Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cosrev

Survey

Security and privacy aspects in MapReduce on clouds: A survey

Philip Derbeko[a], *Shlomi Dolev*[b], *Ehud Gudes*[b], *Shantanu Sharma*[b,*]

[a] *EMC, Beer-Sheva, Israel*
[b] *Department of Computer Science, Ben-Gurion University of the Negev, Israel*

ARTICLE INFO

Article history:
Received 1 July 2015
Received in revised form
1 May 2016
Accepted 2 May 2016
Published online 25 May 2016

Keywords:
Cloud computing
Distributed computing
Hadoop
HDFS
Hybrid cloud
Private cloud
Public cloud
MapReduce algorithms
Privacy
Security

ABSTRACT

MapReduce is a programming system for distributed processing of large-scale data in an efficient and fault tolerant manner on a private, public, or hybrid cloud. MapReduce is extensively used daily around the world as an efficient distributed computation tool for a large class of problems, *e.g.*, search, clustering, log analysis, different types of join operations, matrix multiplication, pattern matching, and analysis of social networks. Security and privacy of data and MapReduce computations are essential concerns when a MapReduce computation is executed in public or hybrid clouds. In order to execute a MapReduce job in public and hybrid clouds, authentication of mappers–reducers, confidentiality of data-computations, integrity of data-computations, and correctness–freshness of the outputs are required. Satisfying these requirements shields the operation from several types of attacks on data and MapReduce computations. In this paper, we investigate and discuss security and privacy challenges and requirements, considering a variety of adversarial capabilities, and characteristics in the scope of MapReduce. We also provide a review of existing security and privacy protocols for MapReduce and discuss their overhead issues.

Contents

* *Corresponding author.*
E-mail addresses: philip.derbeko@emc.com (P. Derbeko), dolev@cs.bgu.ac.il (S. Dolev), ehud@cs.bgu.ac.il (E. Gudes), sharmas@cs.bgu.ac.il (S. Sharma).

http://dx.doi.org/10.1016/j.cosrev.2016.05.001
1574-0137/

1. Introduction

Cloud computing [1] infrastructure provides on-demand, easy, and scalable access to a shared pool of configurable resources, without worrying about managing those resources. Details about cloud computing can be found in [2,3]. Clouds provide three types of services, as follows: (i) *infrastructure-as-a-service*, IaaS, provides infrastructure in terms of virtual machines, storage, and networks, (ii) *platform-as-a-service*, PaaS, provides a scalable software platform allowing the development of custom applications, and (iii) *software-as-a-service*, SaaS, provides software running in clouds as a service, for example, emails and databases. Clouds can be classified into three types, as follows: (i) *public cloud*: a cloud that provides services to many users and is not under the control of a single exclusive user, (ii) *private cloud*: a cloud that has its proprietary resources and is under the control of a single exclusive user, and (iii) *hybrid cloud*: a combination of public and private clouds.

One of the most common *platform-as-a-service* computational paradigms is MapReduce [4], introduced by Google in 2004. MapReduce provides an efficient and fault tolerant parallel processing of large-scale data without any costly and dedicated computing node like a supercomputer. At the beginning, MapReduce was designed to be deployed on-premises under mistaken assumption that local environment can be completely trusted. Thus, security and privacy aspects were overlooked in the initial design. As MapReduce gained popularity the lack of security and privacy in on-premises deployment become severe shortcoming. In addition, MapReduce is being deployed on both hybrid and public clouds, which are prone to many attacks and security threats. In the current days, several public clouds, *e.g.*, Amazon Elastic MapReduce, Google App Engine, IBM's Blue Cloud, and Microsoft Azure, enable users to perform MapReduce cloud computations without considering physical infrastructures and software installations. Thus, the deployment of MapReduce in public clouds enables users to process large-scale data in a cost-effective manner and establishes a relationship between two independent entities, *i.e.*, clouds and MapReduce. As a downside, the deployment of MapReduce in hybrid and public cloud needs to deal with many attacks on the communication networks and (the three service layers of) the cloud.

Data processing in the cloud highlights a tradeoff between the ease of processing and security–privacy of data and computations. Specifically, on one hand, the deployment of MapReduce in a well-managed public cloud provides economical and carefree resource management. On the other hand, public clouds do not guarantee the rigorous security and privacy of computations as well as stored data. Private clouds provide security and privacy of data as well as computations, due to users' ability to physically and electronically constrain data access and execution of computations. However, the user of the private cloud manages the nodes, updates software, and replaces the failed nodes. Such management is time consuming and incurs huge monetary cost.

Our focus is on the security and privacy issues of MapReduce environment in public or hybrid clouds. Private cloud environments are more secure due to a physical security of the cloud. Many of the reviewed below results are applicable to both public and hybrid clouds, unless stated otherwise (for instance, see hybrid cloud specific research in [5,6] and Section 4.4.1). Even though there is a plethora of additional projects and frameworks that add functionality on top of MapReduce (see Apache Hive [7], Cloudera Impala,[1] HBase [8], Apache Zookeeper,[2] Thrift,[3] and Apache Solr[4]), this paper only reviews security related projects in Section 2 (readers interested in security and privacy issues of other projects may refer to [9]).

Security aspects in the context of MapReduce are crucial in order to authenticate and authorize users, auditing–confidentiality–integrity of data and computation, availability

[1] http://impala.io/.
[2] https://zookeeper.apache.org/.
[3] https://thrift.apache.org/.
[4] http://lucene.apache.org/solr/.

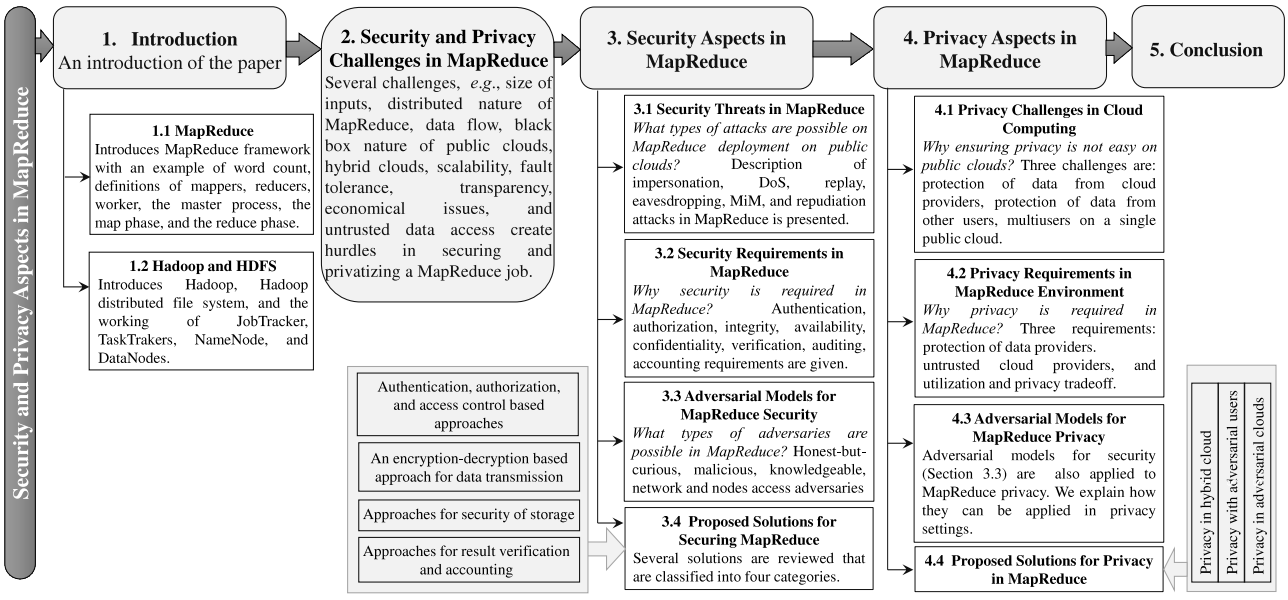


Fig. 1 – Schematic map of the paper.

of mappers and reducers, and verification of outputs. Security of MapReduce ensures a *legitimate* functionality of the framework. A secure MapReduce framework deals with the following attacks: attacks on authentication (impersonation and replay attacks), attacks on confidentiality (eavesdropping and man-in-the-middle attacks), data tampering (modification of input data, intermediate outputs, and the final outputs), hardware tampering, software tampering (modification of mappers and reducers), denial-of-service, interception-release of data as well as computations, and communication analysis.

On the contrary, privacy aspects assume *legitimate* functionality of the framework and thus, are built on top of security. On top of the correctly functioning framework, privacy in the context of MapReduce is an ability of each participating party (data providers, cloud providers, and users) to prevent other, possibly adversarial parties from observing data, codes, computations, and outputs. In order to ensure privacy, a MapReduce algorithm in public clouds hides data storage as well as the computation to public clouds and adversarial users. Additional distinction between security and privacy is that security is much more of a binary issue, *i.e.*, either the attack succeeds or not, whereas in a privacy setting there is a tradeoff between privacy of the data and utilization of the framework.

Scope and outline of the paper. In this paper, we discuss the challenges, requirements, adversarial models, attack scenarios and proposed solutions to the security and privacy concerns in the context of MapReduce (see Fig. 1). Overview of MapReduce environment and its open-source software framework, Apache Hadoop, are given in Sections 1.1 and 1.2, respectively. In Section 2, we discuss security challenges involved in a MapReduce computation.

Security aspects of MapReduce are presented in Section 3, where we present security threats in Section 3.1, requirements of security in MapReduce computations in Section 3.2, adversarial models in Section 3.3, and some proposed solutions for security in MapReduce in Section 3.4. Privacy aspects in MapReduce are presented in Section 4. We first outline privacy aspects in clouds (Section 4.1) and privacy requirements in MapReduce in Section 4.2. Then, we present the common adversarial models in the context of privacy in MapReduce in Section 4.3, and some proposed solutions to privacy in MapReduce in Section 4.4. We conclude and outline important research issues in Section 5.

We would like to emphasize here that we focus on the security and privacy issues of MapReduce framework. Despite advantages of MapReduce deployments in public clouds, they also bring new hurdles in the form of security, data privacy, and computation privacy. Even though the new challenges are mainly due to the public nature of the cloud, ownership separation of platform and data, location of the service, and etc., those challenges are different from security and privacy challenges of a general cloud computing. General security issues in the cloud such as: security of virtual machines and hypervisors, security of services and Service-Level Agreement (SLA), regulations and organization policies, and general availability in the cloud are not discussed in detail, and the interested reader may refer to [10–16] for security and privacy in the cloud. Also note that we do not consider security and privacy of MapReduce scheduling algorithms, rather than we encourage readers to have an understanding of security and privacy of mappers, reducers, and data flow.

1.1. *MapReduce framework*

Parallel processing of large-scale data provides outputs in a timely manner. However, it constrains the computation due to node failure, ordering of outputs, system scalability, transparency, load balancing, fault tolerance, and synchronization among the nodes. MapReduce [4] solves these issues and executes parallel processing using a cluster of computing nodes over large-scale data, but without considering security and privacy of data and computations. Here, we provide an

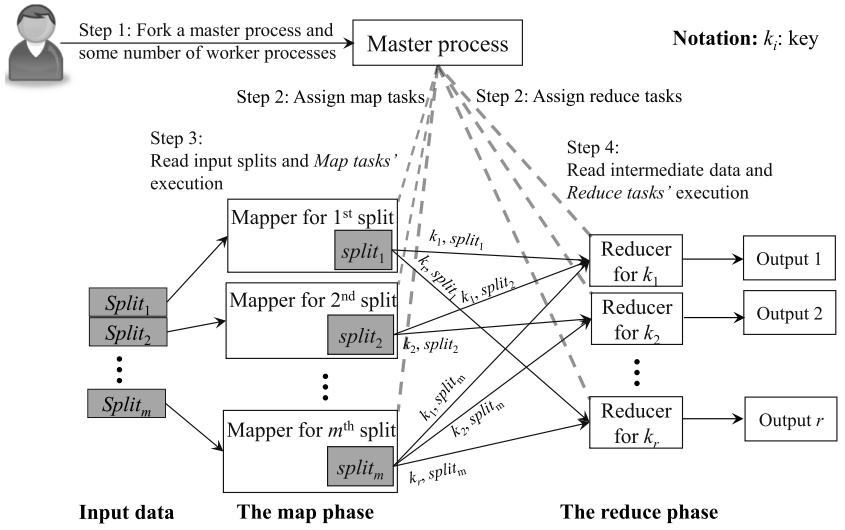


Fig. 2 – A general execution of a MapReduce algorithm.

overview of MapReduce framework, details may be found in Chapter 2 of [17].

A user-defined program forks a *master process* and *worker processes* at different nodes; see Fig. 2. The master process creates and assigns *map tasks* and *reduce tasks* to idle worker processes. A worker process deals with either a map task or a reduce task. The worker processes that handle the map tasks and the reduce tasks are called *map workers* and *reduce workers*, respectively. A MapReduce computation consists of the *Map phase* and the *Reduce phase*, where two user-defined functions, namely the *map function* and the *reduce function*, are executed over (large-scale) data, which is represented in the form of $\langle key, value\rangle$ pairs.

The map phase. The given input data is processed in the map phase, where the map function is applied to data and produces intermediate outputs (of the form $\langle key, value\rangle$), where the number of bits needed to describe the *value* in each $\langle key, value\rangle$ pair is not necessarily identical [18]. The application of the map function to a single input (for example, a tuple of a relational database or a node in a graph) is called a *mapper*.

The reduce phase. The Reduce phase provides the final output of MapReduce computations. The Reduce phase executes the reduce function on intermediate outputs. The application of the reduce function to a single *key* and its associated list of *values* is called a *reducer*.

Word count example. Word count is a traditional example to illustrate a MapReduce computation, where the task is to count the number of occurrences of each word in a collection of documents. The original input data is a collection of documents. Each mapper takes a document and implements a map function that results in a set of $\langle key, value\rangle$ pairs $(\{\langle w_1, 1\rangle, \langle w_2, 1\rangle, \ldots, \langle w_n, 1\rangle\})$, where each key, w_i, represents a word in the document, and each value is 1. The reduce task is executed subsequently, where the reduce function adds up all the values corresponding to a key. Specifically, a reducer for a key w_i takes all the $\langle key, value\rangle$ pairs corresponding to the key (or word) w_i and outputs a $\langle w_i, m\rangle$ pair, where m is the total number of occurrences of the word w_i in all the given documents.

Applications and models of MapReduce. Many applications in different areas exist already for MapReduce. Among them: matrix multiplication [19], similarity join [20–23], detection of near-duplicates [24], interval join [25,26], spatial join [27–29], graph processing [30,31], pattern matching [32], data cube processing [33–36], skyline queries [37], k-nearest-neighbors finding [38,39], star-join [40], theta-join [41,42], and image–audio–video–graph processing [43], are a few applications of MapReduce in the real world. Some research models for efficient MapReduce computation are presented by Karloff et al. [44], Goodrich [45], Lattanzi et al. [46], Pietracaprina et al. [47], Goel and Munagala [48], Ullman [49], Afrati et al. [50,51,18,52], and Fish et al. [53].

1.2. *Hadoop and HDFS*

Apache Hadoop[5] is the most known and widely used open-source software implementation of MapReduce for distributed storage and distributed processing of large-scale data on clusters of nodes. Hadoop includes three major components, as follows: (i) Hadoop Distributed File System (HDFS) [54]: a scalable and fault-tolerant distributed storage system, (ii) Hadoop MapReduce, and (iii) Hadoop Common, the common utilities, which support the other Hadoop modules. Hadoop 2.x, released in 2013, has changed the low-level architecture by separating resource management from job management (see YARN[6]). However, the architectural modification does not change the high-level of the described MapReduce job, and thus, is not considered here.

Hadoop cluster consists of a master node (that runs a JobTracker and a NameNode) and several slave nodes (where each slave node runs a TaskTracker and a DataNode); see

[5] http://hadoop.apache.org/.

[6] http://hadoop.apache.org/docs/r2.6.0/hadoop-yarn/hadoop-yarn-site/YARN.html.

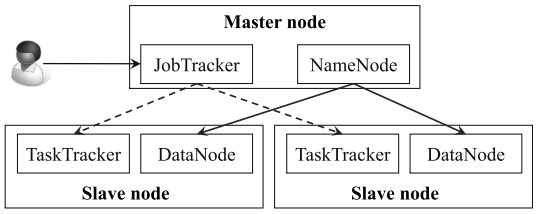


Fig. 3 – Structure of a Hadoop cluster with one master node and two slave nodes.

Fig. 3. JobTracker and TaskTrackers provide an environment for a MapReduce job execution. Specifically, JobTracker accepts a MapReduce job from a user, executes the job on (free) TaskTrackers, receives outputs from TaskTrackers, and provides outputs to the user. TaskTrackers execute assigned jobs, provide outputs to JobTracker, and periodically send heartbeat messages to JobTracker to show its presence and workload.

