2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Tagged-MapReduce: A General Framework for Secure Computing with Mixed-Sensitivity Data on Hybrid Clouds

Chunwang Zhang[1], Ee-Chien Chang, Roland H.C. Yap
School of Computing, National University of Singapore
{*chunwang, changec, ryap*}*@comp.nus.edu.sg*

***Abstract*—This paper presents tagged-MapReduce, a general extension to MapReduce that supports secure computing with mixed-sensitivity data on hybrid clouds. Tagged-MapReduce augments each key-value pair in MapReduce with a sensitivity tag. This enables fine-grained dataflow control during execution to prevent data leakage as well as supporting expressive security policies and complex MapReduce computations. Security constraints for preventing data leakage impose restrictions on computation and data storage/transfer, hence, we present scheduling strategies that can exploit properties of the map and reduce functions to rearrange the computation for greater efficiency under these constraints while maintaining MapReduce correctness. We present a general security framework for analyzing MapReduce computations in the hybrid cloud which captures how dataflow can leak information through execution. Experiments on Amazon EC2 with our prototype in Hadoop show that we are able to obtain security while effectively outsourcing computation to the public cloud and reducing inter-cloud communication.**

***Keywords*-Data security; MapReduce; hybrid clouds; information leakage**

I. Introduction

Cloud computing promises to provide scalable and on-demand compute resources for processing large data. Rather than simply using a public cloud (e.g., Amazon EC2), an enterprise could use a hybrid cloud consisting of a private cloud (e.g., enterprise's in-house servers) together with a public cloud. A seamless combination of these two clouds offers increased scalability. The private cloud can be used for typical workloads which fit within the local resources, but when additional resources are needed during peak computation, the public cloud is harnessed. This hybrid cloud model has gained adoptions and is still undergoing rapid development [1].

However, hybrid cloud computing needs to address the security and privacy issues with public clouds. Security and privacy are ranked as the top concerns for organizations considering moving their data and applications to the cloud [2]. There are good reasons for these concerns, e.g., Ristenpart et al. [17] demonstrate that confidential information can be extracted through side-channel information leakage in VMs. Data breach incidents have also been reported over the years for various cloud service providers, e.g. [3]. On the other hand, an organization's data often involves both sensitive and non-sensitive information, e.g., the organization's filesystem may contain general (non-sensitive) files mixed with confidential business data. Also, many datasets for analytical tasks such as network logs, email archives and healthcare records may involve data from public sources mixed with sensitive private data. Computation on such mixed-sensitivity data should not be carried out on the public cloud without security measures to prevent data leakage. Cryptographic techniques such as fully homomorphic encryption [12] enable computations to be carried out on the encrypted domain on the public cloud but are still far from efficient for large data.

One solution for secure computation in a hybrid cloud is to separate the computation on non-sensitive data from that on sensitive data, such that the former can be comfortably outsourced to the public cloud while the latter, possibly much smaller in size, can be easily handled on the private cloud. In this way, the computation can be carried out both securely while being elastic. However, this hybrid computing model is not well supported by MapReduce [10] (MR), the most popular data-intensive computing framework. MR provides a seamless distribution of computing tasks among nodes in the cloud in a way which is transparent to the programmers/users. MR is designed for only one (logical) cloud and does not distinguish between data and servers with differing sensitivities. Hence, cloud users who want to run MR jobs with mixed-sensitivity data on a hybrid cloud need to manually split the data, compute each partition on one corresponding cloud separately and combine the results in their own code. This is neither transparent nor efficient. Our objective is an automatic and general framework to facilitate secure MR computation on hybrid clouds.

Sedic [21] addresses this problem to some degree by pre-labeling the input data which is then replicated to both the public and private clouds, but with sensitive portions in the public cloud "sanitized". During computation, map tasks operate in both clouds and send all intermediate results to the private cloud for reducing to prevent data leakage from

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Competitive Research Programme (CRP Award No. NRF-CRP8-2011-08).

[1]Zhang is partially supported by the research grant R-252-000-514-112.

978-1-4799-2784-5/14 $31.00 © 2014 IEEE
DOI 10.1109/CCGrid.2014.96

the intermediate results. However, the sanitization approach taken by Sedic has limitations in terms of flexibility - it does not fit well with complex MR jobs such as chained or iterative MR, which is important to many data analytical tasks and realistic applications [16]. In addition, the sanitization approach may still reveal relative locations and length of sensitive data, which could lead to crucial information leakage in certain applications [14]. Hence, a more generic, flexible and secure framework is desired.

