q_\kappa := \bar{R}_1^* \bowtie \cdots \bowtie \bar{R}_n^*$
$\mathsf{view} := \big(\{R_i^*\}_{i\in[\![1,n]\!]}, \{R_{H_i}^*\}_{i\in[\![1,n]\!]}, \{Q_\kappa\}_{\kappa\in\mathcal{K}}\big)$

Figure 7.17: Simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}$ for the proof of Lemma 39.

Assume there exists a polynomial-time distinguisher D such that for all inputs $I \in \mathcal{I}$, we have

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

where μ is a non-negligible function in λ. We show how to build a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}$ has a non-negligible advantage to win the pseudo-random function distinguishing experiment $\mathsf{Exp}_{F,\mathcal{A}}^{\text{prf-}b}$ (with $b \in \{0,1\}$) against the pseudo-random function family F. Then we conclude the proof by contraposition. Adversary $\mathcal{A}$ is presented in Figure 7.18. At the end of its execution, $\mathcal{A}$ uses the distinguisher D to compute the bit b^* before returning it. First, we remark that

$$\Pr\left[\mathsf{Exp}_{F,\mathcal{A}}^{\text{prf-}0}(\lambda) = 1\right] = \Pr\left[D(\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}(1^{\lambda}, I_{\mathcal{C}}, g_{\mathcal{C}}(I))) = 1\right]\ .$$

Indeed, when $b = 0$, the view that $\mathcal{A}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}$. Then the probability that the pseudo-random function distinguishing experiment $\mathsf{Exp}_{F,\mathcal{A}}^{\text{prf-}b}$ experiment returns 1 is equal to the probability that the distinguisher returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}$. On the other hand, we have

$$\Pr\left[\mathsf{Exp}_{F,\mathcal{A}}^{\text{prf-}1}(\lambda) = 1\right] = \Pr\left[D(\mathsf{view}_{\mathcal{C}}^{\mathrm{SHYP}}(I, \lambda)) = 1\right]\ .$$

When $b = 1$, the view that $\mathcal{A}$ uses as input for D is computed as in real SHYP protocol. Then the probability that the experiment returns 1 is equal to the probability that the

Adversary: $\mathcal{A}(pk)$

for $i \in [\![1, |\bar{T}|]\!]$ **do**
  $t_i \xleftarrow{\$} \bigtimes_{A \in \mathbb{X}} A$
$\mathsf{visited} := \emptyset$
for $i \in [\![1, n]\!]$ **do**
  $R_i^* := \emptyset$
  $R_{H_i}^* := \emptyset$
  $\mathbb{R}_i^{\mathcal{E}} := \{A \colon A \in \mathbb{R}_i \setminus \mathsf{visited}\}$
  for $j \in [\![1, |\bar{T}|]\!]$ **do**
    $t_f := \bigtimes_{A \in \mathbb{R}_i^f} f_b(\pi_A(t_j))$
    for $\ell \in [\![1, d]\!]$ **do**
      if $X_\ell \in \mathbb{R}_i^f$ **then**
        $x_\ell := h_\ell(\pi_{X_\ell}(t_f))$
      else
        $x_\ell := \star$
    $\mathcal{K} := \mathcal{K} \cup \{(x_1, \ldots, x_d)\}$
    for $l \in [\![1, T_j[i-1]]\!]$ **do**
      $t \xleftarrow{\$} \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} A$
      $t_{\mathcal{E}} := \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} \mathcal{E}(pk, \pi_A(t))$
      $R_i^* := R_i^* \cup \{t_f \times t_{\mathcal{E}}\}$
      $R_{H_i}^* := R_{H_i}^* \cup \{((x_1, \ldots, x_\ell), (\mathtt{R}_i, t_f \times t_{\mathcal{E}}))\}$,
  $\mathsf{visited} := \mathsf{visited} \cup \mathbb{R}_i$
foreach $\kappa \in \mathcal{K}$ **do**
  for $i \in [\![1, n]\!]$ **do**
    $\bar{R}_i^* := \bigcup_{(\kappa, (\mathtt{R}_i, t)) \in \cup_{j \in [\![1,n]\!]} R_i^*} \{t\}$
  $Q_\kappa := \bar{R}_1^* \bowtie \cdots \bowtie \bar{R}_n^*$
$\mathsf{view} := \big(\{R_i^*\}_{i \in [\![1,n]\!]}, \{R_{H_i}^*\}_{i \in [\![1,n]\!]}, \{Q_\kappa\}_{\kappa \in \mathcal{K}}\big)$
$b_* := D(\mathsf{view})$
return b_*

Figure 7.18: Adversary $\mathcal{A}$ for the proof of Lemma 39.

distinguisher returns 1 on inputs computed as in real protocol. Finally, we evaluate the probability that $\mathcal{A}$ wins the PRF experiment

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

which is non-negligible. However, we assume that F is a secure pseudo-random function, hence, it does not exist D such that:

$$\big|\Pr[D(\mathcal{S}^{H\text{-SHYP}}_{\mathcal{C}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] - \Pr[D(\mathsf{view}^{\mathrm{SHYP}}_{\mathcal{C}}(I, \lambda)) = 1]\big|\ ,$$

is non-negligible. Hence, we have

$$\mathcal{S}^{H\text{-SHYP}}_{\mathcal{C}}(I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I, \lambda)) \overset{\mathrm{c}}{\equiv} \mathsf{view}^{\mathrm{SHYP}}_{\mathcal{C}}(I, \lambda)\ .$$

Finally, we show how to build the simulator $\mathcal{S}^{\mathrm{SHYP}}_{\mathcal{C}}$. The difference between $\mathcal{S}^{H\text{-SHYP}}_{\mathcal{C}}$ and $\mathcal{S}^{\mathrm{SHYP}}_{\mathcal{C}}$ is that $\mathcal{S}^{\mathrm{SHYP}}_{\mathcal{C}}$ substitutes Π encryption of real values by Π encryption of random values of the same size. However, simulator $\mathcal{S}^{H\text{-SHYP}}_{\mathcal{C}}$ still picks random values,

hence we do not formally present $\mathcal{S}_{\mathcal{C}}^{\text{SHYP}}$ and assume that random values are rewritten by other random values.

Now we show that we have

$$\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \stackrel{\text{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{\text{SHYP}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \ .$$

Let λ be the security parameter. Assume there exists a distinguisher $D \in \text{PPT}(\lambda)$ such that for all inputs I, we have

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

where $\mu(\cdot)$ is a non-negligible function in λ.

We construct a guessing adversary $\mathcal{B} \in \text{PPT}(\lambda)$ that uses D to win the IND-CPA experiment. Adversary $\mathcal{B}$ is presented in Figure 7.19.