NameNode and DataNodes provide a distributed file system, called Hadoop Distributed File System, which supports read, write and delete operations on files, and create and delete operations on directories. NameNode manages the cluster metadata and DataNodes, which store the data. NameNode keeps information of files and directories using *inodes*. Inodes store several attributes of files and directories, *e.g.*, permissions, modification and access times, and disk space quotas. In HDFS, data is divided into small splits, called *blocks*, (64 and 128 MB are most commonly used sizes). Each block is independently replicated at multiple DataNodes, and block replicas are processed by mappers and reducers. Each DataNode stores the corresponding block replicas, and two files in the local host's native file system are used to represent each block replica. The first file holds the data itself, and the second file holds the block's metadata. More details about Hadoop and HDFS may be found in Chapter 2 of [55].

As the review focuses on security of MapReduce, it is fitting to provide a short overview of Hadoop and HDFS security features as well. MapReduce security is discussed later on in corresponding sections. By default, Hadoop provides no authentication making it easy to perform destructive changes and possible attacks on other users or computing cluster. While it is possible to configure Hadoop cluster with Kerberos authentication mechanism [56–58], extra work is required to do that [59]. In a similar way, default configuration of HDFS provides a basic protection for the saved data by following Unix access control. However, within HDFS cluster, by default, there is no mechanism for identifying a user or group, though the user is trusted to present himself correctly. Clearly, this makes it very easy for an adversarial client to read and modify data belonging to other users. Just like Hadoop framework, HDFS can be configured to determine user identity by its Kerberos credentials. However, just like in Hadoop's case, the configuration requires additional work and as such, most likely is not performed in all installations.

As mentioned, the job is done to provide security facilities for Hadoop (SecureMode [59]). The facilities include: user and service authentication (based on Kerberos mechanism), authentication for Web consoles and data confidentiality. The data confidentiality consists from features that encrypt data in-transit during Hadoop calls, including data encryption on RPC, block data transfer and HTTP access. Just like the authentication and access control mechanisms, data confidentiality requires configuration and is not configured by default.

2. Security and privacy challenges in MapReduce

The massive parallel processing style of MapReduce is substantially different from the classical computation in the cloud leading to distinct design challenges for security and privacy requirements. In this section, we present specific security and privacy design challenges for MapReduce computations in the cloud.

Size of input data and its storage. Input to a MapReduce job, big-data, which is described by the 4Vs: volume, velocity, variety, and veracity, implies a major challenge in securing MapReduce computations. Security and privacy techniques for processing big-data have to deal with huge amount of data, possibly arriving at high speed from different sources. Moreover, in MapReduce computations, data is partitioned into small-sized splits that are replicated and distributed to several nodes. Each split has to be transferred in a secure and private manner. This replicated and distributed nature constitutes unique challenges in terms of data storage security, as compared to a system that holds the whole data in a single place.

Highly distributed nature of MapReduce computations. The cloud computing itself does not necessarily imply distributed computations. However, MapReduce does require large clusters of nodes that can distributively process replicated data in parallel. The deployment of MapReduce in the cloud requires mechanisms for protecting a large-number of nodes and data, which may be in-transit and may be at the rest at the nodes. In particular, distributed processing over replicated data has a higher probability for attacks as compared to a centralized system, since attackers have a much wider range of targets to choose from. A single adversarial mapper or reducer, out of several mappers and reducers, may provide wrong outputs, copy data for future usage, modify input data, leak confidential data to a third party, or send the whole data to another user. Identifying a (single) adversarial mapper or reducer is not an easy task in the scope of MapReduce.

Data flow. MapReduce computations require a complex data flow among different storage nodes, different computing nodes, and different clouds, as follows:

- *Between data storage and computing nodes*: MapReduce computations are executed near the location of data to minimize data flow; however, ensuring an identical location of data and mappers–reducers cannot always be guaranteed. Thus, computations are typically executed at the nodes that are not storage nodes. This leads to data flow from storage to computing nodes. Moreover, some providers separate the computational cloud from the storage cloud; for example Amazon Elastic MapReduce uses two different clouds: one is for executing a MapReduce computation and the other is for storing data.

This dual cloud structure requires constant data flow between the clouds. Data flow becomes more complex when organizations perform MapReduce computations in the hybrid clouds, where it is necessary for sensitive data (*e.g.*, some attributes of a relation, financial data, or health records) to remain in private clouds and do not reach public clouds; while all the other data, called non-sensitive data (*e.g.*, all the attributes of a relation except some attributes holding critical information), may be processed in public clouds.

- *Between public clouds*: A MapReduce computation may be executed in more than one public cloud [60]. In that case, data flow may occur between two master processes at two different clouds, between two mappers at two different clouds, between two reducers at two different clouds, and a mapper and a reducer at two different clouds. Such scenarios involve data flow over a public network, which is vulnerable to attacks.

The black-box nature of public clouds supporting MapReduce. Public clouds supporting MapReduce do not provide any information regarding the deployment, configuration and execution of mappers and reducers. PaaS infrastructure deploys and configures mappers and reducers dynamically for each computation. This opaque view of MapReduce in public clouds prohibits an efficient execution of a MapReduce computation in terms of the *communication cost* (the total amount of bits that are transferred between the map phase and the reduce phase) and the replication rate [18] (the average number of key–value pairs created for each input). On the upside, the automatic resource management of public clouds offloads the user's burden. Hence, secure MapReduce computations should allow dynamic deployment and configurations of mappers and reducers in public clouds without increasing the communication cost.

Hybrid cloud. The hybrid cloud provides an efficient processing of sensitive and non-sensitive data. The hybrid clouds enjoy efficient and economical resource management (provided by public clouds) with security and privacy of sensitive data (provided by the private cloud). Unfortunately, MapReduce is designed to work in a single cloud, and this characteristic poses additional challenges in supporting a hybrid cloud deployment [5]. In addition, data sanitation, *i.e.*, separation of sensitive and non-sensitive data, and arrangement of outputs from different clouds are additional challenges to a MapReduce computation in the hybrid clouds.

Scalability, fault tolerance, and transparency. MapReduce provides an efficient, scalable, distributed, and fault-free processing of replicated data in parallel. An integration of security and privacy mechanisms should not reduce efficiency, scalability, and fault tolerance of MapReduce algorithms. Also, the involvement of security and privacy protocols in MapReduce must be transparent to users, without any modification of the map and reduce functions.

Economical issues. An execution of MapReduce computations in public clouds is tariffed mainly for three economical factors—data storage, the communication cost, and computation time. MapReduce algorithms also regard these three economical factors. Therefore, security and privacy mechanisms must be economically incorporated into MapReduce computations.

Untrusted data access. MapReduce allows great flexibility in enabling user defined computations; but at the same time, implies a great trust in users for providing mapper and reducer codes that do not impact MapReduce cluster, in terms of slowing/corrupting the entire job, modifying/deleting data, and other unwanted read/write operations. Security and privacy algorithms for MapReduce should be developed to cope with corrupted or even adversarial codes, protecting data, and limiting data access of corrupted mappers and reducers.

All the above challenges to MapReduce framework in clouds indicate new security and privacy requirements. In the next sections, we discuss the requirements of security in MapReduce.

3. Security aspects in MapReduce

The security of data and computations plays a significant role in MapReduce computations on both hybrid and public clouds. Without security, MapReduce computations as well as MapReduce infrastructures can be affected by several types of attacks. In this section, we present security threats and security requirements for MapReduce computations. Notice that even though some security threats and security requirements are common for MapReduce and for generic cloud computing, we will focus on security threats and security requirements in the context of MapReduce. Following that, we provide a brief summary of some existing security algorithms for MapReduce.

3.1. *Security threats in MapReduce*

In this section, we present security threats that can harm a MapReduce computation and the framework in the absence of secure MapReduce environment. Distributed and replicated data processing in MapReduce open an opportunity for a wide range of attacks. While those attacks follow the same ideas as attacks in different cloud computation models, the exact application is different for MapReduce paradigm.

Notice that most of these attacks are specific to MapReduce deployments in public clouds, as physical security and separation of private cloud deployments significantly reduce the risk of attacks and allow a physical separation of resources from attackers.

Impersonation attack. *Definition*: An impersonation attack occurs when an adversary successfully pretends to be a legitimate user of a system by a brute-force attack on weak passwords, weak encryption schemes, or other means.

MapReduce context: After a successful impersonation attack an adversary can act on behalf of a legal user and can execute MapReduce jobs that may result in data leakage, data and computations tampering, or wrong computations on data [61–63]. Moreover, on public clouds under impersonation attack, an attacker may perform MapReduce computations while the impersonated user is tariffed for data storage, the communication cost and computation time.

Denial-of-Service (DoS) attack. *Definition*: A DoS attack occurs when an adversary causes a system and the network to become non-functional and non-accessible by legitimate users.

MapReduce context: A DoS attack occurs when an adversary makes a node, mapper, or reducer to be non-functional and non-accessible by executing undesirable and useless tasks [64]. Moreover, a compromised node, mapper, or reducer may result in non-functionality of other non-compromised nodes, mappers, or reducers by repeatedly sending task requests for them to execute [64]. In addition, if an attacker compromises enough nodes in a Hadoop cluster, it may result in the failure of the whole MapReduce framework and the network overload [65].

Replay attack. *Definition*: A replay attack occurs when an adversary resends (or replays) a captured valid message to the nodes.

MapReduce context: A replay attack occurs when an adversary assigns some old tasks to the nodes, making them continuously busy [64]. In addition, an adversary can replay users' credentials to access the framework, and it may lead to impersonation [66,67] and DoS attacks by spinning excessive amount of nodes.

Eavesdropping. *Definition*: An eavesdropping attack occurs when an adversary (passively) monitors the network and the nodes without consent.

MapReduce context: An eavesdropping attack occurs when an adversary observes input data, intermediate outputs, the final outputs, and MapReduce computations without any consent from the data and computation's owner [60,64,68].

Man-in-the-Middle (MiM) attacks. *Definition*: In MiM, an adversary (actively) modifies, corrupts, or inserts data passing between two legitimate users of a system.

MapReduce context: A MiM attack occurs when an adversary modifies or corrupts the computing codes, input data, intermediate outputs, or the final outputs passing between any two legitimate nodes of the framework [60,64,68]. Moreover, tampering with mappers or reducers may lead to DoS, impersonation, and replay attacks [64].

Repudiation. *Definition*: A repudiation attack occurs when a node falsely denies processing a sent message or a task execution.

MapReduce context: A repudiation attack occurs when a mapper or reducer falsely denies an execution request that it already performed [63].

Despite the severity of all the above mentioned attacks, by default, MapReduce does not provide any way to encounter them. (Details of the above mentioned attacks on the computer networks are given in [69,66].) In the next section, we present security requirements in MapReduce.

3.2. *Security requirements in MapReduce*

Dynamic deployment and configuration of mappers and reducers for each MapReduce computation specify distinct security requirements in MapReduce as compared to the classical parallel processing and cloud computing. Specifically, the allocation of different heterogeneous resources for different MapReduce computations, variability in the locations of resources, and different trust levels for different jobs are a few parameters that complicate the security of MapReduce. In order to overcome security challenges in MapReduce computations, a secure MapReduce must provide authenticated and authorized access, confidentiality (a secure storage and computation), integrity (a fair execution and storage), and accounting–auditing of data and computations. Next, we present the security requirements needed in a MapReduce computation. Most of the security requirements and corresponding cloud layers are depicted in Fig. 4. As can be seen from Fig. 4, we do not deal with issues of security above the MapReduce layer such as those of Pig, Hive, or big-data applications.

Authentication, authorization, and access control of mappers and reducers. Authentication provides a way to identify an adversarial mapper, reducer, or user. In other words, authentication is a process by which only those mappers and reducers that have rights to process data perform assigned tasks, and all the other mappers and reducers who are not allowed to access data and the framework are denied. Once mappers and reducers are authenticated, authorization of mappers and reducers allows them to access and process data by investigating their access privileges to that data. Access control provides pre-configured policies that restrict an unauthorized user to access data and to access the framework. An adversarial mapper, reducer, or user *mimics* a legal mapper, reducer, or user, while breaching authentication and authorization. Attacks on authentication and authorization mechanisms are impersonation and replay attacks. The framework and data cannot be processed by any adversarial mapper, reducer, and user, when authentication, authorization, and access control of mappers and reducers are functioning properly.

Availability of data, mappers, and reducers. Data, mappers, and reducers should be available to authenticated and authorized users without delay. An adversarial code may make mappers–reducers and the network too busy so that they cannot process data and available to transfer data, respectively. An attack on availability of data, mappers, and reducers can *interrupt* MapReduce computations. Specifically, an attack such as Denial-of-Service, is an attack on availability of data, mappers, and reducers. As an example, a single adversarial user in multi-users cloud-based MapReduce environment can considerably impact job completion times of the entire cluster even when using less than 10% of cluster resources [65].

Confidentiality of computations and data. Confidentiality of computations and data refers to the protection of computations and data, which may be in-transit from the user's location to the location of computations or may be stored at public clouds, from unauthorized users and the public cloud itself. An attack on confidentiality of computations and data is *interception* (*i.e.*, eavesdropping and man-in-the-middle attacks). By ensuring confidentiality, one cannot intercept computations and data during the transmission and following transmission on public clouds themselves.

Integrity of computations and data. Integrity of a computation refers to a *fair* transmission (of a MapReduce computation from the user's location to the locations of the

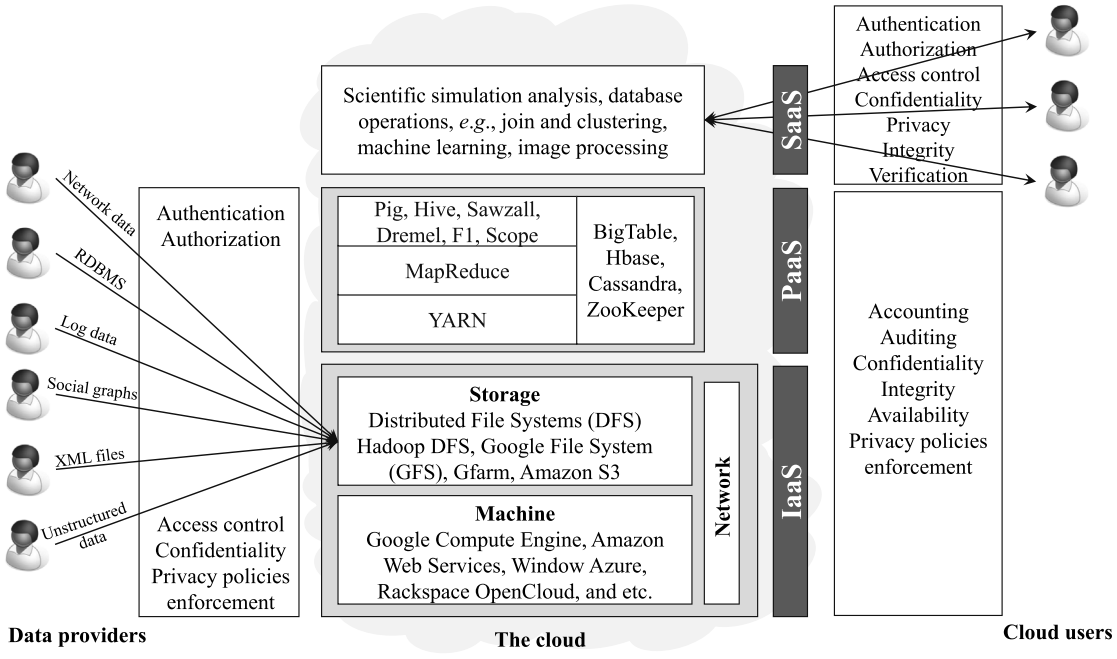


Fig. 4 – Security requirements in MapReduce environment on the cloud. The figure shows a complete picture of the considered cloud structure, with different participating parties: data providers on the left, cloud provider in the middle and users on the right, and their specific security requirements. The figure also depicts various cloud levels and their relation to the security mechanisms.

computation) and execution of mappers and reducers. Similarly, integrity of data refers to a *fair* transmission and storage of data. An attack on integrity of computations and data modifies computations or the data. To overcome such attacks, integrity should be preserved such that any modification and loss of computations and data by public clouds or an adversarial user is detected [70,64]. However, the amount of data hurdles to check the integrity of computations and the integrity of the whole data at a single computing node [64].

