Secure Intersection with MapReduce

Radu Ciucanu[1], Matthieu Giraud[2], Pascal Lafourcarde[2] and Lihua Ye[3]
[1]*INSA Centre Val de Loire, Univ. Orléans, LIFO EA 4022, Bourges, France*
[2]*LIMOS, UMR 6158, Université Clermont Auvergne, Aubière, France*
[3]*Harbin Institute of Technology, China*

Keywords: Intersection, Database, Privacy, Security, MapReduce.

Abstract: Relation intersection is a fundamental problem, which becomes non-trivial when the relations to be intersected are too large to fit on a single machine. Hence, a natural approach is to design parallel algorithms that are executed on a cluster of machines rented from a public cloud provider. Intersection of relations becomes even more difficult when each relation belongs to a different data owner that wants to protect her data privacy. We consider the popular MapReduce paradigm for outsourcing data and computations to a *semi-honest* public cloud. Our main contribution is the SI protocol (for *Secure Intersection*) that allows to securely compute the intersection of an arbitrary number of relations, each of them being encrypted by its owner. The user allowed to query the intersection result has only to decrypt the result sent by the public cloud. SI does not leak (to the public cloud or to the user) any information on tuples that are not in the final relation intersection result, even if the public cloud and the user collude i.e., they share all their private information. We prove the security of SI and provide an empirical evaluation showing its efficiency.

1 INTRODUCTION

The outsourcing of data and computation to the cloud is a frequent scenario in modern applications. While many cloud service providers with an important amount of data storage and of power computation (e.g., Google Cloud Platform, Amazon Web Services, Microsoft Azure) are available for a reasonable price, they do not usually address the fundamental problem of protecting the privacy of users' data.

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 (Dean and Ghemawat, 2004) paradigm for outsourcing data and computations to a semi-honest public cloud. Our goal is to compute the relations' intersection while preserving the data privacy of each of the data owners. We develop a protocol based on public-key cryptography where each participant encrypts and sends their respective relation on the public cloud. The public cloud cannot learn neither the input nor the output data, but may learn only the number of tuples in the intersection. At the end of the computation, the public cloud sends the result to the final user, who only has to decrypt the received data. Moreover, if the public cloud and the user collude, i.e., the public cloud knows the user's private key, then they cannot learn other information than the intersection result. Secure intersection has a lot of applications such that privacy-preserving data mining (Aggarwal and Yu, 2008), homeland security (Cristofaro and Tsudik, 2010), human genome research (Baldi et al., 2011), Botnet detection (Nagaraja et al., 2010), social networks (Mezzour et al., 2009), and location sharing (Narayanan et al., 2011).

1.1 Intersection with MapReduce

A protocol to compute the intersection between two relations with MapReduce is presented in Chapter 2 of (Leskovec et al., 2014). We note 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 (Leskovec et al., 2014), 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 that have the same key are sent to the same reducer (i.e.,

Ciucanu, R., Giraud, M., Lafourcarde, P. and Ye, L.
Secure Intersection with MapReduce.
DOI: 10.5220/0007918902360243
In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019)*, pages 236-243
ISBN: 978-989-758-378-0

the application of the Reduce function to a single key and its associated values). 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 key-value result form required by the MapReduce paradigm. Hence, all tuples received by the user correspond to the tuples that are in both relations. However, the key is irrelevant at the end of the protocol, hence we often 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 ID". They are defined as follows: $\mathsf{NSA} = \{\texttt{F654},\texttt{U840},\texttt{X098}\}$, $\mathsf{GCHQ} = \{\texttt{F654},\texttt{M349},\texttt{P027}\}$, and $\mathsf{Mossad} = \{\texttt{F654},\texttt{M349},\texttt{U840}\}$. 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 1. 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.

1.2 Problem Statement

We assume $n+2$ parties: n data owners, the public cloud, and the external user (simply referred as user in the following). Each data owner is trusted (i.e., they dutifully follow the protocol and do not collude with other party) and outsources a relation R_i, with $i \in [\![1,n]\!]$, to the public cloud, denoted $\mathcal{C}$. We denote by $\mathcal{R}_i$ the owner of the relation R_i for $i \in [\![1,n]\!]$. A user, denoted $\mathcal{U}$, and who does not know the individual relations R_i is authorized to query the intersection of these n relations.