Adversary: $\mathcal{B}(pk)$

for $i \in [\![1, |\bar{T}|]\!]$ **do**
  $t_i \xleftarrow{\$} \bigtimes_{A \in \mathbb{X}} A$
$\mathsf{visited} := \emptyset$
for $i \in [\![1, n]\!]$ **do**
  $R_i^* := \emptyset$
  $R_{H_i}^* := \emptyset$
  $\mathbb{R}_i^{\mathcal{E}} := \{A \colon A \in \mathbb{R}_i \setminus \mathsf{visited}\}$
  for $j \in [\![1, |\bar{T}|]\!]$ **do**
    $t_f := \bigtimes_{A \in \mathbb{R}_i^f} f_b(\pi_A(t_j))$
    for $\ell \in [\![1, d]\!]$ **do**
      if $X_\ell \in \mathbb{R}_i^f$ **then**
        $x_\ell := h_\ell(\pi_{X_\ell}(t_f))$
      else
        $x_\ell := \star$
    $\mathcal{K} := \mathcal{K} \cup \{(x_1, \ldots, x_d)\}$
    for $l \in [\![1, T_j[i-1]]\!]$ **do**
      $t_1 \xleftarrow{\$} \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} A$
      $t_2 \xleftarrow{\$} \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} A$
      $t_{\mathcal{E}} := \bigtimes_{A \in \mathbb{R}_i^{\mathcal{E}}} \mathcal{E}(pk, \mathsf{LoR}_b(\pi_A(t_1), \pi_A(t_2))$
      $R_i^* := R_i^* \cup \{t_f \times t_{\mathcal{E}}\}$
      $R_{H_i}^* := R_{H_i}^* \cup \{((x_1, \ldots, x_\ell), (\mathtt{R}_i, t_f \times t_{\mathcal{E}}))\}$,
  $\mathsf{visited} := \mathsf{visited} \cup \mathbb{R}_i$
foreach $\kappa \in \mathcal{K}$ **do**
  for $i \in [\![1, n]\!]$ **do**
    $\bar{R}_i^* := \bigcup_{(\kappa, (\mathtt{R}_i, t)) \in \cup_{j \in [\![1,n]\!]} R_i^*} \{t\}$
  $Q_\kappa := \bar{R}_1^* \bowtie \cdots \bowtie \bar{R}_n^*$
$\mathsf{view} := \left(\{R_i^*\}_{i \in [\![1,n]\!]}, \{R_{H_i}^*\}_{i \in [\![1,n]\!]}, \{Q_\kappa\}_{\kappa \in \mathcal{K}}\right)$
$b_* := D(\mathsf{view})$
return b_*

Figure 7.19: Adversary $\mathcal{B}$ for the proof of Lemma 39.

First, we remark that

$$\Pr[\mathsf{Exp}_{\Pi,\mathcal{B}}^{\text{indcpa-0}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \ .$$

When $b = 0$, the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the

probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}$. On the other hand, we have

$$\Pr[\mathsf{Exp}_{\Pi,\mathcal{B}}^{\text{indcpa-1}}(\lambda) = 1] = \Pr[D(\mathcal{S}_{\mathcal{C}}^{\text{SHYP}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I))) = 1] \ .$$

Indeed, when $b = 1$ the view that $\mathcal{B}$ uses as input for D is computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{\text{SHYP}}$. Then the probability that the IND-CPA experiment returns 1 is equal to the probability that the distinguisher D returns 1 on inputs computed as in the simulator $\mathcal{S}_{\mathcal{C}}^{\text{SHYP}}$.

Finally, we evaluate the probability that $\mathcal{B}$ wins the experiment

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

which is non-negligible. However, we assumed that Π is IND-CPA. Hence, we have

$$\mathcal{S}_{\mathcal{C}}^{\text{SHYP}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \stackrel{\text{c}}{\equiv} \mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \ .$$

By transitivity, we have

$$\mathcal{S}_{\mathcal{C}}^{\text{SHYP}}(1^\lambda, I_{\mathcal{C}}, g_{I_{\mathcal{C}}}(I)) \stackrel{\text{c}}{\equiv} \mathsf{view}_{\mathcal{C}}^{\text{SHYP}}(I, \lambda) \ ,$$

which concludes the proof. □

Lemma 40. *There exists a probabilistic polynomial-time simulator* $\mathcal{S}_{\mathcal{U}}^{\text{SHYP}}$ *such that*

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

The proof for Lemma 40 is the same than the proof for Lemma 37 on page 146.

7.6 Conclusion

We presented an efficient secure approach for computing joins with MapReduce. This secure approach, namely Secure-Private (SP), is applied on the two state-of-the-art methods to compute natural join, i.e., the cascade [LRU14] and the hypercube [AU10] methods. This secure approach assumes that the public cloud and the MapReduce's user do not collude. We have thoroughly compared this secure approach with respect to their privacy guarantees and their practical performance using a standard real-world dataset.

As future work, we plan to integrate our secure join protocols in a secure query optimizer system based on the MapReduce paradigm. We also aim at designing a protocol that is secure in the standard model, and that it remains secure in case of collusion between the public cloud and the MapReduce's user. Finally, we would like to investigate the use of Apache Spark® [ASF19] framework to study the practical impact on computation performances, specifically on the wall clock time.

CHAPTER 8

Conclusion

In this thesis, we have presented several secure distributed protocols based on the MapReduce programming model. These protocols have a significant impact for practical applications of cloud computing since they allow computation over encrypted data and therefore preserve the user's confidentiality. Indeed, using insecure protocols, a public cloud can learn all information that is in transit. On the other hand, these protocols are practical with regard to current cloud computing platforms (e.g., Amazon EMR [Ama19]) that propose to run big data frameworks such as Apache Hadoop.

Most of our protocols rely on the additive-homomorphic Paillier's cryptosystem [Pai99]. Indeed, if huge progress has been realized on the practical efficiency of homomorphic encryption schemes, they remain slower than partial homomorphic encryption schemes as observed in Section 1.6.1 of this thesis. We model each of our protocol as a multiparty protocol. We have considered one or several data owners (depending on the protocol) outsourcing their respective data into a semi-honest public cloud that performs calculation for an external MapReduce's user.

Summary

We sum up the contributions presented in this thesis.

Contributions of the First Part

We first recall the contributions of the first part of this thesis in which we consider the matrix multiplication using the MapReduce paradigm.

Secure Standard Matrix Multiplication. In Chapter 3, we recalled the two state-of-the-art protocols to compute matrix multiplication with the MapReduce paradigm that has been proposed by Leskovec et al. [LRU14]. The first protocol named MM-1R uses one MapReduce round while the second protocol uses two MapReduce rounds. We have defined two secure approaches based on the additive-homomorphic Paillier's cryptosystem [Pai99].

The first one is called Secure-Private (SP), it assumes that the public cloud nodes do not collude, i.e., they do not share their information. Following the SP approach, we have designed two protocols based on MM-1R and MM-2R protocols, respectively called SP-1R and SP-2R. The protocol SP-1R considers the one-sided model, i.e., only one of the two matrices sent to the public cloud is encrypted. The protocol SP-2R considers

the two-sided model where both matrices sent to the public cloud are encrypted. In this second protocol a set of nodes have to decrypt random masks, used to protect one of the two matrices. However, since the SP approach assumes that public cloud nodes do not collude, others sets of nodes of the public cloud do not learn these elements.