Verification of outputs. Verification of outputs is a difficult task in MapReduce due to a massive parallel processing and a huge amount of input/output data. Verification ensures completeness (*i.e.*, all the possible outputs are produced by mappers and reducers by processing of assigned input data without any tampering), correctness (*i.e.*, all the outputs are legitimate and generated from assigned input data without any tampering), and freshness (*i.e.*, all the outputs are new, not containing any old output, and generated from input data without any tampering) of outputs [71].

Accounting and auditing of computations and data. Accounting of computations and data assists in locating mappers or reducers who are taking adversary actions over data with verifiable evidences [70]. Auditing of computations and data produces details of actions taken by mappers and reducers over data. Specifically, auditing investigates accounted data and provides verifiable evidences of what, when, initiated by whom, and how actions happened over which part of the data. An auditing process involves three parties, the first is data providers, the second is public clouds, and the third is the auditor who performs auditing. Hence, it is required to restrict data with fine-grained access controls, because an attacker who impersonates a legitimate auditor may understand the computation (better than the auditor) and reveal sensitive data.

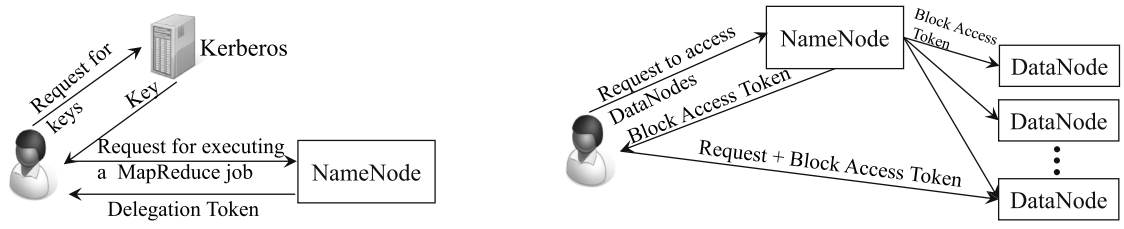
3.3. *Adversarial models for MapReduce security*

There are numerous possible adversarial models in security, and thus, we will concentrate only on a limited set of adversarial models involved in MapReduce security.

Honest-but-curious adversary. An honest-but-curious adversary executes a MapReduce job correctly and does not interfere with the job as well as data; however, it performs some extra computations for understanding the whole data and the job. For example, a public cloud does not interfere a MapReduce job and data, but it can observe the job and data. This type of adversary is also relevant to Privacy in MapReduce, see Section 4.3.

Malicious adversary. A malicious adversary can execute any computation for stealing, corrupting, and modifying data as well as the original MapReduce computation [68,72,62,70]. For example, a malicious mapper or reducer may not perform a computation or may provide wrong outputs (to users).

Malicious adversaries in distributed settings, like MapReduce, are divided into two types, as follows: (i) *non-collusive malicious adversary*: a non-collusive malicious adversary works independently, and hence, provides wrong outputs without consulting other malicious adversaries. In this case, if an identical task is assigned to two nodes and at least one of them is non-collusive, then the malicious behavior of the node can be easily detected by comparing their outputs; (ii) *collusive malicious adversary*: a collusive malicious adversary



(a) Delegation token. **(b) Block access token.**

Fig. 5 – Authentication to Hadoop using Kerberos and two types of tokens.

communicates with all the other malicious adversaries before providing outputs [73,74,64]. In this case, when a collusive (malicious) adversary is assigned a task, it consults other collusive adversaries to find if they are assigned an identical task. If yes, then all the collusive adversaries provide an identical wrong output, which make it harder to detect them.

Note that honest-but-curious and malicious adversaries are assumed to be polynomial-computationally-bounded adversaries, *i.e.*, the adversaries cannot perform brute-force attack.

Knowledgeable adversary. A knowledgeable adversary is assumed to be knowledgeable with respect to the cloud structure, MapReduce algorithms, and implementation of mappers–reducers. In other words, a knowledgeable adversary, which may be the cloud provider or any user, is considered to be capable of leveraging any security (and privacy) threats in the framework.

Network and nodes access adversary. A network and nodes access adversary is assumed to have an access to network and nodes, though it does not have privileged accounts on the nodes. Privileged account on the nodes can be compared to the adversary owning the cloud, which is a situation where cloud user cannot be protected.

3.4. *Proposed solutions for securing MapReduce*

In this section, we provide a brief description of some existing security solutions for MapReduce. Reviewed security algorithms, protocols and frameworks are summarized in Table 1.

3.4.1. *Authentication, authorization, and access control based approaches*

An authentication mechanism for Hadoop using Kerberos and three special types of tokens, namely *delegation token*, *block access token*, and *job token*, is presented [57,58]. The communication between a user and HDFS is divided into two parts: (i) a user accesses NameNode using Hadoop's remote procedure call (RPC) libraries, and all RPCs connect using *Simple Authentication and Security Layer* that uses Kerberos, DIGEST-MD5, or a delegation token and (ii) a user accesses DataNodes using a streaming socket connection that is secured using a block access token. The working of the three types of tokens is as follows:

Delegation token. A delegation token is a secret key between a user and NameNode. After authenticating a user, a delegation token is generated by NameNode using Kerberos, and the token is used for subsequent authentication of the user by NameNode without involving Kerberos; see Fig. 5(a).

Block access token. A block access token provides authentication policies to DataNodes by passing authorization information from NameNode to DataNodes and is generated by NameNode using a symmetric-key scheme where NameNode and all of the DataNodes share a secret key; see Fig. 5(b). When a user wants to access a file in HDFS, it requests NameNode for block ids and locations of the file. In response, NameNode sends block ids and the locations with a block access token for each block, if NameNode verifies authenticity and authority of the user. The user sends the block id with the block access token to the corresponding DataNode, and the DataNode verifies the block access token before allowing access to that block.

Job token. A job token is generated by JobTracker in the form of a secret key, when a MapReduce job is submitted. JobTraker stores the job token for each job as a part of the job and distributes it to TaskTrackers. The job token is used to authenticate tasks at TaskTrackers. MapReduce paradigm itself is security agnostic, and this lead to an initial version of Hadoop implementation to have no built-in security mechanism. However, as MapReduce in general and, specifically, Hadoop gained wide-spread usage, the need for security has become more and more acute. This led to a number of recent projects, which we will review next, trying to add different types of security aspects to Hadoop.

Apache Knox. Apache Knox [75] is a stateless reverse proxy framework that provides gateway-level security and a single access point to a single Hadoop cluster or multiple Hadoop clusters. Apache Knox provides a monitoring of the system, authentication, federation of authenticated users, authorization, and auditing. It has several advantages, such as: integration with enterprise identity management solutions, it hides details of Hadoop cluster deployment, simplifies the number of services that clients need to interact with, limits numbers of access point to Hadoop clusters, scales linearly by adding more Knox nodes as the load increases.

Apache Sentry. Apache Sentry [93] is a system for fine-grained, multi-tenant administration, and role-based authorization of an access to data and metadata in a Hadoop cluster. Apache Sentry can be integrated in relational data model for Apache Hive, Cloudera Impala, and hierarchical data model used by Apache Solr. Sentry allows access control at the server, database, table, and view scopes. It also allows

Table 1 – Summary of security algorithms, protocols, and frameworks for MapReduce.

Algorithms/protocols/ frameworks	Authentication, authorization	Access control	Confidentiality		Integrity		Availability	Accounting	Auditing	Cloud structure	Attack handling					
			Data	Computation	Data	Computation					Impersonation	DoS	Replay	Eavesdropping	MiM	Repudiation
[57,58]	✓	✓								S[a]	✓					
Apache Knox [75]	✓	✓							✓	M[b]						
Apache Sentry [76]	✓	✓								S						
Apache Ranger [76]	✓	✓	✓						✓	M						
Project Rhino [77]	✓	✓	✓						✓	S	✓					
Apache Accumulo [78]		✓								S						
Airavat [79]	✓	✓	✓							S	✓					
Khaled et al. [80]		✓														
Vigiles [81]	✓	✓														
GuardMR [82]	✓	✓														
G-Hadoop [83]	✓	✓									✓		✓		✓	
SecDM [60]	✓	✓	✓	✓	✓	✓				M				✓	✓	
iBigTable [71]					✓	✓					✓					
Lin et al. [84]		✓								S				✓[c]	✓[d]	
SAPSC [68]	✓									M				✓		
ClusterBFT [6]					✓					S, H[e]	✓					
Moca et al. [85]					✓	✓										✓
SecureMR [64]			✓		✓	✓	✓			S	✓	✓	✓	✓		✓
AccountableMR [70]					✓	✓		✓		S						
VIAF [73]					✓	✓				S	✓					
CCMR [86]					✓	✓				M	✓					
IntegrityMR [74]					✓	✓				M	✓					
VAWS [87]					✓	✓				S	✓					
Hatman [88]					✓					S					✓[f]	
TrustMR [89]						✓				S						
TS-TRV [72]					✓	✓				S	✓					
Log-based [62]	✓				✓	✓		✓	✓	S	✓					
Watermarking based [90,91]					✓	✓				S	✓					
Accountable MapReduce (RBAC) [92]	✓	✓	✓					✓	✓	S						

[a] S: Single cloud.
[b] M: Multiple clouds.
[c] Partially.
[d] Partially.
[e] H: Hybrid cloud.
[f] By assumptions of the algorithms.

different privileges for `select`, `insert`, `create`, and `modify`. Sentry defines policies for accessing resources. On receiving a request from a user, Hive/Impala/Solr asks Sentry for validating the request. Sentry builds a map of privileges allowed for the requesting user and then determines whether the given request should be allowed, and then allows or prohibits the user access based on decisions by Sentry.

Apache Ranger. Apache Ranger [76] provides a centralized and comprehensive platform for securing Hadoop. Specially, Apache Ranger provides: (i) authentication: by Kerberos and secured by Apache Knox, (ii) authorization, (iii) fine-grained access control: by role-based access control, attribute-based access control, etc., (iv) auditing of HDFS, Hive, and HBase, and (v) data protection: by wire encryption, volume encryption, and file/column encryption.

Project Rhino. Project Rhino [77], an initiative by Cloudera and Intel, is an open source for enhancing existing data protection in the Hadoop *stack*. Specifically, Project Rhino provides framework support for encryption, key management, and a common authentication–authorization module with single sign on. Recently, Project Rhino added cell level encryption and fine-grained access control to HBase 0.98, and encryption to data at-rest (data stored on persistent storage) in Apache Hadoop. Note that data encryption in Hadoop requires encryption of data at-rest and in-transit; however, except Project Rhino, most Hadoop components provide encryption for data in-transit only.

Apache Accumulo. Apache Accumulo [78] is not a security framework like Apache Knox, Apache Sentry, Apache Ranger, and Project Rhino. However, Apache Accumulo improves Google BigTable [94] design by introducing *cell-based access control*, which emphasizes us to mention Apache Accumulo in this paper. Apache Accumulo stores a logical combination of security labels that must be satisfied at query time in order for keys and values to be returned as part of a user request. This allows users to see only those keys and values for which they are authorized. Apache Accumulo is built on top of Apache Hadoop, Zookeeper, and Thrift.

Airavat. Airavat [79] is a system that provides mandatory access control together with differential privacy for data protection. Airavat is the first system that provides a complete solution for data privacy and secure computations in MapReduce environments. To ensure that untrusted mappers do not leak data outside the cluster, Airavat uses mandatory control system to disallow direct access of mappers to the data and to the network. The system is based on Security-Enhanced Linux (SELinux)[7] and requires a unified deployment of the secure Linux system on all the nodes of the cluster.

Vigiles. Vigiles [81] provides a fine-grained access control mechanism for read operations in MapReduce without modifying computations. Vigiles works as a middleware architecture that stays between untrusted users and MapReduce environment; see Fig. 6. Since Vigiles supports only read operation, it is required to remove sensitive data from outputs (of the read operation). A system administrator manages

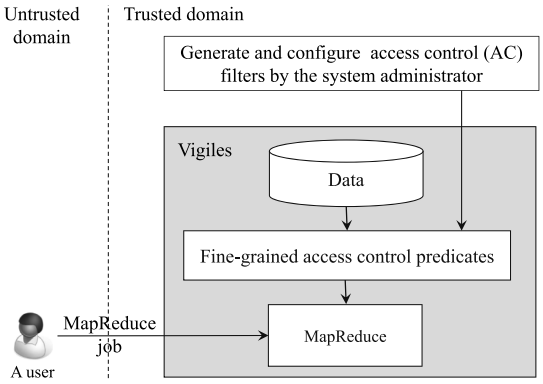


Fig. 6 – Vigiles access control mechanism.

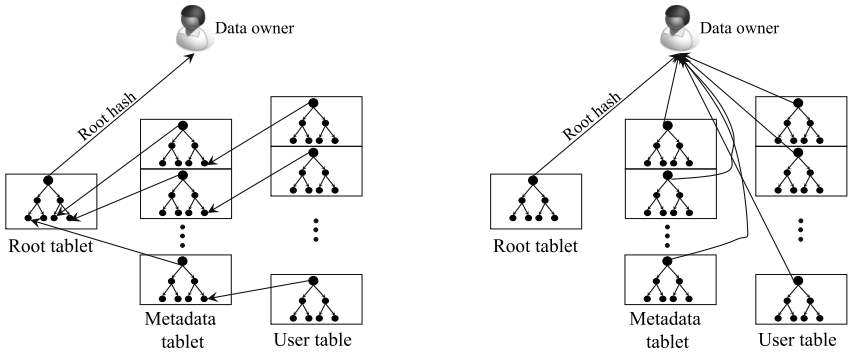
HDFS and creates access control filters, which provide only authorized data to users, based on the given configurations. Access control filters are used to remove sensitive data by following a 3-phase procedure: decompose, fetch, and action. When users provide MapReduce computations, Vigiles executes them by following access control policies and remove sensitive data from the outputs. On the negative side, ad-hoc data types, append and delete operations are not allowed in Vigiles. Nevertheless, it does not require any modification of a MapReduce computation.

GuardMR. GuardMR [82] allows ad-hoc data types and provides access to the record dynamically. GuardMR is composed of two main components: (i) access control module performs the administrative functions that allow to add new data types and preprocessing functions after a proper security analysis; and (ii) reference monitor enforces specified security policies to the underlying MapReduce system after consulting the access control module, resulting in an authorized view of data.

In [80], an access control and enforcement policy based architecture for Resource Description Framework (RDF) [95,96] is proposed, where the system administrator generates an access token for securely accessing data based on the request of users. The token prevents access to the entire data. Six types of secure data accesses are suggested: predicate data access, subject and object data access with or without predicates, and subject model level access. Since the format of RDF is not suitable for a MapReduce computation, a two-layered system is also proposed, where the first layer (data processing layer) converts RDF to N-Triple format [97], and the second layer (query processing layer) is responsible for executing a MapReduce computation. The query processing layer provides outputs regarding an access token. The query processing layer first rewrites a query that satisfies an access token, then performs a MapReduce job according to the rewritten query, and finally performs one or more additional MapReduce jobs to remove sensitive data from outputs according to the access token.

Authentication of malicious users based on storing communication between user and NameNode, and between user and DataNodes is suggested in [62]. The approach stores IP addresses, port numbers, and socket connection related system calls. In [83], a security model for geographically

[7] http://selinuxproject.org/.



(a) The first (centralized) approach.

(b) The second (distributed) approach.

Fig. 7 – Merkle Hash Tree based authenticated storage of data structures in iBigTable.

distributed Hadoop, called G-Hadoop [98], is presented. The model uses two types of tokens, namely proxy token and slave token for the purpose of authentication. A proxy token (contains its expiration time, identity of certificate authority (CA) server, the public key of the master process, and a random message generated by the CA server) is used by slave nodes for authenticating the master process. A slave token (contains identity of the CA server and the public key of the corresponding slave node) is used by the master process for authenticating a slave node. In [99], the authors suggested a honey-pot-based mechanism for detecting an unauthorized data access. Honey data is deliberately produced and mixed in the original data. However, an authorized user never accesses the honey-pot data during a MapReduce job. Since attackers access all the parts of data, it leads to an alarm with a high probability.

3.4.2. *An encryption–decryption based approach for data transmission*

A secure data transmission and security storage of the data (reviewed in Section 4) are critical issues when mappers and reducers that reside on two different clouds share data.

SecDM. Secure Data Migration (SecDM) [60] provides a secure way for data transmission among mappers–reducers at two different clouds. The two master processes, which are located at two different clouds, create a Secure Socket Layer (SSL) connection between them, then send message authentication code with a timestamp and negotiate a random key. After authentication of two master processes is completed, mappers or reducers receive the locations of data in the other cloud. Data transmission (among mappers–reducers at two clouds) is carried out using the negotiated key, where encrypted data is transmitted with a hash value of data and a message authentication code, which prevents tampering.