We assume that the public cloud is *semi-honest* (Lindell, 2017), i.e., it executes dutifully the computation task but tries to learn the maximum of information on relations R_i and on their intersection. In the original protocol (Leskovec et al., 2014), 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 1. To preserve data owners' privacy, the cloud should not learn any plain input data, contrary to what happens for the original protocol.

Moreover, we assume that the public cloud can collude with the user, i.e., they share all their respective private information. We want that the user that queried the intersection of these n relations may learn nothing else than the intersection of the n relations, even in case of collusion with the public cloud.

1.3 Contributions

We revisit the standard protocol for the computation of intersection with MapReduce (Leskovec et al., 2014) and propose a new protocol called SI (for Secure Intersection) 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. 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 of the intersection. SI also satisfies the problem setting in the presence of collusion between the user and the public cloud. The security proof of our protocol is given in the extended version available online[1].
- To show the practical scalability of SI, we present experimental results using the MapReduce open-source implementation Apache Hadoop 3.2.0.
- Our 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 (Leskovec et al., 2014). Our technique is based on classical cryptographic tools such that pseudo-random function, asymmetric

[1] https://hal.archives-ouvertes.fr/hal-02129141

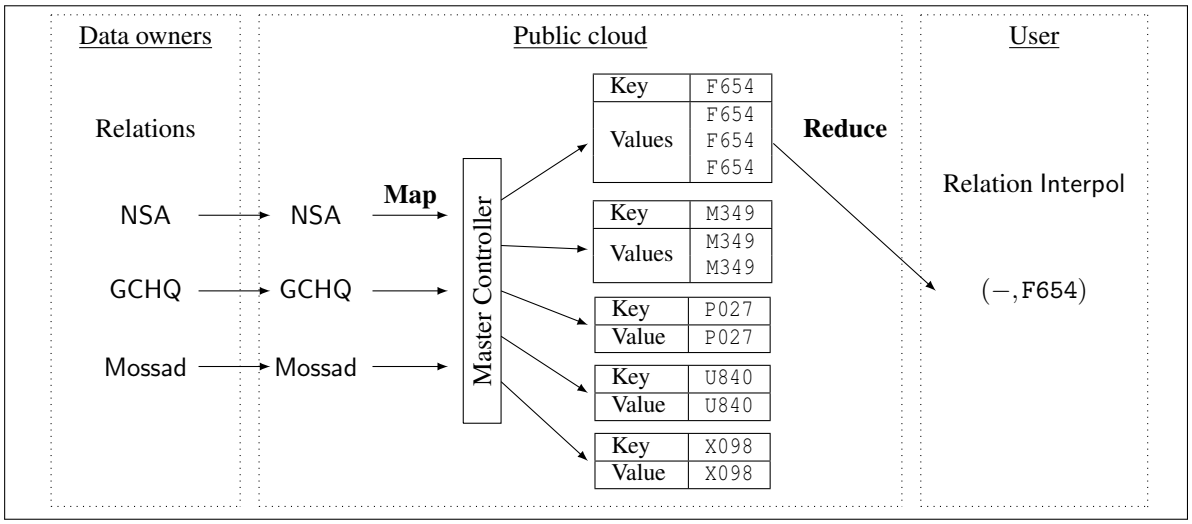

Figure 1: Example of intersection with MapReduce between three relations.

Protocol	*Comp. cost*	*Com. cost*
Standard	$2nN$	$(n+1)N$
SI	$C_{\mathcal{E}} \cdot N$ $+C_f \cdot (3n-2)$ $+C_{\oplus} \cdot (2N \cdot (n-2))$	$(n+1)N$

Figure 2: Trade-offs between computation and communication costs for our secure protocol SI vs the standard MapReduce protocol. Let $N = \max(|R_1|, \ldots, |R_n|)$. Let C_f (resp. $C_{\mathcal{E}}$, $C_{\oplus}$) be the computation cost of a pseudo-random function evaluation (resp. asymmetric encryption, xor operation).

and one-time-pad encryptions. We summarize in Figure 2 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. In our communication cost analysis, we measure the total size of the data that is emitted from a map or reduce node.