The second proposed approach is called Collision-Resistant-Secure-Private (CRSP). Contrary to the SP approach, it assumes that public cloud's nodes may collude, i.e., they can share all their information. Based on this approach and the two original protocols MM-1R and MM-2R, we designed the CRSP-1R protocol using one MapReduce round and the CRSP-2R protocol using two MapReduce rounds. Our two CRSP protocols consider the two-sided model. In order to allow the public cloud to compute needed Paillier's ciphertexts multiplications for the matrix multiplication, our CRSP protocols use the Paillier interactive multiplicative homomorphic protocol proposed by Cramer et al. [CDN01] applied between the public cloud and the MapReduce's user.

Secure Strassen-Winograd Matrix Multiplication. In Chapter 4, we focus on the Strassen-Winograd algorithm, one of the most efficient algorithms to compute matrix multiplication. First, we have proposed a MapReduce protocol for the Strassen-Winograd matrix multiplication. The protocol, denoted SM3, is made of two phases: the deconstruction phase, and the combination phase. Like the original Strassen-Winograd algorithm, our MapReduce protocol requires that the two matrices to be multiplied to have 2-power integer size. We have also proposed two variants using the dynamic padding and the dynamic peeling methods that allow to perform Strassen-Winograd matrix multiplication with compatible matrices of arbitrary size using the MapReduce paradigm.

Moreover, we have proposed a secure approach of the SM3 protocol (denoted S2M3) and of its variants. We have supposed that the public cloud's nodes may collude. On the contrary, we require that the MapReduce's user does not collude with the public cloud or the data owners. The secure protocol S2M3 considers the two-sided model where both matrices to multiply are encrypted. In order to compute the needed Paillier's ciphertexts multiplications, we also rely on the Paillier interactive multiplicative homomorphic protocol [CDN01].

For both standard matrix multiplication and Strassen-Winograd matrix multiplication, we have proved that our protocols securely compute matrix multiplication in the presence of semi-honest adversaries in the standard model.

Contributions of the Second Part

We now recall the contributions of the second part that focus on relational-algebra operations using the MapReduce paradigm.

Secure Intersection with MapReduce. We have considered in Chapter 5 the intersection of relations in the MapReduce paradigm. We started by recalling the state-of-the-art MapReduce protocol proposed by Leskovec et al. [LRU14]. Then, we have proposed a secure approach denoted SI allowing a public cloud to perform intersection between $n \geq 2$ relations following the MapReduce paradigm. Then, the result of the computation is sent to the MapReduce's user. Our protocol SI relies on standard cryptographic primitives such as asymmetric encryption scheme, pseudo-random functions and the bitwise exclusive OR operator.

We also prove in the random oracle model that our protocol remains secure even if the user and the public cloud collude, i.e., they share all their information even the private one.

Secure Grouping and Aggregation with MapReduce. In Chapter 6 we studied the grouping and aggregation operation on several relations. More specifically, we focus on the COUNT, SUM, AVG, MIN, and MAX aggregations. MapReduce protocols for these aggregations have been proposed by Leskovec et al. [LRU14]. We have proposed a secure approach for each of these five aggregations assuming that the public cloud and the MapReduce's user do not collude.

Aggregations COUNT, SUM, and AVG rely on the additive-homomorphic property of Paillier's cryptosystem [Pai99], while aggregations MIN and MAX rely on order-preserving symmetric encryption [AKSX04].

Secure Join with MapReduce. Finally, we focus in Chapter 7 on the computation of natural join between an arbitrary number of relations. We have first presented the two standard algorithms of join computation for MapReduce: the *cascade* algorithm [LRU14], and the *hypercube* algorithm [AU10].

Thereafter, we have designed for both algorithms a secure approach. Our secure approach assumes that the MapReduce's user and the public cloud do not collude, and only requires as cryptographic primitives a pseudo-random function and an asymmetric encryption scheme.

Finally, we have realized implementations of all designed protocols using the Apache Hadoop framework [Fou19b] to demonstrate their feasibility.

Future Work

We propose possible research directions that stem from the results presented in this thesis.

Generally speaking, all presented protocols consider a semi-honest public cloud, i.e., it dutifully follows protocol rules but tries to learn as much information as possible about data and the result of the computation. We would like to adjust our protocols to a *malicious* public cloud that is allowed to perform any operations on the data.

Furthermore, it would be interesting to investigate the use of different big data open-source frameworks such as Apache Spark™ [ASF19] or Apache Flink® [Fou19a] that also implement the MapReduce programming model in order to study the practical impact on computation performances. The main difference between Spark and Hadoop is Spark can process data in RAM while Hadoop has to read from and write to a disk. As a result, the speed of processing differs significantly, e.g., Spark may be up to 100 times faster. In the case of Strassen-Winograd matrix multiplication or natural join computation demanding several MapReduce rounds, Spark may be the best framework since it avoids disk operations for each MapReduce round. However, the volume of data processed also differs: Hadoop is able to work with far larger data sets than Spark. Moreover, Flink can provide a purer stream-processing capability with lower latency compared to Hadoop that can only batch process one job at a time. Hence, it would be interesting to observe practical impact depending on the used framework for the same computation.

Matrix Multiplication. During the first part of this thesis, we have considered matrix multiplication with standard and Strassen-Winograd algorithms. The main open problem is that we currently assume that the MapReduce's user and the public cloud do not collude. Otherwise, the public cloud may learn both matrices and their product using the secret key of the MapReduce's user. On the user-side, she may learn both original matrices in addition to the result corresponding to the matrix multiplication.

For both standard matrix multiplication and Strassen-Winograd matrix multiplication with MapReduce, the Paillier interactive multiplicative homomorphic protocol implies an

important communication cost. This problem can be solved by using somewhat homomorphic encryption schemes such that the Brakerski/Fan-Vercauteren scheme [FV12b] implemented in the `SEAL` library [MR19]. However, the latter remains more computationnally expensive for multiple matrix multiplications than the use of Paillier's cryptosystem. A very interesting solution would be to design a MapReduce protocol using the packing method for secure matrix multiplication proposed by Duong et al. [DMY16]. Such a packing method allows to pack a matrix into a single ciphertext so that it also enables efficient matrix multiplication over the packed ciphertexts. However, it would require very large parameters for matrices used in the big data world.

Relational-Algebra Operations. In the second part of this thesis, we dealt with relational-algebra operations such as intersection, grouping and aggregation, and natural join. All proposed protocols are considered in the Random Oracle Model since they require pseudo-random functions. This security model is not optimal and it would be interesting to design equivalent protocols in the standard model. Pseudo-random functions are used to correctly map each key-value pair to the corresponding reducer. This implies possible frequency-count attacks that we would like to prevent.

Moreover, security for our protocols for grouping and aggregation, with COUNT, SUM, AVG, MIN, and MAX aggregations, as well as for our protocols for natural join computation do not resist in case of collusion between the public cloud and the MapReduce's user. Indeed, if the public cloud knows the secret key of the MapReduce's user, it can decrypt all encrypted data sent by data owners that has been encrypted using the public key of the MapReduce's user. We would like to design protocols such as the one presented for the intersection computation that remains secure even if the public cloud and the MapReduce's user collude.

In conclusion, the works presented in this manuscript solve some concrete problems of secure distributed computations in the cloud. Nevertheless, some protocols can still be improved in terms of security and efficiency. Thereby, this thesis leaves some open problems and future works perspectives in the field of cloud cryptography for distributed computations.