3.4.3. *Approaches for security and integrity of storage*

iBigTable. An enhancement of BigTable, called iBigTable [71], ensures the integrity of data using decentralized authenticated data structures. Two approaches, see Fig. 7(a) and (b), are suggested for storing authenticated data that is used to verify the integrity of data. Both the approaches build a Merkle Hash Tree (MHT) [100] based authenticated data structure for the root tablet, the metadata tablets, and the user tables. MHT allows efficient and secure verification of contents of large-scale data, where non-leaf nodes are labeled with the hash of the labels of their child nodes.

In the first (centralized) approach, each user table and the metadata tablet stores its root hash of the authenticated data structure at the metadata tablets and the root tablet, respectively; see Fig. 7(a). The root hash of the root tablet is stored at the user-end. Whenever any data is updated at a user table, the corresponding authenticated data structure is also updated. An update of the authenticated data structure at a user table requires updates of the corresponding authenticated data structures at the metadata tablet, the root tablet, and the user-end. Such updates of authenticated data structures decrease the performance of BigTable and require the involvement of the user, the metadata tablets, and the root tablet. In the second (distributed) approach, a user stores the root hash of each tablet and each user table, see Fig. 7(b); and storing the root hash at the user-end increases the performance of iBigTable. In addition, it is not required to store the root hash of the authenticated data structures at the higher level.

In order to verify the integrity of data, three rounds of communication between a client and the servers are required. The first round requires communication between the user and the root tablet; the second round requires communication between the user and the metadata tablet; and the third round requires communication between the user and the user table. In each round, a tablet server generates and sends a *verification object*, VO, which contains a set of hashes, for the data sent to the user. On receiving data and the corresponding VO, the user verifies the integrity of the received data. VOs from the root tablet and the metadata tablet allow accessing the metadata tablet and the user table, respectively.

HDFS-RSA and HDFS-pairing. In order to store data in a confidential manner in HDFS, two approaches based on hybrid encryption are suggested in [84]. The hybrid encryption approaches, see Fig. 8, use a block cipher and a stream cipher (see Chapter 3 of [69] for block ciphers and stream ciphers).

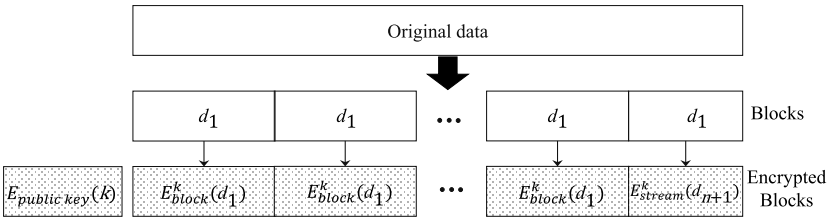


Fig. 8 – A hybrid encryption scheme used to store data in HDFS.

Data is divided into fixed-sized blocks, $d_1, d_2, \ldots, d_n$, according to a block cipher, and the remaining part of data becomes block d_{n+1}. The blocks $d_1, d_2, \ldots, d_n$ are encrypted using a random key k and a block cipher, and the block d_{n+1} is encrypted using the key k and a stream cipher. The key, k, is, then, encrypted using a public key scheme. The first approach, called *HDFS-RSA*, uses the RSA encryption scheme and AES, and the second approach, called *HDFS-pairing*, uses a pairing-based encryption scheme and AES. However, both the approaches are suitable for applications with a few write and many read operations.

SAPSC. Security Architecture of Private Storage Cloud based on HDFS (SAPSC) [68] provides an architecture for ensuring security of data stored in HDFS by *data isolation service*, *secure intra-cloud data migration service*, and *secure inter-cloud migration service*; see Fig. 9. These three services are dependent on five major services of a distributed files system, as follows: (i) fault-tolerant service, (ii) storage service, (iii) configuration of the system and management of the nodes, called node services, (iv) data transmission service, and (v) load balance service.

Data isolation services are invoked when a user reads/writes a file and are responsible for secure storage of data in HDFS and secure operations on data. Data isolation service is based on access policy management, access decision–authorization, access protocol security, and private data encryption. Secure intra-cloud data migration services do a secure data replication for the fault-tolerant service and transfer of data involved in a job, due to the load balance service or the node service. Secure inter-cloud data migration services do a secure data migration among clouds through the transmission service. In addition, three security policies are defined, as follows: (i) a flexible access control policy based on role-based access control, (ii) a label-based intra-cloud data replicating and restructuring policy, and (ii) a temporary-ticket based parallel inter-cloud data transmission policy.

3.4.4. *Approaches for result verification and accounting*

Several approaches for result verification are proposed based on redundancy of data and computations, trust management, and log analysis, which will be presented in this section.

- **Redundancy based approach**

A redundancy based approach replicates all or some of the tasks to multiple nodes and checks their outputs to find inconsistencies. Several approaches based on redundancy of tasks are reviewed below.

ClusterBFT. ClusterBFT [6] uses the Byzantine Failure Tolerant (BFT) [101] replication technique to cope with a situation where the cloud is trusted but there are potentially malicious nodes or users in a cluster. BFT replication is used for computational results verification and for overcoming untrusted, possibly malicious nodes. BFT replication techniques perform calculations in parallel on multiple replicas, then compare all the produced outputs to identify erratic behavioral nodes and decide a correct output based on a majority vote. However, current BFT replication techniques were developed for stand-alone servers and do not suit cloud-based computations, where data flow among different nodes and a computation consists of a number of stages to be performed on different nodes, as it is done in MapReduce. In order to overcome this gap, ClusterBFT algorithm adopts BFT replication for highly-scalable, distributed and high-granularity cloud computations. The algorithm identifies an optimal, according to heuristic function, subgraph of the computational flow that is verified by multiple replicas. The rest of the nodes participating in data flow are not replicated to avoid multiplication of verification messages and performance overhead. To reduce the volume of replicated data in verification phase, the algorithm uses digital digest of the data. In addition, the algorithm allows fault isolation and identification, *i.e.*, identification of components that continuously return incorrect outputs and removing them from the job scheduling.

SecureMR. A decentralized replication-based integrity verification framework, called SecureMR [64], ensures the integrity of data as well as computations, and prevents repudiation, DoS, and replay attacks. SecureMR replicates some map and reduce tasks, and assigns them to different mappers and reducers, *i.e.*, a map (or reduce) task is executed by more than one worker. The proposed framework consists of five security components, as follows: Secure Manager, Secure Scheduler, Secure Task Executor, Secure Committer, and Secure Verifier; see Fig. 10(a). The Secure Manager and the Secure Scheduler are deployed at the master process and perform task duplication, secure task assignment, and commitment-based consistency checking. The communication between the master process and mappers is carried out using the *commitment protocol*. The communication between the master process and reducers, and between mappers and reducers is done using the *verification protocol*.

Commitment protocol. The commitment protocol, see Fig. 10(b), avoids inspection of intermediate outputs by the master process and allows mappers to send a *commit message* with a signed hash value for each of its intermediate outputs (H_{P_i}) to the master process. Note that in the commitment protocol, the master process assigns a map task to a mapper in a secure manner using an encrypted message (signed, *sig*, by the master process and encrypted using the public key of the

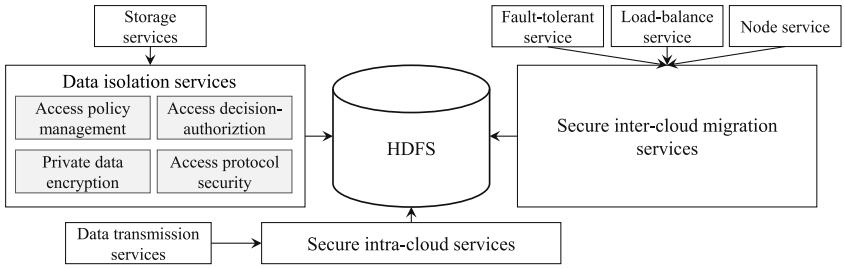
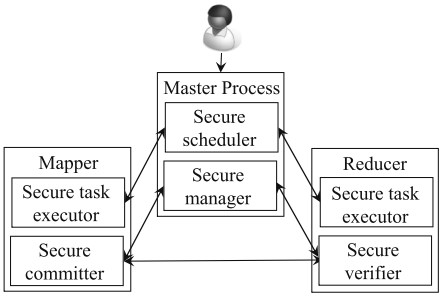
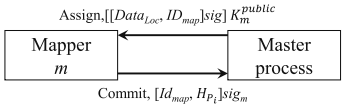


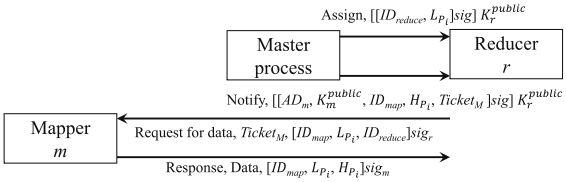
Fig. 9 – SAPSC architecture for data security.



(a) The framework.



(b) Communication between the master process and a mapper, the commitment protocol.



(c) Communication between the master process and a reducer, the verification protocol.

Fig. 10 – SecureMR framework, the commitment protocol, and the verification protocol.

mapper, K_m^{public}) that holds the location of data, $Data_{Loc}$, and identity of the map task, ID_{map}.

Verification protocol. In the *verification protocol*, see Fig. 10(c), reducers verify intermediate outputs and check the signed hash value that was submitted to the master process. The master process assigns a reduce task to a reducer in a secure manner using an encrypted message (encrypted using the public key of the reducer, K_r^{public}) that holds the location of intermediate outputs, L_{P_i}, and identity of the reduce task, ID_{reduce}. After that the master process sends a *notify message* to the verifier of each reducer, which includes the mapper's address (AD_M), K_m^{public}, ID_{Map}, H_{P_i}, and a ticket $Ticket_M$ (which includes K_r^{public}, ID_{Map}, L_{P_i}, and ID_{reduce}). The reducers ask intermediate outputs from the mapper, and the mapper sends data once it verifies the request. The verifier at the reducer verifies the response from the mapper, and in case of inconstancy, the verifier sends two signatures as evidences of an inconsistency to the master process.

In order to check the integrity of a MapReduce computation on a Desktop Grid [102,103], a replication based approach is suggested in [85], where reducers check results produced by mappers and the master process checks results produced by reducers. In addition, a MD5-based scheme is given to check outputs of mappers against a predefined digest code and Map function. The master process computes digest-codes for each split, and the digest-codes are sent to reducers. Mappers process input data splits, compute the code, and attach it with intermediate outputs. Reducers check the codes attached with intermediate outputs and the code received from the master process. If both the codes are different, then intermediate outputs are rejected.

Overhead issues. All the redundancy-based approaches replicate all or some of the tasks, and such a replication increases the communication cost and computation time of a MapReduce job. This becomes more important in non-free public clouds, where users are tariffed for the communication cost and computation time. An additional drawback of redundancy-based approaches is a difficulty to find a collusive malicious, as collusive attackers might provide an identical incorrect answer; thus bypassing the comparison-based verification.

• **Redundancy with trust based approaches**

Redundancy with trust based approaches use replication techniques for identification of inconsistencies, and then, use trusted workers for verification of results. Trusted workers might be located in a private cloud, in a different pool on the same cloud, or in any other trusted location. Also, it is assumed that the number of trusted workers is limited and does not allow execution of all tasks.

Accountable MapReduce. In order to ensure the integrity of MapReduce computations, Accountable MapReduce [70] verifies each map and reduce tasks by reexecuting them at a group of trusted workers. However, the reexecution of each task is not computationally efficient in detecting malicious mappers or reducers. Hence, the Accountable MapReduce also supports reexecution of some of the map and reduce tasks by a group of trusted workers.

VIAF. Verification-based Integrity Assurance Framework (VIAF) [73] builds trust between the master process and each of the mappers, based on replication of tasks and a quiz-based system. The VIAF framework introduces a new task called *verification task* that verifies outputs of the map tasks. The verification task, the reduce task, and the master process execute on a set of trusted workers; and the map task executes on untrusted workers. In the VIAF framework, each map task executes on two workers, and each worker accumulates a sufficient amount of credits by passing verification (quiz-based) process.

In the VIAF framework, the master process assigns each map task to two different workers; see step 1 in Fig. 11. After completion of the map task, both the workers return hash values of the results to the master process; see step 2 in Fig. 11. If the hash values are different, then the master process concludes that one of them is a malicious (non-collusive) worker, and hence, assigns the same task to two different workers. On the other hand, if both the hash values are identical, the master process stores the results of the two workers and the task information in their *history caches*; see step 3 in Fig. 11. After that, the master process may execute the verification task at the verifier in a non-deterministic manner (with a certain probability, called *verification probability*) to verify these consistent results of both the workers; see step 4 in Fig. 11. The master process concludes that the two workers are collusive workers when the verification task provides different results. On receiving an identical result from the verification task, both the workers pass one quiz and accumulate one credit; see step 5 in Fig. 11. When both the workers have an adequate and an identical amount of credits, they become trusted workers, and their results are passed to reducers through a *result buffer*; see steps 6 and 7 in Fig. 11.

CCMR. The framework, Cross Cloud MapReduce (CCMR) [86], extends the VIAF [73] framework by executing a MapReduce computation over a private cloud and a public cloud. The master process and the verification task are executed at the private cloud. In the map phase, each original map task is replicated with a replication probability, and each identical result is verified using the quiz-based approach with a verification probability. In the reduce phase, each reduce task is divided into multiple sub-tasks; and each sub-task is also replicated with a *replication probability*, and each identical result of sub-tasks is verified using the quiz-based approach with a verification probability.

IntegrityMR. IntegrityMR [74] extends the CCMR framework [86] by executing a MapReduce job on a private cloud and multiple public clouds, where the map and reduce tasks are executed on different public clouds. In addition, an invariant construction and checking method for applications written in Pig Latin [104] is also suggested, where an original script is transformed into another equivalent script. In the transformed script, a map task is substituted by two map tasks that work on some *overlapped* inputs and provide an identical result as an invariant of the map task after processing overlapped inputs. In order to achieve high accuracy, the reduce task executes on the private cloud, detects invariant violation, and restores results from the two map tasks, if they provide identical results.

VAWS. Verification-based Anti-collusive Worker Scheduling (VAWS) system [87] improves the scalability of the VIAF framework by removing the bottleneck from the verifier and by enhancing the ability of the verification frameworks to deal with collusive attacks. Majority vote based systems have an inherent flaw when malicious attackers comprise a majority of computational nodes for specific work items. This situation might occur even if the amount of collusive malicious attackers in the cluster is small in comparison to the entire cluster size. The VAWS system deals with both challenges by separating reducers and the master process into trusted domain under the assumption that the majority of MapReduce jobs are mappers. Mapper jobs are executed on two nodes and the results are compared. Nodes that agree on results are considered to be temporary consistent. The algorithm then builds and maintains a sub-graph of consistent nodes. Nodes that disagree in outputs in any execution are then paired with other nodes, both consistent and not consistent, in order to identify malicious nodes. While the system has a low overhead of verification and experiments successfully identify malicious nodes, it has a big disadvantage of allowing rounds of computations with incorrect results. This happens when both nodes that execute an identical job are malicious and continue to happen until the malicious nodes are correctly identified.

Hatman. Another framework based on replication of tasks, Hatman (HAdoop Trust MANager) [88], builds a trust level among different clouds, and NameNode keeps trust levels of DataNodes. A MapReduce job is distributed over kn workers, where k is a replication factor and n is the size of a group that process an assigned job independently. In other words, k non-identical groups of n workers in each group process a job independently. If the system does not contain any malicious worker, then all the k groups provide an identical result. On the other hand, when the master process receives different results, the master process chooses the results of trusted workers. Initially, all the nodes are assumed to be trusted and their relative confidence is also assumed to be uniform (*i.e.*, $\frac{1}{n}$). However, workers decrease their trust and relative confidence in case of compromises. The trust of a worker i towards a worker j is proportional to the percentage of jobs shared by i and j on which i's group agreed with j's

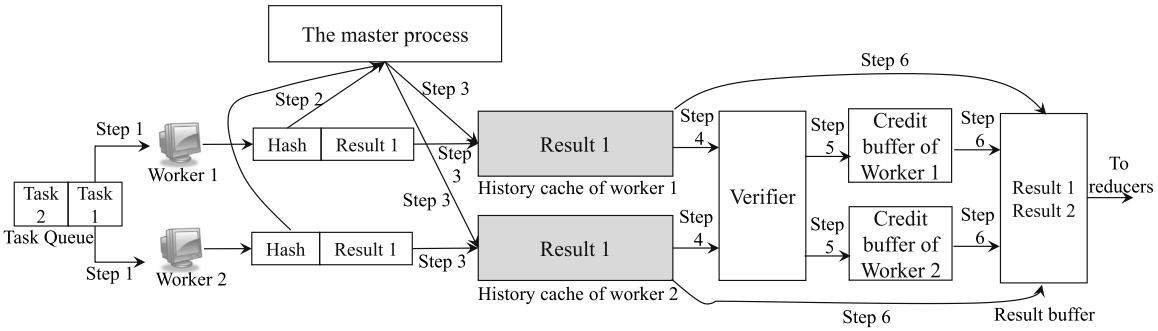


Fig. 11 – VIAF framework.

group, and a worker i relative confidence is the percentage of assessments of j that have been voiced by i.