In this paper, we propose a conservative extension to MR that deals automatically with mixed-sensitivity data in hybrid clouds and supports a new MR programming model where data sensitivity can be manipulated during computation, e.g., security-aware programs can be used to downgrade the sensitivity of data in execution. We propose tagged-MapReduce that (conceptually) augments each key-value pair in MR with a sensitivity tag [20], extending the map and reduce functions appropriately. The tagging helps to achieve the following goals: 1) it enables fine-grained dataflow control during execution to prevent leakage and supports scheduling of map and reduce tasks in the two clouds; 2) it allows programmers to code sophisticated security policies to guide sensitivity transformation during execution and supports sensitivity downgrading which is useful in sensitivity-aware applications; 3) it provides sensitivity information for data across multiple MR jobs which is necessary for complex MR computations with chained jobs. The flexibility also allows legacy MR programs to be supported by simply having a default tagging policy. Sedic programs can be expressed as a special class of tagged-MR programs, however, Sedic cannot express all tagged-MR programs.

The concerns of preventing data leakage mean that there is a security constraint on where computations are run and where data is sent in a hybrid cloud computing job. We provide scheduling strategies for reduce tasks so that some reducers can execute in the public cloud. The scheduling strategies exploit useful properties of common map and reduce functions to rearrange the computation for more effective load-balancing and inter-cloud network usage while maintaining MR correctness. For example, if a reduce operation is "partitionable", tagged-MR will automatically carry out partial reduce computation on the public cloud (with non-sensitive data), which lessens not only the private cloud's workload but also the total amount of inter-cloud data traffic. Our prototype implementation allows the properties met to be easily coded into the tagged-MR programs, from which the scheduler decides automatically which scheduling strategy is to be employed.

Nevertheless, special care must be taken in designing such scheduling strategies as different strategies lead to different actual dataflows during execution, which in turn leads to different amounts and types of information being exposed to the public cloud. In particular, a scheduler that aggressively rearranges the computation to the public cloud, while improving efficiency and maintaining MR correctness, may leak more information than a "conservative" scheduler that carries out all reduce computation in the private cloud. Such leakage is beyond the programmers' anticipation and could be unacceptable in some scenarios. To analyze the scheduling strategies, we propose the first security model that captures how dataflow can leak information during execution. This model is suitable for analyzing what additional leakage a scheduler might make through execution on a hybrid cloud over a reference "baseline" scheduler whose information exposure is deemed to be acceptable. Using this model, we are able to show that some potentially more effective scheduling strategies indeed leak more information than the baseline, whereas, ours do not.

We implement a prototype of tagged-MR on Hadoop with experiments to evaluate the feasibility of the system and the effectiveness of the proposed scheduling strategies. The experiments are run on a small-sized hybrid cloud we built on Amazon EC2, using both single and chained MR jobs. The results show that the tagged-MR prototype which implements the security constraints for preventing data leakage is able to automatically and efficiently outsource computation to the public cloud and reduce inter-cloud data traffic. The system is practical with only small overheads compared to baseline Hadoop which ignores the data confidentiality and security constraints.

II. Overview

A. MapReduce

MapReduce [10] (MR) is a popular framework for performing distributed computation over large data sets. In MR, data are represented as *key-value* pairs (or *tuples*). Users provide a *map* and a *reduce* function. A map function takes as input a key-value pair and produces a set of intermediate key-value pairs, while a reduce function aggregates all intermediate values associated with a same key. The execution of map and reduce *tasks* are automatically distributed across all the nodes in the cluster, and the data movement between map and reduce is known as *shuffling*. MR has mechanisms to ensure scalability and fault tolerance by replicating data multiple times, dynamically scheduling tasks, handling failures, etc. Thus, it is easy to write programs working on large clusters, with no experience in parallel/distributed systems. A popular open-source implementation of MR is Hadoop, which offers a distributed file system (HDFS) and an execution framework for managing MR jobs and tasks.