List of Figures

1.1 Relation *Links*(*From*, *To*) consists of the set of pairs of URLs, such that the first has one or more links to the second. . . . 4
1.2 Schematic of a MapReduce computation. . . . 5
1.3 Running time vs the number of matrix multiplications using Paillier's cryptosystem [Pai99] and the BFV [FV12a] scheme. . . . 8

2.1 PRF experiment. . . . 19
2.2 IND-OCPA experiment. . . . 20
2.3 IND-CPA experiment. . . . 22
2.4 Paillier interactive multiplicative homomorphic protocol [CDN01]. . . . 24

3.1 One Round. . . . 29
3.2 Two Rounds. . . . 29
3.3 Map and Reduce functions for MM-2R protocol. . . . 31
3.4 Map and Reduce functions for the MM-1R protocol. . . . 32
3.5 Preprocessing algorithms for secure matrix multiplication. . . . 34
3.6 Map and Reduce functions for the SP-2R protocol. . . . 34
3.7 Map and Reduce functions for SP-1R protocol. . . . 36
3.8 Map and Reduce functions for the CRSP-2R protocol. . . . 36
3.9 Map and Reduce functions for the CRSP-1R protocol. . . . 37
3.10 CPU time vs the matrices' dimension for the two state-of-the-art protocols [LRU14] computing the matrix multiplication. . . . 39
3.11 CPU time vs the matrices' dimension for our SP and CRSP approaches computing the matrix multiplication. . . . 39
3.12 Simulator $\mathcal{S}_{\mathcal{C}}^{\text{CRSP}}$ for the proof of Lemma 2. . . . 41
3.13 Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 2. . . . 42
3.14 Simulator $\mathcal{S}_{\mathcal{P}}^{\text{CRSP}}$ for the proof of Lemma 3. . . . 43
3.15 Simulator $\mathcal{S}_{\mathcal{M}}^{\text{SP-2R}}$ for the proof of Lemma 5. . . . 45
3.16 Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 5. . . . 45
3.17 Simulator $\mathcal{S}_{\mathcal{N}}^{\text{SP-2R}}$ for the proof of Lemma 6. . . . 46
3.18 Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 6. . . . 46
3.19 Simulator $\mathcal{S}_{\mathcal{R}_1}^{\text{SP-2R}}$ for the proof of Lemma 7. . . . 47
3.20 Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot)),\mathcal{E}(pk_{\mathcal{R}_2},\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 7. . . . 48
3.21 Simulator $\mathcal{S}_{\mathcal{R}_2}^{\text{SP-2R}}$ for the proof of Lemma 8. . . . 49
3.22 Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot)),\mathcal{E}(pk_{\mathcal{R}_2},\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 8. . . . 50
3.23 Simulator $\mathcal{S}_{\mathcal{P}}^{\text{SP-2R}}$ for the proof of Lemma 9. . . . 50

3.24 Simulator $\mathcal{S}_{\mathcal{R}}^{\text{SP-1R}}$ for the proof of Lemma 14. 52
3.25 Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot)),\mathcal{E}(pk_{\mathcal{R}_2},\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 14. 53

4.1 Architecture of Strassen-Winograd matrix multiplication with MapReduce. 57
4.2 Dynamic peeling for matrices M and N. 60
4.3 Map function for the deconstruction phase of the SM3 protocol. 61
4.4 Reduce function for the deconstruction phase of the SM3 protocol. 61
4.5 Reduce function for the combination phase of the SM3 protocol. 62
4.6 Map function for the deconstruction phase of the SM3-Pad protocol. 63
4.7 Reduce function for the deconstruction phase of the SM3-Pad protocol. . . 64
4.8 Reduce function for the combination phase of the SW-Pad protocol. 65
4.9 Reduce function for the deconstruction phase of the SW-Peel protocol. . . . 66
4.10 Reduce function for the combination phase of the SW-Peel protocol. 68
4.11 Reduce function for the deconstruction phase of the S2M3 protocol. 70
4.12 Reduce function for the combination phase of the S2M3 protocol. 71
4.13 Reduce function for the deconstruction phase of the S2M3-Pad protocol. . . 72
4.14 Reduce function for the combination phase of the S2M3-Pad protocol. . . . 73
4.15 Reduce function for the deconstruction phase of the S2M3-Peel protocol. . . 74
4.16 Reduce function for the combination phase of the S2M3-Peel protocol. . . . 75
4.17 CPU time vs order of matrices for the state-of-the-art MM-1R protocol using one MapReduce round [LRU14] and for our SM3 protocols using static and dynamic padding methods, and dynamic peeling method. 76
4.18 CPU time vs order of matrices for the CRSP-1R protocol presented in Chapter 3 and for our S2M3 protocols using static and dynamic padding methods, and dynamic peeling method. 77
4.19 Function SW-Deconstruction for simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}$ presented in Figure 4.21. . 78
4.20 Function SW-Combination for simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}$ presented in Figure 4.21. . . 79
4.21 Simulator $\mathcal{S}_{\mathcal{C}}^{\text{S2M3}}$ for the proof of Lemma 17. 79
4.22 Adversary $\mathcal{A}^{\mathcal{E}(pk,\mathsf{LoR}_b(\cdot,\cdot))}$ for the proof of Lemma 17. 80
4.23 Simulator $\mathcal{S}_{\mathcal{P}}^{\text{S2M3}}$ for the proof of Lemma 18. 81

5.1 Relations NSA, GCHQ, and Mossad. 88
5.2 Example of intersection with MapReduce between three relations. First, data owners outsource their respective relation on the public cloud. The public cloud runs the Map function, then the Reduce function verifies if a key is associated to a list of three values. If that is the case, the public cloud produces and sends tuples corresponding to the intersection to the user. 89
5.3 The system architecture. 90
5.4 MapReduce protocol to compute the intersection of n relations. 91
5.5 Preprocessing algorithm of SI protocol. 92
5.6 Map and Reduce functions of our secure approach SI. 93
5.7 Encrypted relations NSA*, GCHQ*, and Mossad* after the preprocessing phase of our secure protocol SI. 94
5.8 Example of intersection with MapReduce between three relations using our secure protocol SI. First, data owners outsource their respective encrypted relation on the public cloud. The public cloud runs the Map function, then the Reduce function verifies if keys are associated to a list of three values. In that case, the public cloud produces and sends encrypted tuples corresponding to the intersection to the user. In the example, the user obtains the tuples $(-, \mathcal{E}(pk, \texttt{F654}))$ equals to $(-, (\mathcal{E}(pk, \texttt{F654}) \oplus (\oplus_{j=2}^{3} F(k_j, \texttt{F654})) \oplus (\oplus_{j=2}^{3} F(k_j, \texttt{F654}))))$. 94