TrustMR. In order to detect attacks with a high probability while minimizing the overhead, TrustMR [89] decomposes MapReduce tasks into smaller computations by means of aspect-oriented programming and replicates a subset of these task to verify the integrity of computations. TrustMR initiates multiple replicated map tasks on the replicated input splits. Some outputs of the map phase are randomly selected at runtime, and replicated map tasks only generate these key–value pairs. The results of replicated and original map tasks are verified at a map verifier by using a voting system. The results of replicated and original reduce tasks are also verified in the same manner at a reduce verifier.

TS-TRV. Trusted Sampling-based Third-party Result Verification (TS-TRV) [72] performs random sampling and constructs a Merkle tree at a trusted third-party, called *verifier*. The use of a Merkle tree reduces the amount of data that has to be sent to the verifier. A local Merkle tree is constructed by taking outputs of mappers as leaf nodes, when mappers finish the task. After that a global Merkle tree is constructed by the master process, where all root values submitted by mappers become leaf nodes, and the root value of the tree is sent to the verifier. The verifier randomly takes some number of (challenging) inputs from the global tree and sends them to corresponding mappers. On receiving (challenging) inputs, mappers construct responses in the form of a path to the root node of the local Merkle tree. The verifier also constructs the global Merkle tree using the responses sent by mappers and compares the values of the new root node against the old root node's value (to find a malicious mapper).

Overhead issues. Most of the approaches check the trust of the workers before the execution of a MapReduce computation, which causes an overhead in addition to replication overhead. For instance, a malicious worker can behave well for a long period of time to gain the trust of the master process and may attack only after that. Most of the above approaches do not deal well with such sophisticated attackers. In addition, even replication of some tasks together with usage of trusted workers still cannot guarantee the detection of all the malicious mappers and reducers.

- **Log analysis and watermarking-based approaches**

In order to find malicious workers and a malicious update in HDFS, an approach based on log analysis is suggested in [62]. The approach verifies the integrity and the correctness of a MapReduce job without modifying the original MapReduce job. Four types of logs are recorded, as follows: (i) logs of interaction between a user and NameNode and logs of interaction between a user and DataNode, (ii) logs of HDFS access, (iii) logs of interaction between HDFS and mappers–reducers, and (iv) logs of the Map and the Reduce phases. These logs are compared with some pre-defined systems and job invariants to find malicious workers.

Watermarking uses the concept of a watermark that is a kind of indistinguishable marker embedded in data. An approach based on watermark (or probe) injection is given in [90], which is able to detect malicious and lazy (a worker process that can either drop a task at any time or start a task not from beginning) workers. The approach consists of four steps: *watermark generation*, which generates some watermarks and inserts them into original data (before the start of a MapReduce job), *execution of a MapReduce job*, *verification*, which is done by the user by comparing outputs of all the processed watermarks by a MapReduce job and preprocessed outputs of all the watermarks, and *recovery*, which removes injected watermarks from outputs, once the output passes the consistency check at the verification step. The approach works well mostly for text-intensive tasks, *e.g.*, inverted index, word count, distributed grep, and log data processing, while being hard to apply to other types of input data. A method for verifying outputs of PageRank algorithm based on random sampling, called *in-degree weighted sampling*, is also suggested in [90], which proposes a way for verifying outputs of tasks where watermark injection is not possible.

Another watermark based approach is suggested in [91], which is also able to detect malicious workers. Accountable MapReduce [70] also supports watermarking-based detection of malicious workers. In Accountable MapReduce, some predefined watermarks are inserted into the original input data, and the outputs of the map phase, the inputs to the reduce phase, and the outputs of the reduce phase are verified.

A different approach was proposed and implemented in [92]. The paper suggested *purpose-based access control (PBAC)* (see [105,106] for more details) and implemented it over Hadoop MapReduce. The set of purposes is organized in hierarchical tree, where an edge between two purposes represents relations (specialization and generalization) between them. The system modifies MapReduce jobs and records to

include purposes. An access to specific data record is granted if the purposes specified by the security policy include or imply the purpose for the accessing of the data. It is assumed that when Hadoop system is deployed, the hierarchy of purposes is set in place and security policies are defined. A user submitting a job to Hadoop cluster then declares access policy, which is then checked by the AccountableMR system. The system clearly enhances the native security of Hadoop and allows much more fine-grained and sophisticated access control to the data. However, AccountableMR system also requires a considerable effort both in the initial setup of Hadoop cluster and from ongoing work of users.

Restrictive issues. Watermarking-based approaches are better than redundancy based approaches in terms of the workload on the framework, because these approaches do not re-execute all/some map and reduce tasks. However, watermarking approaches cannot be applied to all input data types or even to specific usages of the data. This considerably limits practical applications of such methods. In addition, the watermarking-based approaches do not guarantee finding all the malicious mappers and reducers, due to the fact that not all the data splits contain watermarks.

To summarize this section, there exist solutions to some of the security problems of MapReduce, however, more research is needed in order to provide effective solutions, which will be discussed briefly in Conclusions section (Section 5).

4. Privacy aspects in MapReduce

Privacy ensures that sensitive data is not exposed to untrusted users and trespassers (*i.e.*, cloud providers, other data providers, users of MapReduce, or adversaries). Notice that the data providers are interested in allowing some sorts of computations on the data, however, there is also a requirement to preserve breach of sensitive data. Sensitive data in this case is case specific and might be personal records with identifier information (personally identifiable information PII), organization specific information and etc. In this section, we present a brief summary of privacy aspects in general cloud computing, privacy requirements in MapReduce, and then, review some existing solutions for privacy in MapReduce.

4.1. *Privacy challenges in MapReduce computing*

Cloud computing and the deployment of MapReduce on public clouds present a new set of challenges in privacy of data. Here, we describe privacy challenges of cloud computing in the context of MapReduce and divide them into a few cases according to adversarial behaviors of public clouds and users.

Data privacy protection from adversarial cloud providers. A user may keep private data in public clouds due to its volume or ease of computations on public clouds, while aiming to preserve the privacy of data. In this setting, we consider an adversarial cloud provider that can observe users' data and MapReduce code; but should not change users' queries or results. Ensuring privacy in the presence of an adversarial cloud provider who can modify or delete data and computations is an insurmountable challenge. The goal of privacy in the presence of adversarial clouds is to minimize data leakage to the cloud provider while allowing users to perform operations on data. The majority of the work in this area is based on encryption of users' data and finding a way to allow operations on encrypted data in the cloud.

Protection of data from adversarial users. Data providers allow users to perform MapReduce jobs on their data via cloud providers, but also wish to control and preserve privacy of data. For example, while the average annual income can be calculated, the income of that specific individual should remain secret. Solutions to this use-case are based on anonymization of data (*i.e.*, by dropping sensitive values that can identify individuals), by adding random noise to data, or computational results to hide the real values, (*e.g.*, using differential privacy).

Multiusers on a single public cloud. A public cloud provider and a data provider should allow several users to perform their computations without data leakage. For example, an organization may keep all its data in public clouds, and several users process and access some parts of the data for which they are authorized. A hospital may store data of all the patients on clouds. There are several groups of users, *e.g.*, doctors, insurance companies, patients, and pharmaceutical research companies, which should access some parts of data. In this case, privacy framework has to ensure that each user is able to access all the required data but also that the users cannot access parts of data for which they are not authorized for. This is usually solved by authentication and authorization mechanisms as was explained in Section 3.4. The situation becomes more complex as data is provided by a number of data providers, each one with different privacy requirements. In such public clouds, an adversarial user may also access another user's data by injecting a malicious mapper or reducer that exploits existing security issues.

Various solutions are suggested for privacy of cloud computing. However, not all existing solutions for privacy of the cloud computing can be used for privacy in MapReduce, due to a number of additional constraints and challenges in MapReduce (presented in Section 2), thus requiring adaptation or change in those solutions.

4.2. *Privacy requirements in MapReduce*

MapReduce inherently decouples data providers, cloud providers, and users that execute queries over data. Referring to the cloud structure depicted in Fig. 4, data providers upload data to the cloud provider, and cloud users perform queries on data. However, despite separation between different entities, ensuring privacy in those settings is still a challenging task. Here, we provide requirements of privacy in MapReduce framework, deployed on the hybrid cloud or the public cloud.

Protection of data providers. In a setting where data is uploaded to the cloud by various data providers, each data provider might have a different privacy requirements. The cloud provider has to ensure that those privacy requirements are met even in the presence of adversarial users. Moreover, different data providers might require a different privacy level

for various datasets. The privacy framework should allow adaptation of privacy levels for those requirements.

Untrusted cloud providers. As an adversarial cloud provider can perform any computation on data for revealing data, modifying data, and producing wrong outputs, data has to be protected from cloud providers. In addition to protect the data from cloud providers, privacy framework has to be able to protect the performed computations as well. As an example, consider a user querying for specific information. Even if the data results are not released to the cloud provider, it is possible to learn the intent of the user from observing performed computations. In terms of MapReduce, it may require mappers and reducers to work on encrypted data (see Section 4.4).

Utilization and privacy tradeoff. A data provider can encrypt data in a way that no information can be learnt from it. However, this will also prevent the user from performing some computations on the data, and thus, decreases utilization of MapReduce. As such, MapReduce privacy framework has to provide maximum possible utilization while still preserving data privacy according to data providers' requirements.

Efficiency. In most of the public clouds, users are tariffed for usage and storage. Hence, the privacy framework has to be efficient in terms of CPU and memory consumption, and in the amount of storage required. If the privacy framework provides high overhead, it could be more cost-effective to perform computations on the private cloud, where physical security solves privacy issues.

4.3. *Adversarial models for MapReduce privacy*

All the adversarial models mentioned in Section 3.3 are applicable to MapReduce privacy with small changes. Below, we explain the adaptation required in the definitions of adversaries and how they can be applied in privacy settings.

Honest-but-curious adversary. This type of adversary mostly applies to cloud providers. Curious cloud providers can breach the privacy of data and MapReduce computations very easily, since the whole cluster is under the control of cloud providers, which have all types of privileged access to data and computing nodes. It is important to note that in reality curious cloud providers are not necessarily adversaries by choice, but rather might be compliant by court law, regulations, and governmental requests.[8]

Malicious adversary. This type of adversary applies to a user that tries to learn, modify, or delete information from the data by issuing various queries. In general, cloud providers are not assumed to be malicious, as assuring privacy with malicious cloud providers requires a high level of privacy measures that considerably reduce the utilization of the framework.

Knowledgeable adversary. A knowledgeable adversary applies to both a cloud provider and a user, who is trying to learn, modify, or delete information. Knowledgeable adversary is assumed to have a complete knowledge of MapReduce framework, the cloud structure, and is able to use any algorithm or cryptography drawback. In other words, there is no "security by obscurity".

Network and node adversary. As opposite to the network and nodes access adversary in security adversarial models, a cloud provider working as a network and node adversary has all the privileged access to computing nodes and the entire cloud infrastructure. A real-world example of such adversary is a cloud provider employee that breaches sensitive information most clearly shown by Edward Snowden case. It is impossible to hide any MapReduce computation or data from this type of adversary [107].

4.4. *Proposed solutions for privacy in MapReduce*

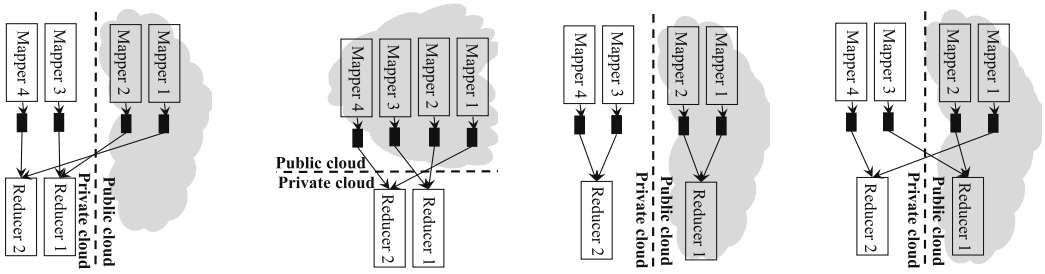
This section summarizes some existing solutions for privacy in MapReduce. We categorize privacy algorithms in MapReduce into three types, as follows: (i) algorithms for ensuring privacy in hybrid clouds, (ii) algorithms ensuring data privacy in the presence of adversarial users, and (iii) algorithms for ensuring privacy in the presence of adversarial cloud providers. A comparison of privacy algorithms, protocols, and frameworks for MapReduce is given in Table 2.

4.4.1. *Data privacy in hybrid clouds*

An increasing growth of data within organizations and lower maintenance costs are two factors that force data processing on public clouds instead of private clouds. Despite the change in the location of data processing, the need for privacy preservation of sensitive data remains identical. Thus, it is beneficial to process data based on sensitivity on the organization's private cloud and public clouds. Since MapReduce is designed for a single cloud, hybrid cloud based MapReduce computations require modification of MapReduce framework in order to deal with privacy (and security) issues on public clouds. In this section, we review some MapReduce privacy frameworks for the hybrid cloud computing model.

HybrEx. Hybrid Execution (HybrEx) [108] is the first MapReduce framework designed for the hybrid cloud. In HybrEx, data is divided into sensitive and non-sensitive data, non-sensitive data is sent to public clouds while sensitive data is kept in a private cloud. HybrEx allows four types of execution models of MapReduce computations, as follows: (i) Map hybrid: the map phase is executed at both public and private clouds, however, the reduce phase is executed at a private cloud only (Fig. 12(a)); (ii) Horizontal partitioning: the map phase is executed (on encrypted data) at public clouds only, while the reduce phase is executed at a private cloud (Fig. 12(b)); (iii) Vertical partitioning: the map phase and the reduce phase are executed on both public and private clouds while data transmission between private and public clouds is not allowed (Fig. 12(c)); and (iv) Hybrid: the map phase and the reduce phase are executed on both public and private clouds and data transmission among clouds is also possible (Fig. 12(d)). Two integrity check models, namely full integrity checking and quick integrity checking, are also suggested. However, HybridEx does not deal with a key that is generated at public and private clouds in the map phase.

[8] http://www.zdnet.com/article/microsoft-admits-patriot-act-can-access-eu-based-cloud-data/.



(a) Map hybrid. (b) Horizontal partitioning. (c) Vertical partitioning. (d) Hybrid.

Fig. 12 – Four execution models in HybrEx.

Table 2 – Summary of privacy algorithms, protocols, and frameworks for MapReduce.

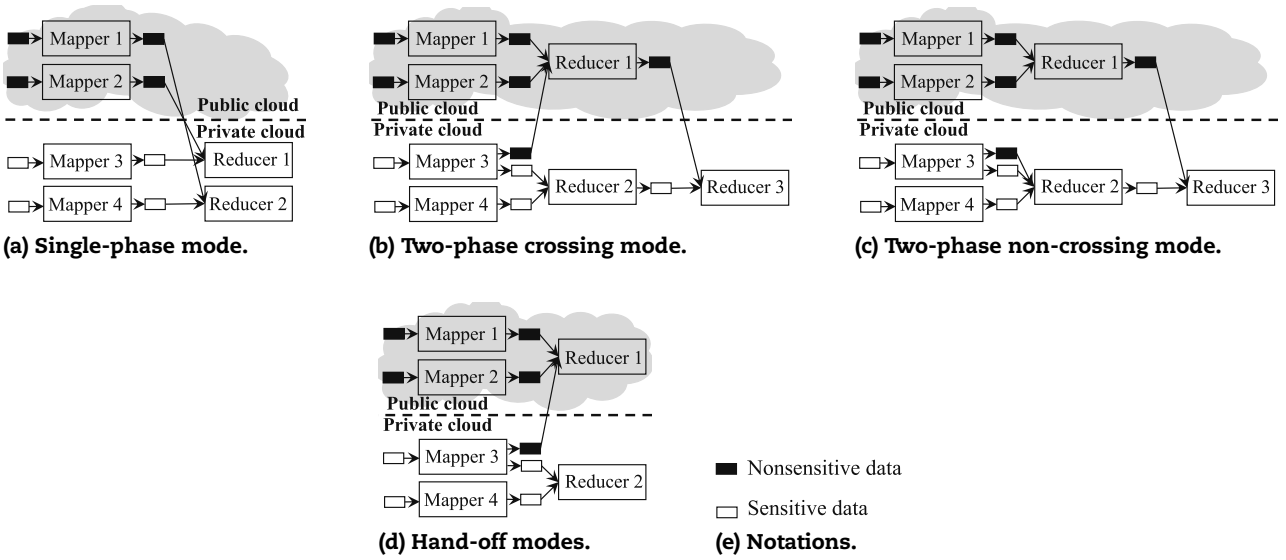
Algorithms/protocols/ frameworks	Privacy of data providers	Protection from adversarial		Approach	Cloud structure
		User	Cloud		
HybrEx [108]	✓		✓	Data separation	H[a]
Sedic [5]	✓		✓	Data separation	H
Tagged-MapReduce [109]	✓		✓	Data separation	H
SEMROD [110]	✓		✓	Data separation	H
Prometheus [111]	✓		✓	Data separation	H
PPL [112,113]		✓		Data anonymization	S[b]
Airavat [79]	✓	✓		Differential privacy	S
PRISM [114]	✓		✓	Encryption	S
PIRMAP [115]	✓		✓	Encryption and PIR schema	S
EPiC [116]	✓		✓	Homomorphic encryption	S
[117]		✓		Homomorphic Paillier encryption	S
PFC [118]		✓		FPGAs and proxy re-encryption	M[c]
CryptDB [119]	✓		✓	Variable homomorphic encryptions	S
MrCrypt [120]	✓		✓	Variable homomorphic encryptions	M
Crypsis [121]	✓		✓	Variable homomorphic encryptions	
[122]	✓	✓	✓	Secret-sharing	M

[a]H: Hybrid cloud.
[b]S: Single cloud.
[c]M: Multiple clouds.