1.4 Related Work

As previously mentioned, a relation can be seen as a set where tuples are set's elements. *Private Set Intersection* (PSI) refers to the cryptographic primitive where two parties compute the intersection of their respective sets such that minimal information is revealed during the process. It was introduced by Freedman et al. (Freedman et al., 2004). The aim of such a 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* (Cristofaro et al., 2010). 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* (Cristofaro et al., 2010). Contrary to these approaches, our protocol SI does not reveal any information on the intersection to the data owner. Only the user knows the intersection.

The seminal work (Freedman et al., 2004) uses two-party computation and partial homomorphic encryption allowing two owners to securely compute the intersection of two sets. The proposed protocol is proven against *semi-honest* adversaries in the standard model and also proven for a malicious adversary in random oracle model. Authors consider one client and one server, each of them owning a secret dataset where the client sends polynomial coefficients associated to her dataset in an encrypted way to the server. At the end of the protocol the server knows which elements are shared with the server while the later learns nothing. In our protocol SI, we consider an arbitrary number of clients owning different relations and using a *semi-honest* public cloud to send the intersection of the relations to the user.

Following this work, (Hazay and Nissim, 2010) proposed an improved construction considering the presence of a malicious adversaries in the standard model. Contrary to us, the complexity of this construction still remains not linear in the number of elements in sets. (Cristofaro et al., 2010), and (Kissner and Song, 2005) proposed protocols for mutual PSI with linear complexity while in our protocol the user does not have any set to intersect. The scheme proposed by (Cristofaro et al., 2010) considers a malicious adversary using zero-knowledge proofs. Their scheme requires that the user performs computations at the beginning and at the end of the protocol while

she has only to decrypt the final result in our protocol. In the scheme of (Kissner and Song, 2005), each data owner learns the result of the intersection. In this paper, we consider semi-honest adversary and prove the security of our protocol in the random oracle model. As remarked above, the intersection computed by our protocol is only known by the user and not by the data owners and the public cloud.

More recently, Hazay and Venkitasubramaniam (Hazay and Venkitasubramaniam, 2017) proposed a protocol considering a star topology between data owners where a designated party first learns the encrypted cross intersection with other parties, then deduces the outcome. In our protocol, we consider a different topology where a public cloud receives encrypted sets to intersect and sends the outcome to the user. In (Kolesnikov et al., 2017), the authors proposed a protocol based on oblivious programmable PRF (OPPRF) where each party has to compute a share of zero for each set's element in a coordinate way. The outcome is obtained by a centralized data owner considering only OPPRF evaluations that are equal to zero. In our protocol, the user that queries the intersection is not a data owner, moreover, data owners just have to share PRF secret keys and no to perform secret sharing computation.

Outline. We introduce the needed cryptographic tools in Section 2. We recall the standard MapReduce set intersection protocol and present our secure protocol SI in Section 3. Before to conclude, we present in Section 4 experimental evaluations of SI protocol.

2 CRYPTOGRAPHIC TOOLS

We recall definitions of public-key encryption scheme and of pseudo-random function used in SI.

Definition 1 (Public-key encryption)**.** *Let η be a security parameter. A* public-key encryption *(PKE) scheme consists of three algorithms.*

- *The randomized* key generation algorithm $\mathcal{G}$ *takes the security parameter to return a public/secret key pair (pk, sk).*
- *The* encryption algorithm $\mathcal{E}$ *takes a public key pk and a plaintext m to return a ciphertext c. We denote by $\mathcal{E}(pk, m)$ the encryption of m.*
- *The deterministic* decryption algorithm $\mathcal{D}$ *takes a secret key sk and a ciphertext c to return a corresponding plaintext m or a special symbol $\perp$ indicating that the ciphertext was invalid. We denote by $\mathcal{D}(sk, c)$ the decryption of c.*