5.9 CPU time vs the number of tuples for the standard MapReduce protocol [LRU14] and our secure approach SI computing the intersection between two relations. . . . 96
5.10 CPU time vs the number of intersected relations for the standard MapReduce protocol [LRU14] and our secure approach SI computing the intersection between two relations. . . . 97
5.11 Random oracle OPRF. . . . 99
5.12 Simulator $\mathcal{S}_{\mathcal{C},\mathcal{U}}^{H_1-\mathrm{SI}}$ for the proof of Lemma 27. . . . 99
5.13 Adversary $\mathcal{A}(pk)$. . . . 100
5.14 Simulator $\mathcal{S}_{\mathcal{C},\mathcal{U}}^{\mathrm{SI}}$ for the proof of Lemma 27. . . . 101
5.15 Adversary $\mathcal{B}$ for the proof of Lemma 27. . . . 102

6.1 Relation R. . . . 107
6.2 Result of $\gamma_{\mathrm{Department,SUM(Salary)}}(R)$. . . . 107
6.3 The system architecture. . . . 108
6.4 Map and Reduce functions for the COUNT protocol. . . . 110
6.5 Map and Reduce functions for the SUM protocol. . . . 111
6.6 Map and Reduce functions for the AVG protocol. . . . 111
6.7 Map and Reduce functions for the MIN protocol. . . . 112
6.8 Preprocessing algorithm of SGA protocols. . . . 113
6.9 Map and Reduce functions for the $\mathrm{SGA}_{\mathrm{COUNT}}$ protocol. . . . 114
6.10 Map and Reduce functions for the $\mathrm{SGA}_{\mathrm{SUM}}$ protocol. . . . 115
6.11 Map and Reduce functions for the $\mathrm{SGA}_{\mathrm{AVG}}$ protocol. . . . 115
6.12 Map and Reduce functions for the $\mathrm{SGA}_{\mathrm{MIN}}$ protocol. . . . 116
6.13 CPU time vs the number of tuples for no-secure and secure protocols which perform grouping and aggregation with the COUNT, SUM, AVG, and MIN operations. . . . 119
6.14 Random oracle OPRF. . . . 121
6.15 Simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-}\mathrm{SGA}_{\mathrm{SUM}}}$ for the proof of Lemma 30. . . . 121
6.16 Adversary $\mathcal{A}$ for the proof of Lemma 30. . . . 122
6.17 Simulator $\mathcal{S}_{\mathcal{C}}^{\mathrm{SGA}_{\mathrm{SUM}}}$ for the proof of Lemma 30. . . . 122
6.18 Adversary $\mathcal{B}$ for the proof of Lemma 30. . . . 123
6.19 Simulator $\mathcal{S}_{\mathcal{C}}^{H_1\text{-}\mathrm{SGA}_{\mathrm{MIN}}}$ for the proof of Lemma 33. . . . 125
6.20 Adversary $\mathcal{A}$ for the proof of Lemma 33. . . . 126
6.21 Simulator $\mathcal{S}_{\mathcal{C}}^{H_2\text{-}\mathrm{SGA}_{\mathrm{MIN}}}$ for the proof of Lemma 33. . . . 127
6.22 Adversary $\mathcal{B}$ for the proof of Lemma 33. . . . 127
6.23 Adversary $\mathcal{D}$ for the proof of Lemma 33. . . . 129

7.1 The system architecture. . . . 132
7.2 Joins between relations R_1, R_2 and R_3. . . . 132
7.3 Cascade of joins with MapReduce between n relations. . . . 133
7.4 Running example with hypercube protocol. Underlined tuples correspond to tuples that participate to the final join result. . . . 134
7.5 Map and Reduce functions for CAS protocol. . . . 136
7.6 Map and Reduce functions for HYP protocol. . . . 137
7.7 Preprocessing of relations. . . . 138
7.8 Intuition of the secure approach of the cascade protocol. We denote ciphertexts of an IND-CPA encryption scheme by $\{\cdot\}$, and pseudo-random evaluations by integers. . . . 139
7.9 Map and Reduce functions for SCAS protocol. . . . 139
7.10 Map and Reduce functions for SHYP protocol. . . . 140

7.11 CPU time vs the number of tuples per relation for CAS and HYP protocols, and their respective secure approach. 141
7.12 Random oracle OPRF. 143
7.13 Simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SCAS}}$ for the proof of Lemma 36. 144
7.14 Adversary $\mathcal{A}$ for the proof of Lemma 36. 144
7.15 Adversary $\mathcal{B}$ for the proof of Lemma 36. 146
7.16 Simulator $\mathcal{S}_{\mathcal{U}}^{\text{SCAS}}$ for the proof of Lemma 37. 146
7.17 Simulator $\mathcal{S}_{\mathcal{C}}^{H\text{-SHYP}}$ for the proof of Lemma 39. 148
7.18 Adversary $\mathcal{A}$ for the proof of Lemma 39. 149
7.19 Adversary $\mathcal{B}$ for the proof of Lemma 39. 150

List of Tables

1.1 Parameters for the BFV scheme to compute multiple matrix multiplications. 8

3.1 Computation and communication costs (big-$\mathcal{O}$) analysis of MapReduce matrix multiplication protocols. Let $C_\times$ (resp. C_+, C_{exp}, $C_\mathcal{E}$, $C_\mathcal{D}$, C_{inv}, $C_\$$) be the cost of multiplication (resp. addition, exponentiation, encryption, decryption, inversion, sampling). 38

5.1 Summary of results. Let $N = \max(|R_1|, \ldots, |R_n|)$ be the biggest cardinal of relations R_i with $i \in [\![1, n]\!]$. Let $C_\oplus$ be the computation cost of a bitwise exclusive OR operation). 95

6.1 Complexity of original MapReduce grouping and aggregation protocols and of our SGA protocols. Let N_1 be the number of tuples of the relation R. Let $\mathcal{C}_+$ (resp. $\mathcal{C}_{\times_{\mathsf{mod}}}$, $\mathcal{C}_\mathcal{E}$, $\mathcal{C}_\div$, $\mathcal{C}_{\mathsf{comp}}$) be the cost of addition (resp. modular multiplication, asymmetric encryption, division, comparison). 117

6.2 Wall clock times of the secure protocols preprocessing. 119

7.1 Wall clock times of the secure protocols preprocessing. 141

Bibliography

[ABGL17] Alexandre Anzala-Yamajako, Olivier Bernard, Matthieu Giraud, and Pascal Lafourcade. No Such Thing as a Small Leak: Leakage-Abuse Attacks Against Symmetric Searchable Encryption. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE)*, pages 253–277, 2017. https://doi.org/10.1007/978-3-030-11039-0_12.

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic Encryption Security Standard. Technical report, November 2018. http://homomorphicencryption.org/standard/.

[AE07] Artak Amirbekyan and Vladimir Estivill-Castro. A New Efficient Privacy-Preserving Scalar Product Protocol. In *Proceedings of the 6th Australasian Data Mining Conference (AusDM)*, pages 209–214, 2007. http://crpit.com/abstracts/CRPITV70Amirbekyan.html.

[AKSX04] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-Preserving Encryption for Numeric Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–574, 2004. https://doi.org/10.1145/1007568.1007632.

[Ama19] Inc. Amazon.com. Amazon EMR. https://aws.amazon.com/emr/, 2019.

[ASF19] Databricks Apache Software Foundation, UC Berkley AMPLap. Apache Spark (release 2.4.3). https://spark.apache.org/, May 2019.