Sedic. In order to solve key problem of HybridEx [108], Sedic [5] provides strategic data movement from the map phase that executes on public clouds to the reduce phase that executes on a private cloud, by using an automatic analysis and transformation of the reduce code. In order to decrease the communication between a public cloud and a private cloud, outputs of the map phase (at the public cloud) are aggregated before their transmission to the private cloud. In addition, Sedic framework automatically partitions a job by following security levels of data and distributes a job between private and public clouds.

Tagged-MapReduce. HybrEx [108] and Sedic [5] consider data sensitivity before a job's execution. Tagged-MapReduce [109] identifies data-sensitivity during execution of a job, where the map phase and the reduce phase are executed on public and private clouds. The framework handles sensitivity of intermediate outputs that may contain sensitive data, and hence, cannot be processed by the reduce phase at public clouds. Two policies, non-upgrading policy and downgrading policy, help in identifying on-the-fly data sensitivity, and four scheduling modes (single-phase, two-phase crossing, two-phase non-crossing, and hand-off modes), see Fig. 13, assign outputs of the map phase to reducers regarding data sensitivity. In addition, Tagged-MapReduce supports iterative MapReduce jobs. However, HybrEx [108], Sedic [5], and Tagged-MapReduce [109] are unable to handle the situation efficiently when a key is generated at public and private clouds. In order to solve this, Oktay et al. [110] suggested Secure and Efficient MapReduce Over hybriD clouds (SEMROD) that prevents the leakage of sensitive data and efficiently exploits public resources for executing a given single (or multi-level) MapReduce job.

SEMROD. SEMROD [110] first finds sensitive and non-sensitive data and sends non-sensitive data to public clouds. Private and public clouds execute the map phase. However, instead of sending only outputs of the map phase containing sensitive keys to the private cloud, the private cloud pulls all the outputs, but executes the reduce phase operation only on record associated with sensitive keys and ignores non-sensitive keys. Public clouds execute the reduce phase on all the outputs. Hence, they are unable to know the sensitive keys. At the end, a filtering step removes duplicate entries, creating by sensitive key.



(a) Single-phase mode. (b) Two-phase crossing mode. (c) Two-phase non-crossing mode.

(d) Hand-off modes. (e) Notations.

Fig. 13 – Four scheduling modes in Tagged-MapReduce.

Prometheus. In order to outsource non-sensitive data, which is stored in relations, to public clouds, Prometheus [111] removes quasi-identifiers (a quasi-identifier refers to a subset of attributes that can uniquely identify most tuples in a relation [123,124]) using a hypergraph. After the discovery of quasi-identifiers, attributes are distributed over public clouds, and an *attribute location table* is used to store name of relations and the location of relations-attributes. This allows the system to ensure that no sensitive data is stored in untrusted public clouds. It also avoids heavy workload on reducers at the user-end by sending merged outputs of public clouds and a *mapping table* of tuples from the private cloud to the user. Reducers (at the user-end) construct the final output. On the downside, Prometheus allows only search operations on a hybrid cloud.

A new framework for multiple clouds is proposed in [125]. The framework is divided into three layers: (i) *the physical layer*—holds computational resources; (ii) *the virtualization layer*—allows users to share the computational resources in a secure and isolated manner; and (iii) *the infrastructure-as-a-service (IaaS) layer*—manages and creates virtual resources, provides user management, and an access control method for accessing virtual resources. The IaaS layer has two sub-layers, namely *automatic deployment layer* that (i) creates virtual machines on the user-defined cloud, (ii) installs and configures Hadoop's master process based on an assigned job, (iii) executes a MapReduce job on the user-defined cloud, and *monitoring layer* that monitors all the virtual machines and resources. The framework allows processing of sensitive data at a private cloud and processing of non-sensitive data at a public cloud. The framework uses existing encryption, authentication, and access control methods. A secure data exchange is carried out using secure transport protocols. After a job completes, data is deleted immediately from the virtual resources, virtual machines are cleaned, and results are sent back to the user. Hence, the cloud does not hold data for a long time, which incurs users to send data every time to the cloud before computations.

Overhead issues. In most of the cases, apart from Sedic [5], the separation of sensitive and non-sensitive data is performed manually at a private cloud. Such a process drastically reduces the performance if datasets are huge. Moreover, a private cloud becomes a bottleneck if almost all the tuples contain sensitive data, which is processed on a private cloud.

4.4.2. *Data privacy with adversarial users*

In many applications, data providers and data users are different parties and might be completely separated. Examples of such applications are health service providers, pharmaceutical companies, and genomic data providers. In those cases, there is a clear need for providing data access to external parties for the purpose of research, monitoring, or knowledge sharing while providing sufficient data protection for data providers. However, for those purposes, the management of access controls or data encryption is not enough.

Data anonymization [126,127] is a promising solution for ensuring data privacy on public clouds. It works by hiding data identifiers, *i.e.*, attributes that allow identification of specific individuals, by changing information to some values, inserting records, and suppressing information [128–130].

Another notion of providing privacy, called *Differential Privacy* [131]. The idea of Differential Privacy is to ensure that an addition or removal of a single dataset item does not substantially affects the outcome of computations. In other words, adversary cannot distinguish between the results with and without a specific dataset item. The most common ways to achieve differential privacy is by addition of (specific) random noise to the sensitive data or computations, to hide an existence of any individual record. Differential Privacy is currently considered *de facto* standard of private data publishing as it provides rather strong privacy guarantee as it does not depend on auxiliary information known to adversary or computational power.

Additional privacy preserving methods for MapReduce are encryption–decryption-based solutions [114,115] and an accountability-based solution [70].

In [112,113], the authors presented a new framework for MapReduce computations based on data anonymization. The framework introduces a new layer, called *Privacy-Preserving Layer* (PPL, see Fig. 14), that exists between the original data and MapReduce framework for executing an assigned job. The PPL layer takes privacy requirements and original data as inputs. The layer then can apply different anonymization approaches according to the privacy requirements of data providers. This allows flexibility in choosing different privacy mechanisms within a single framework.

Specifically, the PPL layer consists of four main modules, as follows: (i) Privacy Specification Interface (PSI): takes several parameters as privacy specifications, (ii) Data Anonymizing (DA): does anonymization of data using MapReduce based anonymization algorithms [132] and privacy specifications, (iii) Data Update (DU): does anonymization of new data without anonymization of data from scratch, and (iv) Anonymized Datasets Management (ADM): provides methods for storing anonymized data in public clouds without breaching privacy of data.

Airavat [79] (see also Section 3.4.1) is the first system that combines mandatory access control (MAC) and differential privacy for ensuring data privacy according to differential privacy definition from untrusted users. Airavat allows users to submit MapReduce jobs with custom mappers and reducers chosen from a pre-defined set of trusted reducers. Mandatory access control ensures that untrusted mappers do not leak data outside the system via network or file system. Differential privacy requirements are ensured on intermediate outputs by adding a random noise to them. In order to maximize the utilization of the system and minimize the amount of added noise, the user has to provide a range of output values for every provided mapper. If the output exceeds the range, it is re-mapped to a random value in the range. Naturally, the system requires full trust in the cloud provider that implements the protocol and ensures the integrity of its components.

Utility issues. All data anonymization methods have an inherent tradeoff between utilization and privacy. Since data anonymization methods provide a high level of data privacy, the system utilization may decrease. Also, it may be hard to ensure anonymized data after performing operations on it. For example, joining of a relations having anonymized data with other relation having non-anonymized data may reveal data of the first relation.[9]

4.4.3. *Data privacy in adversarial clouds*

The most obvious solution for ensuring privacy of data in public clouds is encryption of data; however, it creates hurdles for an efficient utilization of MapReduce. In this section, we present some existing techniques that enable cloud users to perform MapReduce computations on encrypted data, while preserving privacy of data.

[9] http://privacyguidance.com/blog/10-big-data-analytics-privacy-problems/.

PRISM. Privacy-Preserving Search in MapReduce (PRISM) [114] alleviates the problem of storing data in curious cloud providers by allowing searching for any user specified word in privacy preserving manner, *i.e.*, the cloud provider should not be able to learn the user query and data. The proposed protocol consists of three phases, as follows: (i) upload of the data to the cloud, (ii) search operation, and (iii) result analysis phase. During the upload phase, the user encrypts data using state-full encryption algorithms, which add a frequency counters (as one of the possible options) to each word to prevent the cloud provider from computing statistics about frequency of encrypted text, and uploads the data to the cloud. In order to search the data, the user sends mappers and reducers based on Trapdoor Private Information Retrieval [133] for acquiring search results. Note that the cloud provider is considered honest-but-curious, and it will not change received mappers and reducers.

PIRMAP. Another system leveraging Private Information Retrieval (PIR), first defined in [134], is Private Information Retrieval for MapReduce (PIRMAP) [115]. PIRMAP is the first potentially practical cPIR algorithm (a cPIR algorithm is an algorithm that assumes that the cloud (or data) provider is polynomial-computational-bounded, as opposite to a generic case where the data provider is not bounded), which can be used in a real-world scenario. PIRMAP follows a "classical" PIR scheme (as defined [134] and improved in [135]) where the user sends an encrypted vector to the cloud provider. The cloud provider splits the data into blocks and multiplies each block by the received vector. Then, the cloud column-wise adds the results of the multiplication to create one-result vector. The vector is then returned to the user who decrypts it. These two stages of the algorithm, *i.e.*, multiplication and column-wise sum, are quite easily mapped into two stages of MapReduce, *i.e.*, map and reduce. Calculation of PIR scheme by MapReduce algorithm is done concurrently, according to the paradigm, and thus, allows great performance of otherwise computationally extensive scheme. The output of the mapper is a key–value pair, where the key is the index of the block and the value is a result of multiplication. Reducers then receive values of the column and perform the sum operation. Consequently, PIRMAP allows users to privately retrieve information from the cloud, using MapReduce.

EPiC. Efficient Privacy-Preserving Counting (EPiC) [116] protocol allows privacy-preserving counting using MapReduce and allows users to store their data in public clouds privately, *i.e.* protected from curious cloud providers. At the first phase, the user encrypts the data and uploads it to the cloud. The data is encrypted in such way that an identical data value does not generate an identical ciphertext, and hence, the cloud provider, which stores the (encrypted) data, cannot learn anything from the data apart from trivial characteristics, such as data size. At the query stage, the user specifies a searching pattern as a Boolean formula and generates mapper/reducer code for working on the encrypted data. The computation is performed by using partially homomorphic encryption for protecting outputs of the computation from the cloud provider. The cloud provider performs an assigned MapReduce computation and counts the total number of occurrences of an assigned pattern without learning neither the

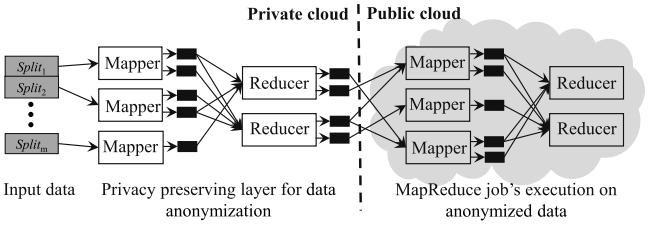


Fig. 14 – MapReduce framework with privacy-preserving layer.

data, the pattern, nor how often it occurs. EPiC is based on an idea of transforming the pattern search into a summation and polynomial evaluations, which can be done by partially homomorphic encryption scheme in an efficient manner. The protocol uses weaker encryption scheme for allowing more efficient execution of assigned queries. However, EPiC supports only counting operations, which is a limitation of the protocol.

A similar protocol was presented, in [117], for allowing privacy preserving implementation of the power iteration algorithm (a method for finding dominant eigenvectors for large matrices) on MapReduce. The protocol uses partially homomorphic Paillier encryption [136] scheme for algorithm computations. At the first stage of the protocol, the user encrypts the data using this encryption scheme and uploads the data to the public cloud. At processing stage, the user uses random vectors for protecting intermediate outputs and performs MapReduce computations by utilizing homomorphic properties of Paillier encryption scheme. The protocol is limited to computations of the specific algorithm only.

PFC. Field programmable gate arrays (FPGAs) and proxy re-encryption based privacy preserving solution for MapReduce computation is presented, in [118], where data is kept in an encrypted form in public clouds. An encryption algorithm is selected in a manner that data is easily partitioned into a number of splits, to be processed by mappers. However, mappers and reducers are not allowed to process encrypted splits and intermediate outputs, respectively. Mappers decrypt assigned splits before processing them and again encrypt intermediate outputs. The reducer also first decrypt intermediate outputs before processing and decrypt final outputs.

CryptDB. CryptDB [119] executes SQL queries over encrypted data providing practical confidentiality for the users. The idea of CryptDB is that most of the queries use well-defined set of operations, each of which is possible to support efficiently over encrypted data. CryptDB protects data from curious DBA that snoops on the database server and from a curious cloud provider that holds the servers and the data. The adversary does not change user queries. The tradeoff is between strong encryption, which will not allow many operations on the data, and between weaker encryption with more operations. Another tradeoff is minimizing the amount of leaked data when application servers are compromised. The authors do not see arbitrary computations on encrypted data as practical; thus, the application server has to be able to process decrypted data. (Their analysis over 128,840 queries from MIT applications showed that CryptDB can support 99.5% of all queries. It reduces throughput by 14.5% for full Web forums and by 26% for TPC-C queries comparing to unmodified MySQL.)

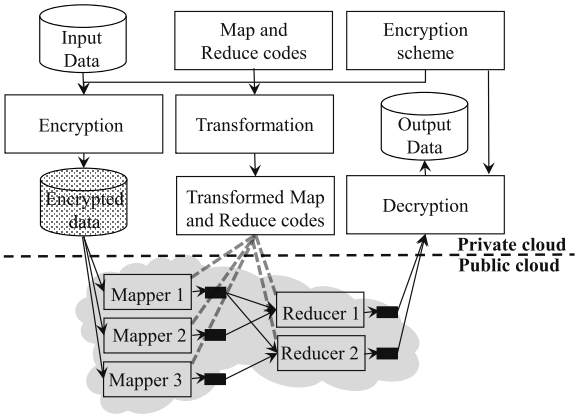


Fig. 15 – MrCrypt framework.

MrCrypt. Following the work of CryptDB [119], MrCrypt [120] suggests a way for executing MapReduce computations on encrypted data stored on curious cloud providers. MrCrypt's privacy preserving mechanism is based on two observations, as: (i) many MapReduce jobs perform only a limited set of basic operations on input data and (ii) homomorphic encryption schemas that enable specific operations are much more efficient than fully homomorphic encryption [137,138].

MrCrypt, see Fig. 15, performs static analysis of Java code for mapper and reducers at a private cloud. Following the analysis, a minimal homomorphic encryption scheme is chosen for supporting all the required operations in a legal and correct manner. The Java programs are then transformed using this encryption scheme, and data is also encrypted using the scheme. Next, the user uploads data and the transformed programs to a public cloud provider, which executes MapReduce job. The final outputs of the job are sent back to the user and are decrypted using the chosen homomorphic scheme. However, the downside of the approach is that it limits the range of possible queries on the system.