Definition 2 (Pseudo-random function)**.** *Let η be a security parameter. A* pseudo-random function *(PRF) F is a deterministic algorithm that has two inputs: $k \in \{0,1\}^{\ell(\eta)}$ (where $\ell(\cdot)$ is a polynomial function), and $x \in \mathcal{X}$. Its output is $y := F(k,x) \in \mathcal{Y}$. We said that F is defined over $(\{0,1\}^{\ell(\eta)}, \mathcal{X}, \mathcal{Y})$.*

In the rest of the paper, we assume that data of participants are included in $\mathcal{X}$, and the size of $\mathcal{Y}$ is larger enough to avoid collisions. Moreover, we denote by $f_k(\cdot) = F(k,\cdot)$ an instance of F.

3 MAPREDUCE INTERSECTION

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 3.1 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 (Leskovec et al., 2014). This protocol obviously does not verify 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 3.2 our secure protocol denoted SI that computes the intersection of n relations using MapReduce. We prove that contrary to the standard 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 still knows, and cardinals of relations R_i for $i \in [\![1,n]\!]$ that the public cloud still knows, and nothing else.

3.1 Standard MapReduce Intersection

In the standard protocol (Leskovec et al., 2014), 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 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 standard protocol computing the intersection protocol with MapReduce in Figure 3.

Map function:
// *key*: id of a chunk of R_i
// *value*: collection of tuples $t \in R_i$
foreach $t \in R_i$ **do**
  emit (t,t).

Reduce function:
// *key*: tuple $t \in \cup_{i=1}^{n} R_i$
// *values*: collection of tuples t
$L = [\,]$
foreach $v \in$ values **do**
  $L \leftarrow L \cup \{v\}$
if $|L| = n$ **then**
  emit $(-,t)$.

Figure 3: 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.

3.2 Secure MapReduce Intersection

In order to compute intersection with MapReduce in a privacy-preserving way between $n \geq 2$ relations, our protocol uses pseudo-random function, asymmetric and one-time encryptions. We denote by F a secure pseudo-random function defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ and by $\Pi = (\mathcal{G}, \mathcal{E}, \mathcal{D})$ an IND-CPA asymmetric encryption scheme. We also assume that the length of values outputted by the pseudo-random function is equal to the length of ciphertext outputted by the asymmetric encryption scheme. In practice, we can use the *Advanced Encryption Standard* (AES) (Daemen and Rijmen, 2002) with the *Cipher Block Chaining* mode on padded message in order to obtain a ciphertext of the same length than the ciphertext obtained with the asymmetric encryption.

Preprocessing of our Secure Protocol SI. 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 a *protected* relation denoted R_i^*. We present the key setup and the preprocessing phase in Figure 4.

First, we need a secret key $k_1 \in \mathcal{X}$ that is shared between the n data owners. Moreover we need $n-1$ other secret keys $k_i \in \mathcal{K}$ (for $2 \leq i \leq n$) 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 secret keys equals 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 note 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.

Preprocessing:
// *input*: relation R_i with $i \in [\![1,n]\!]$
// *outputs*: protected relation R_i^*
for $1 \leq i \leq n$ **do** $k_i \xleftarrow{\$} \mathcal{K}$;
$R_i^* \leftarrow \emptyset$;
if $i = 1$ **then**
  foreach $t \in R_1$ **do**
    $R_1^* \leftarrow R_1^* \cup \{(f_{k_1}(t), (\mathcal{E}(pk,t) \oplus_{j=2}^{n} f_{k_j}(t)))\}$
else
  foreach $t \in R_i$ **do**
    $R_i^* \leftarrow R_i^* \cup \{(f_{k_1}(t), (f_{k_i}(t)))\}$
return R_i^*.

Figure 4: Preprocessing of our secure protocol SI run by each data owner.

The aim of this preprocessing is to protect owners' data in order to avoid the public cloud to learn tuples of each relation and the result of the intersection sent to the user. Moreover, this preprocessing is in agreement with the MapReduce paradigm. Indeed, each protected 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 protected 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 $(f_{k_1}(t), \mathcal{E}(pk,t) \oplus_{j=2}^{n} f_{k_j}(t))$. 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.