B. Overview of the Proposed System

A hybrid cloud seamlessly integrates an enterprise's private data center with public cloud resources. The idea is to use the elasticity of the public cloud to handle peak loads which are beyond one's private cloud resources. While the private cloud is trusted, the public cloud may be vulnerable

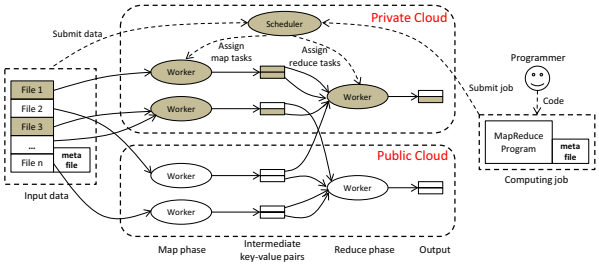


Figure 1. Overview of tagged-MapReduce from the perspective of users and programmers. Shaded rectangles are files/tuples marked as sensitive, shaded ellipses are workers/scheduler run in the private cloud. Note that the output tuples carry sensitivity information which can be fed to the next job, thus multiple MapReduce jobs can be naturally supported.

to various data leakages [17], hence, one should ensure that using the hybrid cloud does not leak sensitive information.

MR is not designed for processing a mix of sensitive and non-sensitive data in the hybrid cloud, as data can flow freely between all nodes in the two clouds, increasing the risk of information leakage. To prevent such leakage, we propose to extend MR by explicitly tagging each key-value pair as either sensitive or non-sensitive. The tags serve as auxiliary information for the system to move data during execution, ensuring that sensitive tuples never leave the private cloud. We propose *tagged-MapReduce*, as shown in Fig. 1, which involves: (1) a *scheduler* in the private cloud that schedules map and reduce tasks to workers and controls the flow of intermediate data w.r.t. their security tags, and (2) multiple *workers* across the public and private clouds that carry out the assigned tasks.

Tagged-MapReduce programs are similar to MR programs, the difference being that a *programmer* can program in the map and reduce routines various policies that guide how the sensitivity should be changed during execution. For instance, in the classic word-count example that reads in text files and counts how often each word occurs, one can code in the map routine that: a tuple, i.e. $\langle word, 1\rangle$ is output as sensitive iff $word$ is from a sensitive input file and not in the set of "stop words". The logic for deciding the sensitivity of the output tuples is broadly called the *sensitivity policy* in this paper (Sect. III). Fig. 2 illustrates how sensitivity policies can be programmed using the word-count example with the aforementioned security policy.

To perform a computation over tagged-MapReduce, the input data have to be tagged first by indicating the sensitive portions. As manual tagging or individual tuple-level tagging can be tedious, for simplicity and usability, our prototype implementation considers *file-level tagging*,[1] i.e., the input data consists of multiple files and each file contains either all sensitive data or all non-sensitive data. The sensitivity of input files is specified in a meta file which is then uploaded to the framework together with the input data. The underlying distributed file system then starts to replicate the data in a privacy-aware way, ensuring that sensitive files are only stored in the private cloud.

In addition, the programmer can also specify the *properties* (partitionable-reduce or unique-tag as in Sect. IV) that the map or reduce function meets. Such information helps the system to decide how to schedule the tasks using the appropriate *scheduling modes* (Sect. IV).

III. Programming Model

Original MR has map and reduce functions operating on key-value tuples. Our tagged-MapReduce framework extends the programming model of MR to support tags with the corresponding functions *tagged-map* and *tagged-reduce* operating on *tagged-tuples*. Specifically, we extend the key-value pair $\langle k, v\rangle$ in MR where k and v are binary strings with tagged-tuples of the form $\langle k, v; t\rangle$, where t is a symbol *sensitive* or *non-sensitive*,[2] and k and v are as in MR.

A *tagged-map* $\hat{\mu}$ extends a given map μ in the original MR framework. If μ on input $\langle k, v\rangle$ with a random string r as the auxiliary bits for randomness,[3] outputs a finite multiset $\{\langle k_1, v_1\rangle, \ldots, \langle k_m, v_m\rangle\}$ for some m, then the corresponding tagged-map $\hat{\mu}$ is a function that on input $\langle k, v; t\rangle$ and with r as the auxiliary data, outputs $\{\langle k_1, v_1; t_1\rangle, \ldots, \langle k_m, v_m; t_m\rangle\}$ for some tags $t, t_1, \ldots, t_m$.

Similarly, a *tagged-reduce* $\hat{\rho}$ extends a given reduce ρ. If ρ on input a multiset $\{\langle k, v_1\rangle, \langle k, v_2\rangle, \ldots, \langle k, v_n\rangle\}$ with random string r as the auxiliary bits, outputs a multiset of pairs $\{\langle k, w_1\rangle, \ldots, \langle k, w_{n'}\rangle\}$ for some n and n', then the tagged-reduce $\hat{\rho}$ is a function that on input $\{\langle k, v_1; t_1\rangle, \langle k, v_2; t_2\rangle, \ldots, \langle k, v_n; t_n\rangle\}$ with r as the auxiliary data, outputs the multiset $\{\langle k, w_1; t'_1\rangle, \ldots, \langle k, w_{n'}; t'_{n'}\rangle\}$ for some tags $t_1, t_2, \ldots, t_n, t'_1, t'_2, \ldots, t'_{n'}$.