Crypsis. The ideas of CryptDB [119] and MrCrypt [120] were taken to higher data languages by Crypsis [121]. The system enables execution of Pig Latin jobs on a curious cloud provider without exposing data. Crypsis executes a MapReduce job on encrypted data without decrypting it. In order to do that, the system transforms a Pig Latin script so that it can be executed on encrypted data. Crypsis uses existing practical partially homomorphic encryption schemes for

data encryption. The system works in the following phases: (i) script transformation, Pig Latin script is analyzed and required encryption schemes are identified, the script is then changed to use encrypted data; (ii) update cloud with missing encryption schemes: it is possible that data stored in the cloud is missing some encryption schemes that are required for the given script, in that case those schemes are identified and the cloud is updated with newly encrypted data; (iii) execute encrypted script on the cloud infrastructure using pre-defined code provided by user stored with the data; (iv) re-encryption, it is possible that intermediate outputs are generated during the execution of the script, in such cases the data should be re-encrypted and (v) results, the results are sent to the user where they can be decrypted.

Overhead issues. The main obstacle of providing privacy-preserving framework for MapReduce cloud computations with a cloud provider acting as an adversary is computational and storage efficiency. The currently known fully homomorphic encryption schemes computational overhead is still prohibitively expensive [138]; thus, there is a need to find new schemes or methods of ensuring data privacy. The research papers reviewed in this section show that a considerable advance was made towards this goal. Nevertheless, all the above mentioned algorithms have common drawbacks such as: limited range of allowed queries (as a tradeoff between preserving data privacy and utilization), increased computation time, and in many cases an increased storage space for storing encrypted data. Despite those difficulties, the future of privacy preserving computations in public clouds looks promising and interesting.

4.4.4. *Data privacy in adversarial clouds using secret-sharing*

All the approaches suggested in Section 4.4.3 are based on encryption–decryption, which comes at a price of computation and limited operations [99]. In [122], a Shamir's secret-sharing [139] based solution for five types of MapReduce computations such as count, search, fetch, equijoin, and range selection is provided. The creation of secret-shares is computationally less expensive as compared to encryption–decryption techniques, and it provides information-theoretically secure computation. The suggested approach makes secret-shares of data and sends them to non-communicating clouds. A user can execute MapReduce computations of the form of secret-shares in those clouds and receives an answer of the form of secret-shares. By performing an interpolation technique, the user get the desired result. By using secret-sharing of data and computation, the cloud cannot learn the database and computation. Also, the user cannot learn the whole database. However, the use of more than one cloud increases costs.

5. Conclusions and future research directions

Processing a huge amount of data is not simple using the classical parallel computing, due to the failure of computing nodes and scalability of the system. MapReduce, developed by Google in 2004, provides an efficient, fault tolerant, scalable, and transparent processing of large-scale data. However, MapReduce was not designed to be deployed on public and hybrid clouds, where security and privacy of data and computations are two prime concerns. Since public clouds provide an easy way for computations and storage, a number of algorithms and frameworks regarding security and privacy of data-computations were developed for executing a MapReduce job on public and hybrid clouds.

In this survey, we discussed security and privacy challenges and requirements in MapReduce. Security attacks in MapReduce – impersonation, denial-of-services, replay, eavesdropping, man-in-the-middle, and repudiation attacks – are presented. We consider four types of adversarial models, namely honest-but-curious, malicious, knowledgeable, and network and nodes access adversaries, and show how they can impact a MapReduce computation. We reviewed many of the existing algorithms and frameworks for ensuring security and privacy in the scope of MapReduce.

Existing algorithms and frameworks succeed in solving the specific security and privacy problems in MapReduce. For example, data transmission and data storage are protected by encryption mechanisms; authentication and authorization solutions are based on existing secret key and integrated systems (such as SELinux); the result verification is done by replication of tasks; and privacy is ensured using data anonymization, differential privacy, and private information retrieval. Privacy preserving research is still struggling with providing a high utilization of MapReduce framework. While the reviewed papers show potentially practical solutions for specific problems, there is still considerable overhead (in terms of the workload on the framework) and limitations in utilization of the framework.

Based on this survey, we identified several important issues and challenges that require further research, as follows:

- Extending the authorization framework (security of MapReduce), *i.e.*, how to incorporate advanced authorization policies (*e.g.*, role-based or attribute-based access control policy) in MapReduce framework? This is particularly important if the mappers need to access different sources of data within the cluster.
- Integrating with a trust infrastructure (security of MapReduce). There are several domains of trust that must be made explicit and verified for MapReduce framework. These include: trust in the hardware, virtual machine, and file system that mappers and reducers use, trust that MapReduce code is not malicious or does not try to leak confidential data, and trust in the cloud provider for providing the necessary resources to run MapReduce algorithms.
- Processing on encrypted data (security and privacy of MapReduce). Although, as we have seen, some work has been done in this area, especially using homomorphic encryption, more research is needed in order to enable various MapReduce algorithms on encrypted data.
- Supporting multiple geographically distributed clusters for executing a single job (security and privacy of MapReduce). Often data and computing resources for a single job may exist in different independent clusters. For example, a bioinformatic application that tries to analyze genomes existing in different countries and labs in order to track the sources of a potential epidemic. How MapReduce can be

extended to multiple clusters, with support for privacy of the sensitive information across the clusters is an open research problem.
- Extending MapReduce algorithms with privacy preserving support (privacy of MapReduce). These include support for secure computations between reducers and across clusters. Also, privacy policies, which define exactly what kind of aggregated or anonymized data can be output, need to be defined.

Another lacking field of the research is holistic frameworks (with a salient exception of Airavat [79]), *i.e.*, frameworks that solve more than a single problem, especially solving both the security and privacy aspects, and integrating some of the mentioned algorithms and frameworks, which provide computational security and privacy of data for MapReduce computations. We believe that in the future we will have MapReduce frameworks that provide multiple types of computations in information secure manner.

Acknowledgments

Part of this research was supported by a grant from EMC Corp. during the development of world-wide-Hadoop. We thank the project coordinator Dr. Patricia Florissi for this support and very helpful comments.

Second author was partially supported by the Rita Altura Trust Chair in Computer Sciences, Lynne and William Frankel Center for Computer Sciences, Israel Science Foundation (grant 428/11), the Israeli Internet Association, and the Ministry of Science and Technology, Infrastructure Research in the Field of Advanced Computing and Cyber Security.

REFERENCES

[1] P. Mell, T. Grance, The NIST definition of cloud computing. 2011.
[2] R. Buyya, J. Broberg, A.M. Goscinski, Cloud computing: Principles and paradigms, Vol. 87, John Wiley & Sons, 2010.
[3] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges, J. Internet Serv. Appl. 1 (1) (2010) 7–18.
[4] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, in: 6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6–8, 2004, 2004, pp. 137–150.
[5] K. Zhang, X. Zhou, Y. Chen, X. Wang, Y. Ruan, Sedic: privacy-aware data intensive computing on hybrid clouds, in: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17–21, 2011, 2011, pp. 515–526.
[6] J.J. Stephen, P. Eugster, Assured cloud-based data analysis with ClusterBFT, in: Middleware 2013—ACM/IFIP/USENIX 14th International Middleware Conference, Beijing, China, December 9–13, 2013, Proceedings, 2013, pp. 82–102.
[7] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, R. Murthy, Hive—A warehousing solution over a Map-Reduce framework, Proc. VLDB 2 (2) (2009) 1626–1629.
[8] L. George, HBase: The Definitive Guide, O'Reilly Media, Inc., 2011.
[9] S. Sakr, A. Liu, A.G. Fayoumi, The family of MapReduce and large-scale data processing systems, ACM Comput. Surv. 46 (1) (2013) 11.
[10] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[11] G. Anthes, Security in the cloud, Commun. ACM 53 (11) (2010) 16–18.
[12] H. Takabi, J.B.D. Joshi, G. Ahn, Security and privacy challenges in cloud computing environments, IEEE Secur. Privacy 8 (6) (2010) 24–31.
[13] D. Zissis, D. Lekkas, Addressing cloud computing security issues, Future Gener. Comput. Syst. 28 (3) (2012) 583–592.
[14] Z. Xiao, Y. Xiao, Security and privacy in cloud computing, IEEE Commun. Surv. Tutor. 15 (2) (2013) 843–859.
[15] N.H.A. Rahman, K.R. Choo, A survey of information security incident handling in the cloud, Comput. Secur. 49 (2015) 45–69.
[16] M. Ali, S.U. Khan, A.V. Vasilakos, Security in cloud computing: Opportunities and challenges, Inf. Sci. 305 (2015) 357–383.
[17] J. Leskovec, A. Rajaraman, J.D. Ullman, Mining of Massive Datasets, Cambridge University Press, 2014.
[18] F.N. Afrati, S. Dolev, E. Korach, S. Sharma, J.D. Ullman, Assignment problems of different-sized inputs in MapReduce. CoRR, abs/1507.04461, 2015.
[19] F.N. Afrati, A.D. Sarma, S. Salihoglu, J.D. Ullman, Vision paper: Towards an understanding of the limits of map-reduce computation. CoRR, abs/1204.1754, 2012.
[20] R. Vernica, M.J. Carey, C. Li, Efficient parallel set-similarity joins using MapReduce, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6–10, 2010, 2010, pp. 495–506.
[21] C. Xiao, W. Wang, X. Lin, J.X. Yu, Efficient similarity joins for near duplicate detection, in: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21–25, 2008, 2008, pp. 131–140.
[22] F.N. Afrati, A.D. Sarma, D. Menestrina, A.G. Parameswaran, J.D. Ullman, Fuzzy joins using MapReduce, in: IEEE 28th International Conference on Data Engineering, ICDE 2012, Washington, DC, USA (Arlington, Virginia), 1–5 April, 2012, 2012, pp. 498–509.
[23] R.J. Bayardo, Y. Ma, R. Srikant, Scaling up all pairs similarity search, in: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8–12, 2007, 2007, pp. 131–140.
[24] G.S. Manku, A. Jain, A.D. Sarma, Detecting near-duplicates for web crawling, in: Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8–12, 2007, 2007, pp. 141–150.
[25] B. Chawda, H. Gupta, S. Negi, T.A. Faruquie, L.V. Subramaniam, M.K. Mohania, Processing interval joins on Map-Reduce, in: Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24–28, 2014., 2014, pp. 463–474.
[26] F.N. Afrati, S. Dolev, S. Sharma, J.D. Ullman, Bounds for overlapping interval join on MapReduce. in: Proceedings of the Workshops of the EDBT/ICDT 2015 Joint Conference, EDBT/ICDT, Brussels, Belgium, March 27th, 2015., 2015, pp. 3–6.
[27] H. Gupta, B. Chawda, S. Negi, T.A. Faruquie, L.V. Subramaniam, M.K. Mohania, Processing multi-way spatial joins on Map-Reduce, in: Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18–22, 2013, 2013, pp. 113–124.

[28] H. Gupta, B. Chawda, ϵ-controlled-replicate: An improvedcontrolled-replicate algorithm for multi-way spatial join processing on Map-Reduce, in: Web Information Systems Engineering—WISE 2014—15th International Conference, Thessaloniki, Greece, October 12–14, 2014, Proceedings, Part II, 2014, pp. 278–293.

[29] F. Tauheed, T. Heinis, A. Ailamaki, THERMAL-JOIN: A scalable spatial join for dynamic workloads, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31–June 4, 2015, 2015, pp. 939–950.

[30] F.N. Afrati, D. Fotakis, J.D. Ullman, Enumerating subgraph instances using Map-Reduce. in: 29th IEEE International Conference on Data Engineering, ICDE, Brisbane, Australia, April 8–12, 2013, 2013, pp. 62–73.

[31] P. Malhotra, P. Agarwal, G. Shroff, Graph-parallel entity resolution using LSH & IMM, in: Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference, EDBT/ICDT 2014, Athens, Greece, March 28, 2014, 2014, pp. 41–49.

[32] Y. Liu, X. Jiang, H. Chen, J. Ma, X. Zhang, MapReduce-based pattern finding algorithm applied in motif detection for prescription compatibility network, in: Advanced Parallel Processing Technologies, 8th International Symposium, APPT 2009, Rapperswil, Switzerland, August 24–25, 2009, Proceedings, 2009, pp. 341–355.

[33] A. Nandi, C. Yu, P. Bohannon, R. Ramakrishnan, Data cube materialization and mining over MapReduce, IEEE Trans. Knowl. Data Eng. 24 (10) (2012) 1747–1759.

[34] K. Rohitkumar, S. Patil, Data cube materialization using MapReduce, Int. J. Innov. Res. Comput. Commun. Eng. 11 (2) (2014) 6506–6511.

[35] B. Wang, H. Gui, M. Roantree, M.F. O'Connor, Data cube computational model with Hadoop MapReduce, in: WEBIST 2014—Proceedings of the 10th International Conference on Web Information Systems and Technologies, Volume 1, Barcelona, Spain, 3–5 April, 2014, 2014, pp. 193–199.

[36] F.N. Afrati, S. Sharma, J.D. Ullman, J.R. Ullman, Computing marginals using MapReduce. CoRR, abs/1509.08855, 2015.

[37] F.N. Afrati, P. Koutris, D. Suciu, J.D. Ullman, Parallel skyline queries. in: 15th International Conference on Database Theory, ICDT'12, Berlin, Germany, March 26–29, 2012, 2012, pp. 274–284.

[38] C. Zhang, F. Li, J. Jestes, Efficient parallel kNN joins for large data in MapReduce, in: 15th International Conference on Extending Database Technology, EDBT'12, Berlin, Germany, March 27–30, 2012, Proceedings, 2012, pp. 38–49.

[39] W. Lu, Y. Shen, S. Chen, B.C. Ooi, Efficient processing of k nearest neighbor joins using MapReduce, Proc. VLDB 5 (10) (2012) 1016–1027.

[40] G. Zhou, Y. Zhu, G. Wang, Cache conscious star-join in MapReduce environments, in: 2nd International Workshop on Cloud Intelligence (colocated with VLDB 2013), Cloud-I '13, Riva del Garda, Trento, Italy, August 26, 2013, 2013, pp. 1:1–1:7.

[41] A. Okcan, M. Riedewald, Processing theta-joins using MapReduce, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12–16, 2011, 2011, pp. 949–960.

[42] X. Zhang, L. Chen, M. Wang, Efficient multi-way theta-join processing using MapReduce, Proc. VLDB 5 (11) (2012) 1184–1195.

[43] Z. Yu, C. Wang, C.D. Thomborson, J. Wang, S. Lian, A.V. Vasilakos, Multimedia applications and security in MapReduce: Opportunities and challenges, Concurr. Comput.: Pract. Exper. 24 (17) (2012) 2083–2101.

[44] H.J. Karloff, S. Suri, S. Vassilvitskii, A model of computation for MapReduce, in: Proceedings of the Twenty-First Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17–19, 2010, 2010, pp. 938–948.

[45] M.T. Goodrich, Simulating parallel algorithms in the MapReduce framework with applications to parallel computational geometry. CoRR, abs/1004.4708, 2010.

[46] S. Lattanzi, B. Moseley, S. Suri, S. Vassilvitskii, Filtering: a method for solving graph problems in MapReduce, in: SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4–6, 2011, Co-located with FCRC 2011, 2011, pp. 85–94.

[47] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, E. Upfal, Space-round tradeoffs for MapReduce computations, in: International Conference on Supercomputing, ICS'12, Venice, Italy, June 25–29, 2012, 2012, pp. 235–244.

[48] A. Goel, K. Munagala, Complexity measures for MapReduce, and comparison to parallel computing. CoRR, abs/1211.6526, 2012.

[49] J.D. Ullman, Designing good MapReduce algorithms, ACM Crossroads 19 (1) (2012) 30–34.

[50] F.N. Afrati, A.D. Sarma, S. Salihoglu, J.D. Ullman, Upper and lower bounds on the cost of a Map-Reduce computation, Proc. VLDB 6 (4) (2013) 277–288.

[51] F.N. Afrati, J.D. Ullman, Matching bounds for the all-pairs MapReduce problem, in: 17th International Database Engineering & Applications Symposium, IDEAS'13, Barcelona, Spain—October 09–11, 2013, 2013, pp. 3–4.

[52] F.N. Afrati, S. Dolev, S. Sharma, J.D. Ullman, Meta-MapReduce: A technique for reducing communication in MapReduce computations. CoRR, abs/1508.01171, 2015.

[53] B. Fish, J. Kun, Á. D. Lelkes, L. Reyzin, G. Turán, On the computational complexity of mapreduce, in: Distributed Computing—29th International Symposium, DISC 2015, Tokyo, Japan, October 7–9, 2015, Proceedings, 2015, pp. 1–15.

[54] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop Distributed File System, in: 26th Symposium on Mass Storage Systems and Technologies, 2010, pp. 1–10.