[AU10] Foto N. Afrati and Jeffrey D. Ullman. Optimizing Joins in a Map-Reduce Environment. In *Proceedings of the 13th International Conference on Extending Database Technology (EDBT)*, pages 99–110, 2010. https://doi.org/10.1145/1739041.1739056.

[AWK17] Wael Y. Alghamdi, Hui Wu, and Salil S. Kanhere. Reliable and Secure End-to-End Data Aggregation Using Secret Sharing in WSNs. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2017. https://doi.org/10.1109/WCNC.2017.7925558.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-Key Encryption in a Multi-user Setting: Security Proofs and Improvements. In *Proceedings of*

the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT), pages 259–274, 2000. https://doi.org/10.1007/3-540-45539-6_18.

[BCG+18] Xavier Bultel, Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, and Lihua Ye. Secure Joins with MapReduce. In *Proceedings of the 11th International Symposium on Foundations and Practice of Security (FPS)*, pages 78–94, 2018. https://doi.org/10.1007/978-3-030-18419-3_6.

[BCGL17] Xavier Bultel, Radu Ciucanu, Matthieu Giraud, and Pascal Lafourcade. Secure Matrix Multiplication with MapReduce. In *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES)*, pages 11:1–11:10, 2017. https://doi.org/10.1145/3098954.3098989.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Proceedings of the 16th Annual International Cryptology Conference (CRYPTO)*, pages 1–15, 1996. https://doi.org/10.1007/3-540-68697-5_1.

[BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-Preserving Symmetric Encryption. In *Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 224–241, 2009. https://doi.org/10.1007/978-3-642-01001-9_13.

[BDG+17] Xavier Bultel, Manik Lal Das, Hardik Gajera, David Gérault, Matthieu Giraud, and Pascal Lafourcade. Verifiable Private Polynomial Evaluation. In *Proceedings of the 11th International Conference on Provable Security (ProvSec)*, pages 487–506, 2017. https://doi.org/10.1007/978-3-319-68637-0_29.

[BDG+18] Xavier Bultel, Jannik Dreier, Matthieu Giraud, Marie Izaute, Timothée Kheyrkhah, Pascal Lafourcade, Dounia Lakhzoum, Vincent Marlin, and Ladislav Moták. Security Analysis and Psychological Study of Authentication Methods with PIN Codes. In *Proceedings of the 12th International Conference on Research Challenges in Information Science (RCIS)*, pages 1–11, 2018. https://doi.org/10.1109/RCIS.2018.8406648.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS)*, pages 309–325, 2012. https://doi.org/10.1145/2090236.2090262.

[BIK+17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1175–1191, 2017. https://doi.org/10.1145/3133956.3133982.

[BLGR17] Mouhebeddine Berrima, Pascal Lafourcade, Matthieu Giraud, and Narjes Ben Rajeb. Formal Analyze of a Private Access Control Protocol to a Cloud Storage. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE)*, pages 495–500, 2017. https://doi.org/10.5220/0006461604950500.

[BLLN13] Joppe W. Bos, Kristin E. Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In *Proceedings of the 14th IMA International Conference on Cryptography and Coding (IMACC)*, pages 45–64, 2013. https://doi.org/10.1007/978-3-642-45239-0_4.

[BPMÖ12] Erik-Oliver Blass, Roberto Di Pietro, Refik Molva, and Melek Önen. PRISM - Privacy-Preserving Search in MapReduce. In *Proceedings of the 12th International Symposium on Privacy Enhancing Technologies (PETS)*, pages 180–200, 2012. https://doi.org/10.1007/978-3-642-31680-7_10.

[BR93] Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS)*, pages 62–73, 1993. https://doi.org/10.1145/168588.168596.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal Asymmetric Encryption. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 92–111, 1994. https://doi.org/10.1007/BFb0053428.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 97–106, 2011. https://doi.org/10.1109/FOCS.2011.12.

[CBS15] Shumo Chu, Magdalena Balazinska, and Dan Suciu. From Theory to Practice: Efficient Join Query Evaluation in a Parallel Database System. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 63–78, 2015. https://doi.org/10.1145/2723372.2750545.

[CDF+07] Jon Callas, Lutz Donnerhacke, Hal Finney, David Shaw, and Rodney Thayer. OpenPGP Message Format. RFC 4880, RFC Editor, November 2007. https://tools.ietf.org/rfc/rfc4880.txt.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 280–299, 2001. https://doi.org/10.1007/3-540-44987-6_18.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *Proceedings of the 22nd International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 3–33, 2016. https://doi.org/10.1007/978-3-662-53887-6_1.

[CGLY18] Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, and Lihua Ye. Secure Grouping and Aggregation with MapReduce. In *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications (ICETE)*, pages 514–521, 2018. https://doi.org/10.5220/0006843805140521.

[CGLY19a] Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, and Lihua Ye. Secure Intersection with MapReduce. In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE)*, 2019.

[CGLY19b] Radu Ciucanu, Matthieu Giraud, Pascal Lafourcade, and Lihua Ye. Secure Strassen-Winograd Matrix Multiplication with MapReduce. In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE)*, 2019.

[CGPR15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-Abuse Attacks Against Searchable Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 668–679, 2015. https://doi.org/10.1145/2810103.2813700.

[Cha12] Gary Chartrand. *Introductory Graph Theory*. Dover Books on Mathematics. Dover Publications, 2012. ISBN: 9780486247755.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*, pages 409–437, 2017. https://doi.org/10.1007/978-3-319-70694-8_15.

[CKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 213–231, 2010. https://doi.org/10.1007/978-3-642-17373-8_13.

[CLS09] Sherman S. M. Chow, Jie-Han Lee, and Lakshminarayanan Subramanian. Two-Party Computation Model for Privacy-Preserving Queries over Distributed Databases. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2009. https://www.ndss-symposium.org/ndss2009/two-party-computation-model-privacy-preserving-queries-over-distributed-datab

[CS98] Ronald Cramer and Victor Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *Proceedings of the 18th Annual International Cryptology Conference (CRYPTO)*, pages 13–25, 1998. https://doi.org/10.1007/BFb0055717.

[DA01] Wenliang Du and Mikhail J. Atallah. Privacy-Preserving Cooperative Statistical Analysis. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC)*, pages 102–110, 2001. https://doi.org/10.1109/ACSAC.2001.991526.

[Dan15] Quynh H. Dang. Secure hash standard. *Federal Inf. Process. Stds.*, 2015.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When Private Set Intersection Meets Big Data: An Efficient and Scalable Protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 789–800, 2013. https://doi.org/10.1145/2508859.2516701.

[DDC16] F. Betül Durak, Thomas M. DuBuisson, and David Cash. What Else is Revealed by Order-Revealing Encryption? In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1155–1166, 2016. https://doi.org/10.1145/2976749.2978379.

[DDGS16] Philip Derbeko, Shlomi Dolev, Ehud Gudes, and Shantanu Sharma. Security and Privacy Aspects in MapReduce on Clouds: A Survey. *Computer Science Review*, 20:1–28, May 2016. https://doi.org/10.1016/j.cosrev.2016.05.001.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, pages 137–150, 2004. https://www.usenix.org/legacy/events/osdi04/tech/dean.html.