Map and Reduce Phases of SI Protocol. The preprocessing presented in Figure 4 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.

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.

Map function:
// *key*: id of a chunk of R_i^* with $i \in [\![1,n]\!]$
// *values*: collection of $(f_{k_1}(t), \mathcal{E}(pk,t) \oplus_{j=2}^{n} f_{k_j}(t))$
// or $(f_{k_1}(t), f_{k_j}(t))$ with $j \in [\![2,n]\!]$
foreach $(k,v) \in$ *values* **do**
  emit (k,v)

Reduce function:
// *key*: $f_{k_1}(t)$ such that $t \in \cup_{i=1}^{n} R_i$
// *values*: collection of $\mathcal{E}(pk,t) \oplus_{j=2}^{n} f_{k_j}(t)$ or $f_{k_j}(t)$ // with $j \in [\![2,n]\!]$
$L \leftarrow [\,]$
foreach $v \in$ *values* **do**
  $L \leftarrow L \cup \{v\}$
if $|L| = n$ **then**
  $\mathcal{E}(pk,t) \leftarrow \mathcal{E}(pk,t) \oplus_{j=2}^{n} f_{k_j}(t) \oplus_{j=2}^{n} f_{k_j}(t)$
  emit $(-,\mathcal{E}(pk,t))$

Figure 5: Map and Reduce functions of SI.

Example. We illustrate our SI protocol following the example presented in Section 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 protected relations denoted NSA*, GCHQ*, and Mossad*: NSA* has tuples $(f_{k_1}(\texttt{F65}), \mathcal{E}(pk,\texttt{F65}) \oplus f_{k_2}(\texttt{F65}) \oplus f_{k_3}(\texttt{F65}))$, $(f_{k_1}(\texttt{U84}), \mathcal{E}(pk,\texttt{U84}) \oplus f_{k_2}(\texttt{U84}) \oplus f_{k_3}(\texttt{U84}))$, and $(f_{k_1}(\texttt{X09}), \mathcal{E}(pk,\texttt{X09}) \oplus f_{k_2}(\texttt{X09}) \oplus f_{k_3}(\texttt{X09}))$; GCHQ* has tuples $(f_{k_1}(\texttt{F65}), f_{k_2}(\texttt{F65}))$, $(f_{k_1}(\texttt{M34}), f_{k_2}(\texttt{M34}))$, $(f_{k_1}(\texttt{P02}), f_{k_2}(\texttt{P02}))$; Mossad* has tuples $(f_{k_1}(\texttt{F65}), f_{k_3}(\texttt{F65}))$, $(f_{k_1}(\texttt{M34}), f_{k_3}(\texttt{M34}))$, $(f_{k_1}(\texttt{U84}), f_{k_3}(\texttt{U84}))$. We illustrate the execution of intersection computation with MapReduce using our secure protocol SI in Figure 6.

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 1. *Assume that the pseudo-random function family F perfectly emulates a random oracle, then protocol* SI *is correct.*

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 $t \in R$, there exists a key-value pair in relation R_1^* of the form $(f_{k_1}(t), \mathcal{E}(pk,t) \oplus_{i=2}^{n} f_{k_i}(t))$, 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:

$$\mathcal{E}(pk,t) \oplus_{i=2}^{n} f_{k_i}(t) \oplus_{i=2}^{n} f_{k_i}(t) = \mathcal{E}(pk,t)\,.$$

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. □

4 EXPERIMENTAL RESULTS

We present an experimental comparison between the standard MapReduce set intersection protocol (Leskovec et al., 2014), and our secure protocol SI. We run two types of experiments, where we vary two different parameters: the *number of tuples per relation* for a fixed number of 2 relations, and the *number of intersected relations*.