[55] J. Lin, C. Dyer, Data-intensive text processing with MapReduce, Synth. Lect. Hum. Lang. Technol. 3 (1) (2010) 1–177.

[56] http://web.mit.edu/kerberos/.

[57] O. O'Malley, K. Zhang, S. Radia, R. Marti, C. Harrell, Hadoop security design. Tech. Rep, Yahoo, Inc., 2009.

[58] D. Das, O. O'Malley, S. Radia, K. Zhang, Adding security to Apache Hadoop. Hortonworks Technical Report 1, 2010.

[59] https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SecureMode.html.

[60] Q. Shen, L. Zhang, X. Yang, Y. Yang, Z. Wu, Y. Zhang, SecDM: Securing data migration between cloud storage systems, in: IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, DASC 2011, 12–14 December 2011, Sydney, Australia, 2011, pp. 636–641.

[61] A. Ruan, A. Martin, TMR: towards a trusted MapReduce infrastructure, in: Eighth IEEE World Congress on Services, SERVICES 2012, Honolulu, HI, USA, June 24–29, 2012, 2012, pp. 141–148.

[62] E. Yoon, A.C. Squicciarini, Toward detecting compromised MapReduce workers through log analysis, in: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26–29, 2014, 2014, pp. 41–50.

[63] W. Du, J. Jia, M. Mangal, M. Murugesan, Uncheatable grid computing, in: 24th International Conference on Distributed Computing Systems (ICDCS 2004), 24–26 March 2004, Hachioji, Tokyo, Japan, 2004, pp. 4–11.

[64] W. Wei, J. Du, T. Yu, X. Gu, SecureMR: A service integrity assurance framework for MapReduce, in: Twenty-Fifth Annual Computer Security Applications Conference, ACSAC 2009, Honolulu, Hawaii, 7–11 December 2009, 2009, pp. 73–82.

[65] J. Huang, D.M. Nicol, R.H. Campbell, Denial-of-service threat to Hadoop/YARN clusters with multi-tenancy, in: 2014 IEEE International Congress on Big Data, Anchorage, AK, USA, June 27–July 2, 2014, 2014, pp. 48–55.

[66] M. Pastore, M. Pastore, E. Dulaney, CompTIA Security+ Study Guide: Exam SY0-101, Wiley, 2006.

[67] Y. Desmedt, Relay attack, in: Encyclopedia of Cryptography and Security, second ed., 2011, p. 1042.

[68] Q. Shen, Y. Yang, Z. Wu, X. Yang, L. Zhang, X. Yu, Z. Lao, D. Wang, M. Long, SAPSC: security architecture of private storage cloud based on HDFS, in: 26th International Conference on Advanced Information Networking and Applications Workshops, WAINA 2012, Fukuoka, Japan, March 26–29, 2012, 2012, pp. 1292–1297.

[69] S. William, W. Stallings, Cryptography and Network Security, 4/E, Pearson Education, India, 2006.

[70] Z. Xiao, Y. Xiao, Achieving accountable MapReduce in cloud computing, Future Gener. Comput. Syst. 30 (2014) 1–13.

[71] W. Wei, T. Yu, R. Xue, ibigtable: practical data integrity for bigtable in public cloud, in: Third ACM Conference on Data and Application Security and Privacy, CODASPY'13, San Antonio, TX, USA, February 18–20, 2013, 2013, pp. 341–352.

[72] Y. Ding, H. Wang, P. Shi, H. Fu, C. Guo, M. Zhang, Trusted sampling-based result verification on mass data processing, in: Seventh IEEE International Symposium on Service-Oriented System Engineering, SOSE 2013, San Francisco, CA, USA, March 25–28, 2013, 2013, pp. 391–396.

[73] Y. Wang, J. Wei, VIAF: verification-based integrity assurance framework for MapReduce, in: IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA, 4–9 July, 2011, 2011, pp. 300–307.

[74] Y. Wang, J. Wei, M. Srivatsa, Y. Duan, W. Du, IntegrityMR: Integrity assurance framework for big data analytics and management applications, in: Proceedings of the 2013 IEEE International Conference on Big Data, 6–9 October 2013, Santa Clara, CA, USA, 2013, pp. 33–40.

[75] Apache Knox, available at: https://knox.apache.org/index.html.

[76] Apache Ranger, available at: http://ranger.incubator.apache.org/.

[77] Project Rhino, available at: https://github.com/intel-hadoop/project-rhino/.

[78] Apache Accumulo, available at: https://accumulo.apache.org/.

[79] I. Roy, S.T.V. Setty, A. Kilzer, V. Shmatikov, E. Witchel, Airavat: Security and privacy for MapReduce, in: Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28–30, 2010, San Jose, CA, USA, 2010, pp. 297–312.

[80] A. Khaled, M.F. Husain, L. Khan, K.W. Hamlen, B.M. Thuraisingham, A token-based access control system for RDF data in the clouds, in: Cloud Computing, Second International Conference, CloudCom 2010, November 30–December 3, 2010, Indianapolis, Indiana, USA, Proceedings, 2010, pp. 104–111.

[81] H. Ulusoy, M. Kantarcioglu, E. Pattuk, K.W. Hamlen, Vigiles: Fine-grained access control for MapReduce systems, in: 2014 IEEE International Congress on Big Data, Anchorage, AK, USA, June 27–July 2, 2014, 2014, pp. 40–47.

[82] H. Ulusoy, P. Colombo, E. Ferrari, M. Kantarcioglu, E. Pattuk, GuardMR: Fine-grained security policy enforcement for MapReduce systems, in: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS'15, Singapore, April 14–17, 2015, 2015, pp. 285–296.

[83] J. Zhao, L. Wang, J. Tao, J. Chen, W. Sun, R. Ranjan, J. Kolodziej, A. Streit, D. Georgakopoulos, A security framework in G-Hadoop for big data computing across distributed cloud data centres, J. Comput. System Sci. 80 (5) (2014) 994–1007.

[84] H. Lin, S. Shen, W. Tzeng, B.P. Lin, Toward data confidentiality via integrating hybrid encryption schemes and Hadoop distributed file system, in: IEEE 26th International Conference on Advanced Information Networking and Applications, AINA, 2012, Fukuoka, Japan, March 26–29, 2012, 2012, pp. 740–747.

[85] M. Moca, G.C. Silaghi, G. Fedak, Distributed results checking for MapReduce in volunteer computing, in: 25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011, Anchorage, Alaska, USA, 16–20 May 2011—Workshop Proceedings, 2011, pp. 1847–1854.

[86] Y. Wang, J. Wei, M. Srivatsa, Result integrity check for MapReduce computation on hybrid clouds, in: 6th International Conference on Cloud Computing, 2013, pp. 847–854.

[87] Y. Ding, H. Wang, L. Wei, S. Chen, H. Fu, X. Xu, VAWS: Constructing trusted open computing system of MapReduce with verified participants, IEICE Trans. 97 (D(4)) (2014) 721–732.

[88] S.M. Khan, K.W. Hamlen, Hatman: Intra-cloud trust management for Hadoop, in: 2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, June 24–29, 2012, 2012, pp. 494–501.

[89] H. Ulusoy, M. Kantarcioglu, E. Pattuk, TrustMR: Computation integrity assurance system for MapReduce, in: 2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29–November 1, 2015, 2015, pp. 441–450.

[90] C. Huang, S. Zhu, D. Wu, Towards trusted services: Result verification schemes for MapReduce, in: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012, Ottawa, Canada, May 13–16, 2012, 2012, pp. 41–48.

[91] Y. Ding, H. Wang, S. Chen, X. Tang, H. Fu, P. Shi, PIIM: method of identifying malicious workers in the MapReduce system with an open environment, in: 8th IEEE International Symposium on Service Oriented System Engineering, SOSE 2014, Oxford, United Kingdom, April 7–11, 2014, 2014, pp. 326–331.

[92] H. Ulusoy, M. Kantarcioglu, E. Pattuk, L. Kagal, AccountableMR: Toward accountable MapReduce systems, in: 2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29–November 1, 2015, 2015, pp. 451–460.

[93] Apache Sentry, available at: http://sentry.incubator.apache.org/ and https://blogs.apache.org/sentry/.

[94] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, R.E. Gruber, Bigtable: A distributed storage system for structured data, ACM Trans. Comput. Syst. 26 (2) (2008).

[95] O. Lassila, R.R. Swick, Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, Feb 1999. Available at: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[96] Z. Kaoudi, I. Manolescu, RDF in the clouds: a survey, VLDB J. 24 (1) (2015) 67–91.

[97] G. Carothers, A. Seabourne, RDF 1.1 N-Triples. W3C Recommendation, Feb 2014. Available at: http://www.w3.org/TR/2014/REC-n-triples-20140225/.
[98] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, D. Chen, G-Hadoop: MapReduce across distributed data centers for data-intensive computing, Future Gener. Comput. Syst. 29 (3) (2013) 739–750.
[99] H. Ulusoy, M. Kantarcioglu, B.M. Thuraisingham, L. Khan, Honeypot based unauthorized data access detection in MapReduce systems, in: 2015 IEEE International Conference on Intelligence and Security Informatics, ISI 2015, Baltimore, MD, USA, May 27–29, 2015, 2015, pp. 126–131.
[100] R.C. Merkle, A digital signature based on a conventional encryption function, in: Advances in Cryptology—CRYPTO'87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16–20, 1987, Proceedings, 1987, pp. 369–378.
[101] L. Lamport, R.E. Shostak, M.C. Pease, The byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401.
[102] G. Fedak, C. Germain, V. Néri, F. Cappello, Xtremweb: A generic global computing system, in: First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15–18, 2001, Brisbane, Australia, 2001, pp. 582–587.
[103] D.P. Anderson, BOINC: A system for public-resource computing and storage, in: 5th International Workshop on Grid Computing, GRID 2004, 8 November 2004, Pittsburgh, PA, USA, Proceedings, 2004, pp. 4–10.
[104] C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins, Pig latin: a not-so-foreign language for data processing, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10–12, 2008, 2008, pp. 1099–1110.
[105] J.-W. Byun, E. Bertino, N. Li, Purpose based access control of complex data for privacy protection, in: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies, SACMAT '05, ACM, New York, NY, USA, 2005, pp. 102–110.
[106] J.-W. Byun, N. Li, Purpose based access control for privacy protection in relational database systems, VLDB J. 17 (4) (2006) 603–619.
[107] M.R. Randazzo, M. Keeney, E. Kowalski, D. Cappelli, A. Moore, Insider threat study: Illicit cyber activity in the banking and finance sector, 2005.
[108] S.Y. Ko, K. Jeon, R. Morales, The HybrEx model for confidentiality and privacy in cloud computing, in: 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'11, Portland, OR, USA, June 14–15, 2011, 2011.
[109] C. Zhang, E. Chang, R.H.C. Yap, Tagged-MapReduce: A general framework for secure computing with mixed-sensitivity data on hybrid clouds, in: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, May 26–29, 2014, 2014, pp. 31–40.
[110] K.Y. Oktay, S. Mehrotra, V. Khadilkar, M. Kantarcioglu, SEMROD: secure and efficient MapReduce over hybrid clouds, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31–June 4, 2015, 2015, pp. 153–166.
[111] Z. Zhou, H. Zhang, X. Du, P. Li, X. Yu, Prometheus: Privacy-aware data retrieval on hybrid cloud, in: Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14–19, 2013, 2013, pp. 2643–2651.
[112] X. Zhang, C. Liu, S. Nepal, W. Dou, J. Chen, Privacy-preserving layer over MapReduce on cloud, in: 2012 Second International Conference on Cloud and Green Computing, CGC 2012, Xiangtan, Hunan, China, November 1–3, 2012, 2012, pp. 304–310.
[113] X. Zhang, C. Liu, S. Nepal, C. Yang, J. Chen, Privacy preservation over big data in cloud systems, in: Security, Privacy and Trust in Cloud Systems, 2014, pp. 239–257.
[114] E. Blass, R.D. Pietro, R. Molva, M. Önen, PRISM—privacy-preserving search in MapReduce, in: Privacy Enhancing Technologies—12th International Symposium, PETS 2012, Vigo, Spain, July 11–13, 2012. Proceedings, 2012, pp. 180–200.
[115] T. Mayberry, E. Blass, A.H. Chan, PIRMAP: efficient private information retrieval for mapreduce, in: Financial Cryptography and Data Security—17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers, 2013, pp. 371–385.
[116] E. Blass, G. Noubir, T.V. Huu, EPiC: Efficient privacy-preserving counting for MapReduce, 2012.
[117] J. Powers, K. Chen, Secure MapReduce power iteration in the cloud. CoRR, abs/1211.3147, 2012.
[118] L. Xu, W. Shi, T. Suh, PFC: privacy preserving FPGA cloud—A case study of MapReduce, in: 2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, June 27–July 2, 2014, 2014, pp. 280–287.
[119] R.A. Popa, C.M.S. Redfield, N. Zeldovich, H. Balakrishnan, CryptDB: processing queries on an encrypted database, Commun. ACM 55 (9) (2012) 103–111.
[120] S.D. Tetali, M. Lesani, R. Majumdar, T.D. Millstein, MrCrypt: static analysis for secure cloud computations, in: Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26–31, 2013, 2013, pp. 271–286.
[121] J.J. Stephen, S. Savvides, R. Seidel, P. Eugster, Practical confidentiality preserving big data analysis, in: 6th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud '14, Philadelphia, PA, USA, June 17–18, 2014., 2014.
[122] S. Dolev, Y. Li, S. Sharma, Private and secure secret shared MapReduce—(extended abstract), in: Data and Applications Security and Privacy XXX—30th Annual IFIP WG 11.3 Working Conference, DBSec 2016, Trento, Italy, July 18–21, 2016. Proceedings, 2016.
[123] R. Motwani, Y. Xu, Efficient algorithms for masking and finding quasi-identifiers. 2007.
[124] J. Pei, Y. Tao, J. Li, X. Xiao, Privacy preserving publishing on multiple quasi-identifiers, in: Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009–April 2 2009, Shanghai, China, 2009, pp. 1132–1135.
[125] S. Loughran, J.M.A. Calero, A. Farrell, J. Kirschnick, J. Guijarro, Dynamic cloud deployment of a MapReduce architecture, IEEE Internet Comput. 16 (6) (2012) 40–50.
[126] B. Zhou, J. Pei, W. Luk, A brief survey on anonymization techniques for privacy preserving publishing of social network data, SIGKDD Explor. 10 (2) (2008) 12–22.
[127] B.C.M. Fung, K. Wang, R. Chen, P.S. Yu, Privacy-preserving data publishing: A survey of recent developments, ACM Comput. Surv. 42 (4) (2010).
[128] N.R. Adam, J.C. Wortmann, Security-control methods for statistical databases: A comparative study, ACM Comput. Surv. 21 (4) (1989) 515–556.
[129] F.M. Malvestuto, M. Moscarini, M. Rafanelli, Suppressing marginal cells to protect sensitive information in a two-dimensional statistical table, in: Proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 29–31, 1991, Denver, Colorado, USA, 1991, pp. 252–258.

[130] P. Chu, Cell suppression methodology: The importance of suppressing marginal totals, IEEE Trans. Knowl. Data Eng. 9 (4) (1997) 513–523.
[131] C. Dwork, Differential privacy, in: Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10–14, 2006, Proceedings, Part II, 2006, pp. 1–12.
[132] X. Zhang, C. Yang, S. Nepal, C. Liu, W. Dou, J. Chen, A MapReduce based approach of scalable multidimensional anonymization for big data privacy preservation on cloud, in: 2013 International Conference on Cloud and Green Computing, Karlsruhe, Germany, September 30–October 2, 2013, 2013, pp. 105–112.
[133] J.T. Trostle, A. Parrish, Efficient computationally private information retrieval from anonymity or trapdoor groups, in: Information Security—13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25–28, 2010, Revised Selected Papers, 2010, pp. 114–128.
[134] B. Chor, E. Kushilevitz, O. Goldreich, M. Sudan, Private Information Retrieval, Vol. 45, ACM, 1998, pp. 965–981.
[135] H. Lipmaa, An oblivious transfer protocol with log-squared communication, in: Information Security, 8th International Conference, ISC 2005, Singapore, September 20–23, 2005, Proceedings, pp. 314–328, 2005.
[136] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: Advances in Cryptology—EUROCRYPT'99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2–6, 1999, Proceeding, 1999, pp. 223–238.
[137] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31–June 2, 2009, 2009, pp. 169–178.
[138] C. Gentry, S. Halevi, Implementing gentry's fully-homomorphic encryption scheme, in: Advances in Cryptology—EUROCRYPT 2011—30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15–19, 2011. Proceedings, 2011, pp. 129–148.
[139] A. Shamir, How to share a secret, Commun. ACM 22 (11) (1979) 612–613.