[DGL15] Shlomi Dolev, Niv Gilboa, and Ximing Li. Accumulating Automata and Cascaded Equations Automata for Communicationless Information Theoretically Secure Multi-Party Computation: Extended Abstract. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing (SCC@ASIACCS)*, pages 21–29, 2015. https://doi.org/10.1145/2732516.2732526.

[DH76] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. https://doi.org/10.1109/TIT.1976.1055638.

[DLF+19] Jean-Guillaume Dumas, Pascal Lafourcade, Julio Lopez Fenner, David Lucas, Jean-Baptiste Orfila, Clément Pernet, and Maxime Puys. Secure Multiparty Matrix Multiplication Based on Strassen-Winograd Algorithm. In *Proceedings of the 14th International Workshop on Security (IWSEC)*, 2019.

[DLMM] Manlio De Domeniceo, Antonio Lima, Paul Mougel, and Mirco Musolesi. Higgs twitter dataset. http://snap.stanford.edu/data/higgs-twitter.html. Accessed: 2018-02-20.

[DLMM13] Manlio De Domeniceo, Antonio Lima, Paul Mougel, and Mirco Musolesi. The Anatomy of a Scientific Rumor, 2013.

[DLOP16] Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. Private Multi-party Matrix Multiplication and Trust Computations. In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE)*, pages 61–72, 2016. https://doi.org/10.5220/0005957200610072.

[DLOP17] Jean-Guillaume Dumas, Pascal Lafourcade, Jean-Baptiste Orfila, and Maxime Puys. Dual Protocols for Private Multi-Party Matrix Multiplication and Trust Computations. *Computers & Security*, 71:51–70, November 2017. https://doi.org/10.1016/j.cose.2017.04.013.

[DLS16] Shlomi Dolev, Yin Li, and Shantanu Sharma. Private and Secure Secret Shared MapReduce (Extended Abstract). In *Proceedings of the 30th Annual Conference on Data and Applications Security and Privacy (DBSec)*, pages 151–160, 2016. https://doi.org/10.1007/978-3-319-41483-6_11.

[DMY16] Dung Hoang Duong, Pradeep Kumar Mishra, and Masaya Yasuda. Efficient secure matrix multiplication over lwe-based homomorphic encryption. *Tatra mountains mathematical publications*, 67(1), 2016. https://doi.org/10.1515/tmmp-2016-0031.

[DR98] Joan Daemen and Vincent Rijmen. The Block Cipher Rijndael. In *Proceedings of the 3rd International Conference on Smart Card Research and Applications (CARDIS)*, pages 277–284, 1998. https://doi.org/10.1007/10721064_26.

[DSC+15] Tien Tuan Anh Dinh, Prateek Saxena, Ee-Chien Chang, Beng Chin Ooi, and Chunwang Zhang. M2R: Enabling Stronger Privacy in MapReduce Computation. In *Proceedings of the 24th USENIX Security Symposium*, pages 447–462, 2015. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/dinh.

[Dwo15] Morris J. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. https://doi.org/10.6028/NIST.FIPS.202, aug 2015.

[EAEG06] Fatih Emekçi, Divyakant Agrawal, Amr El Abbadi, and Aziz Gulbeden. Privacy Preserving Query Processing Using Third Parties. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, page 27, 2006. https://doi.org/10.1109/ICDE.2006.116.

[EMST78] William Friedrich Ehrsam, Carl H. W. Meyer, John Lynn Smith, and Walter Leonard Tuchman. Message Verification and Transmission Error Detection by Block Chaining. US Patent 4074066, February 1978.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 1–19, 2004. https://doi.org/10.1007/978-3-540-24676-3_1.

[Fou19a] Apache Software Foundation. Apache Flink (release 1.8.0). https://flink.apache.org/, April 2019.

[Fou19b] Apache Software Foundation. Apache Hadoop (release 3.2.0). https://hadoop.apache.org/, January 2019.

[FV12a] Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

[FV12b] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

[GABL17] Matthieu Giraud, Alexandre Anzala-Yamajako, Olivier Bernard, and Pascal Lafourcade. Practical Passive Leakage-abuse Attacks Against Symmetric Searchable Encryption. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE)*, pages 200–211, 2017. https://doi.org/10.5220/0006461202000211.

[Gal14] François Le Gall. Powers of Tensors and Fast Matrix Multiplication. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014. https://doi.org/10.1145/2608628.2608664.

[Gam85] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. https://doi.org/10.1109/TIT.1985.1057074.

[Gen09] Craig Gentry. Fully Homomorphic Encryption using Ideal Lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 169–178, 2009. https://doi.org/10.1145/1536414.1536440.

[GMN+16] Paul Grubbs, Richard McPherson, Muhammad Naveed, Thomas Ristenpart, and Vitaly Shmatikov. Breaking Web Applications Built On Top of Encrypted Data. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1353–1364, 2016. https://doi.org/10.1145/2976749.2978351.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987. https://doi.org/10.1145/28395.28420.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2001. ISBN: 9780521830843.

[GPT10] Robert Griesemer, Rob Pike, and Ken Thompson. Go (release 1.12.6). https://golang.org/, June 2010. The Go Authors.

[Gri15] Joseph Grifone. *Algèbre linéaire*. Cépaduès-Editions, 2015. ISBN: 9782364931831.

[GSB+17] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-Abuse Attacks against Order-Revealing Encryption. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 655–672, 2017. https://doi.org/10.1109/SP.2017.44.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Proceedings of the 33rd Annual Cryptology Conference (CRYPTO)*, pages 75–92, 2013. https://doi.org/10.1007/978-3-642-40041-4_5.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *Proceedings of the 19th Annual Network and Distributed System Security Symposium (NDSS)*, 2012. https://www.ndss-symposium.org/ndss2012/private-set-intersection-are-garbled-circuits-better-custom-protocols.

[HJJ+96] Steven Huss-Lederman, Elaine M. Jacobson, J. R. Johnson, Anna Tsao, and Thomas Turnbull. Implementation of Strassen's Algorithm for Matrix Multiplication. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, page 32, 1996. http://doi.ieeecomputersociety.org/10.1109/SC.1996.18.

[HK16] F. Maxwell Harper and Joseph A. Konstan. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19:1–19:19, January 2016. https://doi.org/10.1145/2827872.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. In *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography (PKC)*, pages 312–331, 2010. https://doi.org/10.1007/978-3-642-13013-7_19.

[HS19] Shai Halevi and Victor Shoup. HElib – An Implementation of Homomorphic Encryption. https://github.com/homenc/HElib, June 2019.

[HW08] Godfrey Harold Hardy and Sir Edward Maitland Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 2008. ISBN: 9780199219865.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast Secure Computation of Set Intersection. In *Proceedings of the 7th International Conference on Security and Cryptography for Networks (SCN)*, pages 418–435, 2010. https://doi.org/10.1007/978-3-642-15317-4_26.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, RFC Editor, February 1997. https://tools.ietf.org/rfc/rfc2104.txt.

[Ker15] Florian Kerschbaum. Frequency-Hiding Order-Preserving Encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 656–667, 2015. https://doi.org/10.1145/2810103.2813629.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series. Chapman and Hall/CRC, 2014. ISBN: 9781466570269.