The tags in tagged-tuples are just auxiliary data which do not affect the keys and values. Two different tagged-reduces $\hat{\rho_1}$ and $\hat{\rho_2}$ that extend the same reduce ρ but with different algorithms for deciding the output tags, will output the same distribution of keys and values.[4] Hence, our extension is conservative since the program will not be changed if all data are non-sensitive and it does not affect the output distribution. The role of the tags is to feed information to the scheduler which decides where computation is to be carried out. This segregation of roles provides clarity in coding programs to process sensitive data and in analyzing algorithms. Moreover, the tags also carry sensitivity information for data across multiple MR jobs (see

[1]File-level tagging is simple and yet does not lose generality as more sophisticated tagging, e.g., tuple-level tagging, can be simply done by having an initial tagged-MapReduce job with all input files being tagged sensitive and output tuples with the desired sensitivities.

[2]For simplicity, we use binary tags but this can be generalized.

[3]The random string r provides randomness if the computation is probabilistic, and can be omitted if μ is deterministic.

[4]The overall MapReduce computation may be non-deterministic, hence we consider the possible outputs to be a distribution. In the case that they are deterministic, they always output the same key-value pairs.

```
public void map(LongWritable key, Text value,
                Context context)
{
    // key: line number
    // value: content of this line
    String line = value.toString();
    StringTokenizer tokenizer =
                new StringTokenizer(line);
    while (tokenizer.hasMoreTokens())
    {
        String val = tokenizer.nextToken();
        word.set(val);
        context.write(word, one);
    }
}
```

```
public void map(LongWritable key, Text value,
                Context context)
{
    Boolean sensitive = false;
    String line = value.toString();
    StringTokenizer tokenizer =
                new StringTokenizer(line);
    while (tokenizer.hasMoreTokens())
    {
        String val = tokenizer.nextToken();
        sensitive = context.getInputSplit().getSensitivity()
                && !inStopWords(val);
        word.setIsSensitive(sensitive);
        word.set(val);
        context.write(word, one);
    }
}
```

Figure 2. Example code corresponding to original map (left) and tagged-map (right) for the `WordCount` job. The difference is the code within the dashed box that computes and sets the tags of the output tuples.

Sect. VI), and thus complex MR computation with chained or iterative MR can be supported naturally.

Fig. 2 gives Hadoop Java code in our prototype of tagged-map for the extended `WordCount` job working on a set of sensitive and non-sensitive files (right). Compared to the original map (left), the difference is the extra statement (in the dashed box) to compute the sensitivity of each word based on some sensitivity rules (see below) and an API `setIsSensitive()` to set the tags of output tuples.

When it is clear from the context, we will omit the word "tagged" and call tagged-tuple, tagged-map and tagged-reduce as tuple, map and reduce respectively.

A. Sensitivity Policy

In addition to normal MR programs (which do not have code for data sensitivity), with explicit tagging, programmers can now implement MR programs which are sensitivity-aware, applying a policy to govern the sensitivity of tuples created during execution in the map and reduce routines. Such policies are called the *sensitivity policies* in this paper. An example policy has the following rules: (1) A map $\hat{\mu}$ does not modify the sensitivity of the data, i.e., each tuple in the output of $\hat{\mu}$ has the same sensitivity as the input tuple; and (2) the output of the reduce is sensitive iff at least one input is sensitive. Our prototype uses this policy as the default if the program does not specify any policy. It is also how legacy (normal) MR programs are supported. Programmers can choose to implement more sophisticated and application-specific policies to override the default policy. Fig. 2 gives an example of programming sensitivity policies.

We now address the question of whether the sensitivity of an output tuple can be upgraded or downgraded. Let us first consider upgrading.

1) Non-upgrading policy: An upgrading happens if, for either a map or reduce, (all of) the input is non-sensitive but the output contains sensitive tuples. Assume that all map and reduce algorithms are public knowledge, the public servers can collude and all non-sensitive tuples are stored in the public servers, then it is meaningless to have a policy that deems the output as sensitive when all of the input data are non-sensitive, given that an adversary in the public cloud can compute the output anyway. This gives the following condition for tagged-map $\hat{\mu}$ and tagged-reduce $\hat{\rho}$:

CONDITION 1 (NON-UPGRADING MAP AND REDUCE)
Consider map $\hat{\mu}(t)$, if input tuple t is non-sensitive then all tuples output from $\hat{\mu}$ will be non-sensitive. Similarly for reduce $\hat{\rho}(k, \{t_1, \ldots, t_n\})$, if all input tuples t_i's are non-sensitive, all tuples output from $\hat{\rho}$ will be non-sensitive.