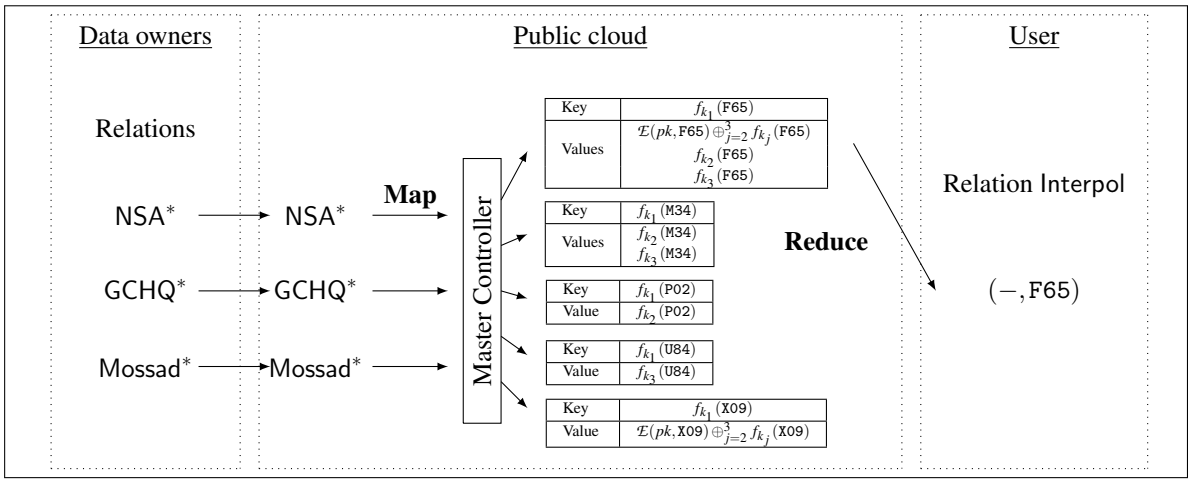

Figure 6: Example of intersection with MapReduce between three relations using our secure protocol SI.

Settings. We have done all computations on a cluster running Ubuntu Server 16.04 LTS with Hadoop 3.2.0 using Java 1.8.0. We use the Hadoop streaming utility and implement the Map and Reduce functions in Golang 1.6.2. The cluster is composed of one master node and of ten slave nodes. Each node has four CPU cadenced to 2.4 GHz, 80 GB of disk, and 8 GB of RAM.

According to our SI protocol, we use as pseudo-random function the *Advanced Encryption Standard* (AES) symmetric encryption scheme in *Cipher Block Chaining* (CBC) encryption mode with a key size of 128 bits. For the purpose of our protocol, the initial vector is fixed and common to all data owners. For the asymmetric encryption scheme, we use the RSA-OAEP scheme with a key size of 2048 bits and SHA-256 as hash function.

For each experiment, we report average CPU times over 8 runs. Since the cluster environment is not isolated from other machines of the netword, we do not give measures for the communication cost.

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. 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 (Leskovec et al., 2014) and our secure protocol SI on couples of relations of cardinal $500,000$ to $3,000,000$, by step of $250,000$.

We remark in Figure 7 that the computation complexity of our secure protocol is linear as determined in the complexity study (cf. Figure 2).

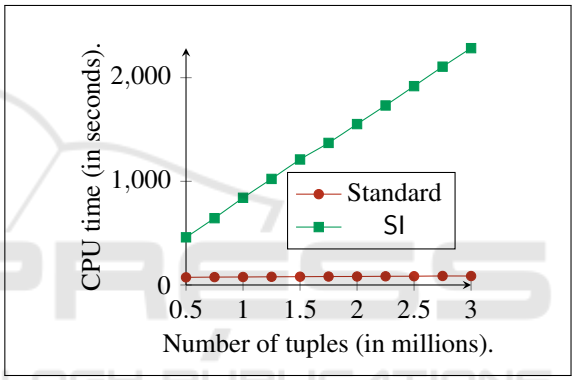

Figure 7: CPU time vs the number of tuples for the Standard and SI protocols to compute the intersection between two relations.