[KM15] Neal Koblitz and Alfred J. Menezes. The Random Oracle Model: A Twenty-Year Retrospective. *Designs, Codes and Cryptography*, 77(2-3):587–610, 2015. https://doi.org/10.1007/s10623-015-0094-2.

[Koz84] Wladyslaw Kozaczuk. *Enigma: How The German Machine Cipher Was Broken, And How It Was Read By The Allies In World War Two*. Foreign Intelligence Book. Praeger, 1984. ISBN: 9780313270079.

[KS98] Burt Kaliski and Jessica Staddon. PKCS 1: RSA Cryptography Specifications Version 2.0. RFC 2437, RFC Editor, October 1998. https://tools.ietf.org/rfc/rfc2437.txt.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-Preserving Set Operations. In *Proceedings of the 25th Annual International Cryptology Conference (CRYPTO)*, pages 241–257, 2005. https://doi.org/10.1007/11535218_15.

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6):84–90, May 2017. https://doi.acm.org/10.1145/3065386.

[Lin17] Yehuda Lindell. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. 2017. https://doi.org/10.1007/978-3-319-57048-8.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014. ISBN: 9781107077232.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, pages 1219–1234, 2012. https://doi.org/10.1145/2213977.2214086.

[LTW13] Sven Laur, Riivo Talviste, and Jan Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *Proceedings of the 11th International Conference on Applied Cryptography and Network Security (ACNS)*, pages 84–101, 2013. https://doi.org/10.1007/978-3-642-38980-1_6.

[Mac16] Hugo Daniel Macedo. Gaussian Elimination is Not Optimal. *Journal of Logical and Algebraic Methods in Programming*, 85(5):999–1010, August 2016. https://doi.org/10.1016/j.jlamp.2016.06.003.

[MBC13] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. PIRMAP: Efficient Private Information Retrieval for MapReduce. In *Proceedings of the 17th International Conference on Financial Cryptography and Data Security (FC)*, pages 371–385, 2013. https://doi.org/10.1007/978-3-642-39884-1_32.

[MD08] Qingkai Ma and Ping Deng. Secure Multi-party Protocols for Privacy Preserving Data Mining. In *Proceedings of the 3rd International Conference on Wireless Algorithms, Systems, and Applications (WASA)*, pages 526–537, 2008. https://doi.org/10.1007/978-3-540-88582-5_49.

[MR19] WA. Microsoft Research, Redmond. Microsoft SEAL (release 3.2). https://github.com/Microsoft/SEAL, February 2019.

[MRDY18] Pradeep Kumar Mishra, Deevashwer Rathee, Dung Hoang Duong, and Masaya Yasuda. Fast Secure Matrix Multiplications over Ring-Based Homomorphic Encryption. Cryptology ePrint Archive, Report 2018/663, 2018. https://eprint.iacr.org/2018/663.

[MRS17] Matteo Maffei, Manuel Reinert, and Dominique Schröder. On the Security of Frequency-Hiding Order-Preserving Encryption. In *Proceedings of the 16th International Conference on Cryptology and Network Security (CANS)*, pages 51–70, 2017. https://doi.org/10.1007/978-3-030-02641-7_3.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press, 1996. ISBN: 9780849385230.

[NKW15] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference Attacks on Property-Preserving Encrypted Databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 644–655, 2015. https://doi.org/10.1145/2810103.2813651.

[NLV11] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can Homomorphic Encryption be Practical? In *Proceedings of the 3rd ACM Cloud Computing Security Workshop (CCSW)*, pages 113–124, 2011. https://doi.org/10.1145/2046660.2046682.

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 223–238, 1999. https://doi.org/10.1007/3-540-48910-X_16.

[Pro19] GNU Project. The GNU Privacy Guard (release 2.2.16). https://gnupg.org/, May 2019.

[PRZB11] Raluca A. Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 85–100, 2011. https://doi.org/10.1145/2043556.2043566.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. https://doi.org/10.1145/359340.359342.

[RSN12] Sushmita Ruj, Milos Stojmenovic, and Amiya Nayak. Privacy Preserving Access Control with Authentication for Securing Data in Clouds. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 556–563, 2012. https://doi.org/10.1109/CCGrid.2012.92.

[SB19] Ramalingam Saravanan and Patrick Brunschwig. Enigmail (release 2.0.11). https://www.enigmail.net/, May 2019.

[SCF+15] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. VC3: Trustworthy Data Analytics in the Cloud Using SGX. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, pages 38–54, 2015. https://doi.org/10.1109/SP.2015.10.

[Sha49] Claude Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4):656–715, October 1949. https://doi.org/10.1002/j.1538-7305.1949.tb00928.x.

[Sha79] Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979. https://doi.org/10.1145/359168.359176.

[SSW77] Robert H. Morris Sr., N. J. A. Sloane, and Aaron D. Wyner. Assessment of the national bureau of standards proposed federal data encryption standard. *Cryptologia*, 1(3):281–291, 1977. https://doi.org/10.1080/0161-117791833020.

[SZ99] Avi Shoshan and Uri Zwick. All Pairs Shortest Paths in Undirected Graphs with Integer Weights. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 605–615, 1999. https://doi.org/10.1109/SFFCS.1999.814635.

[TNP16] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. Private and Scalable Execution of SQL Aggregates on a Secure Decentralized Architecture. *ACM Transactions on Database Systems (TODS)*, 41(3):16:1–16:43, 2016. https://doi.org/10.1145/2894750.

[TTP19] Inc. The Tor Project. Tor Project (release 0.4.0.5). https://www.torproject.org/, May 2019.

[VBN15] Triet D. Vo-Huu, Erik-Oliver Blass, and Guevara Noubir. EPiC: Efficient Privacy-Preserving Counting for MapReduce. In *Proceedings of the 3rd International Conference on Networked Systems (NETYS)*, pages 426–443, 2015. https://doi.org/10.1007/978-3-319-26850-7_29.

[WSZ+09] I-Cheng Wang, Chih-Hao Shen, Justin Zhan, Tsan-sheng Hsu, Churn-Jung Liau, and Da-Wei Wang. Toward Empirical Aspects of Secure Scalar Product. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(4):440–447, April 2009. https://doi.org/10.1109/TSMCC.2009.2016430.

[Yao82] Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *Proceedings of the 23rd IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982. https://doi.org/10.1109/SFCS.1982.38.

[YSK+15a] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. New Packing Method in Somewhat Homomorphic Encryption and its Applications. *Security and Communication Networks*, 8(13):2194–2213, September 2015. https://doi.org/10.1002/sec.1164.

[YSK+15b] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure Statistical Analysis Using RLWE-Based Homomorphic Encryption. In *Proceedings of the 20th Australasian Conference on Information Security and Privacy (ACISP)*, pages 471–487, 2015. https://doi.org/10.1007/978-3-319-19962-7_27.

[Zwi98] Uri Zwick. All Pairs Shortest Paths in Weighted Directed Graphs Exact and Almost Exact Algorithms. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 310–319, 1998. https://doi.org/10.1109/SFCS.1998.743464.