Varying the Number of Intersected Relations. In the experiment on the number of intersected relations, we consider intersection between different number of relations of the same schema composed of only one attribute. We start by computing the intersection of 2 relations to finish with the intersection of 10 relations. In each case, relations have $500,000$ tuples and shares $250,000$ tuples. In practice, for the intersection of $2 \leq n \leq 10$ relations, tuples of the first relation consist in integers from 1 to $500,000$, and tuples of the i-th relation, with $2 \leq i \leq n$, consist in integers from 1 to $250,000$ and integers from $i \cdot 250,000+1$ to $(i+1) \cdot 250,000$.

We compare the standard protocol (Leskovec et al., 2014) and our secure protocol SI for the experiment on the number of intersected relations. As shown in Figure 8, the computation complexity of our secure protocol is linear as determined in the complexity study (cf. Figure 2). We observe that the computation complexity is less compare to the experiment

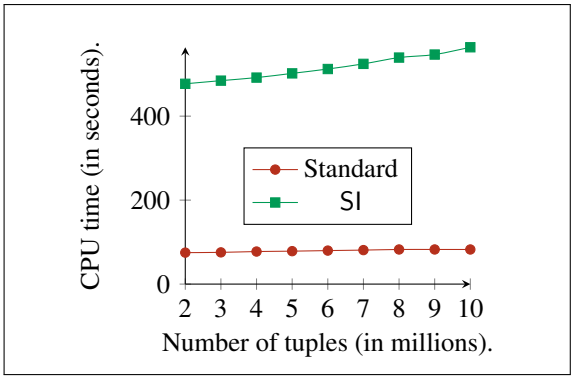

Figure 8: CPU time of the experiment on the number of intersected relation on the original protocol(Leskovec et al., 2014) and of our secure protocol SI.

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 equals 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 equals 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 performs a large number of exclusive or on 2048-bits strings.

5 CONCLUSION

We have presented an efficient privacy-preserving protocol using the MapReduce paradigm to compute the intersection between an arbitrary number of relations. In fact, in the standard protocol (Leskovec et al., 2014), the public cloud performing the computation learns all tuples of the data owners along the intersection result that it sends to the user. In our protocol SI, the public cloud cannot learn such information on the input sets. Moreover, if the cloud and the user collude, then they cannot learn more than the 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, and the cardinal of their intersection. 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 with the Hadoop framework and presented empirical results showing the scalability of SI.

Looking forward to future work, we plan to study secure set intersection with MapReduce while considering a malicious public cloud, i.e., the public cloud can perform any operations on data that it process.

REFERENCES

Aggarwal, C. C. and Yu, P. S. (2008). A general survey of privacy-preserving data mining models and algorithms. In *Privacy-Preserving Data Mining*.

Baldi, P., Baronio, R., Cristofaro, E. D., Gasti, P., and Tsudik, G. (2011). Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *CCS*.

Cristofaro, E. D., Kim, J., and Tsudik, G. (2010). Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *ASIACRYPT*.

Cristofaro, E. D. and Tsudik, G. (2010). Practical private set intersection protocols with linear complexity. In *FC*.

Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael: AES*. Information Security and Cryptography. Springer.

Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *OSDI*.

Freedman, M. J., Nissim, K., and Pinkas, B. (2004). Efficient Private Matching and Set Intersection. In *EUROCRYPT*.

Hazay, C. and Nissim, K. (2010). Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC*.

Hazay, C. and Venkitasubramaniam, M. (2017). Scalable multi-party private set-intersection. In *PKC*.

Kissner, L. and Song, D. X. (2005). Privacy-preserving set operations. In *CRYPTO*.

Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., and Trieu, N. (2017). Practical multi-party private set intersection from symmetric-key techniques. In *CCS*.

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

Lindell, Y. (2017). How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*.

Mezzour, G., Perrig, A., Gligor, V. D., and Papadimitratos, P. (2009). Privacy-preserving relationship path discovery in social networks. In *CANS*.

Nagaraja, S., Mittal, P., Hong, C., Caesar, M., and Borisov, N. (2010). Botgrep: Finding P2P bots with structured graph analysis. In *USENIX*.

Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., and Boneh, D. (2011). Location privacy via private proximity testing. In *NDSS*.