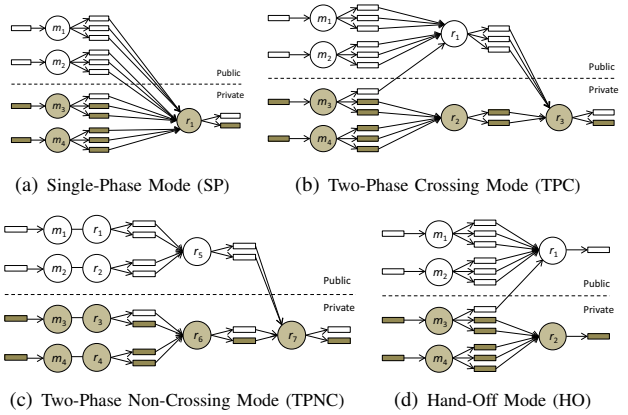
Violation of this condition during processing does not compromise confidentiality of tuples previously tagged as sensitive, and thus may not be harmful in terms of security. Nevertheless, it tags data already known to the public cloud as sensitive, and hence imposes unnecessary constrains which in turn lowers the effectiveness of the scheduler.

2) Downgrading policy: However, there are situations where the sensitivity may be "downgraded" to non-sensitive, even if the input is sensitive. The downgrading can occur in either a map or reduce. For example, consider a tagged-map that takes in a surveillance video (tagged as sensitive), analyzes the video, and outputs a set of short video clips. Video clips with a low-level of activity are to be tagged as non-sensitive, whereas video clips with a high-level of activity are to be tagged as sensitive. Here, the final sensitivity is derived from both the key and value of the input, allowing certain video clips to be downgraded from sensitive to non-sensitive. Another application is data anonymization where a tagged-reduce takes as input a list of sensitive values and outputs an aggregated value. The output value is considered "anonymized" and thus tagged as non-sensitive. In general, downgrading allows to further push computation to the public cloud and is useful for applications where the input data are sensitive but only few of them turn out to be important after simple pre-processing. Explicit tagging makes such downgrading possible.

IV. SCHEDULING MODES

After being tagged, input data are selectively distributed and replicated to the public and private clouds based on their sensitivity status with sensitive data placed in private nodes. Upon the data placement, a set of tagged-map tasks are then created, across the private and public clouds, to operate on the sensitive and non-sensitive data accordingly. After all tagged-map tasks have completed, a key can appear in tuples that are produced by both the public and private clouds with different sensitivities in different tuples. As a tagged-reduce task may receive both sensitive and non-sensitive tuples, it cannot be directly executed on the public cloud.

To prevent data leakage, a conservative scheduler might push all intermediate results produced in both clouds to the private cloud for reducing. This scheduling strategy is illustrated in Fig. 3(a) which we call the *single-phase (SP) mode*. However, SP mode may overload the private servers (i.e., public servers are not enrolled in the reduce phase) and also lead to high volumes of data flowing from the public to the private cloud during MR shuffling. Inter-cloud data



(a) Single-Phase Mode (SP) (b) Two-Phase Crossing Mode (TPC)

(c) Two-Phase Non-Crossing Mode (TPNC) (d) Hand-Off Mode (HO)

Figure 3. Scheduling modes in tagged-MR. m_i's are map tasks, r_i's are reduce tasks. Shaded blocks are sensitive tuples and shaded circles are tasks running on the private cloud.

traffic can be significant as inter-cloud bandwidth may be much smaller than the intra-cloud one (internally, within each cloud). Inter-cloud traffic may also be charged by the cloud provider.[5] What is desired is to outsource more reduce computation to the public cloud when needed while reducing the total amount of inter-cloud data movement.

To this end, we investigate certain properties of the map and reduce functions whereby if a function meets, the reduce computation can be rearranged in the two clouds for more effective load-balancing and inter-cloud bandwidth usage while maintaining MR correctness. More specifically, we consider two properties: *partitionable reduce* and *unique tag* with three *scheduling modes*: *two-phase crossing* (TPC), *two-phase non-crossing* (TPNC) and *hand-off* (HO) modes. The properties met can be coded in tagged-MR programs (we add new APIs) to set the scheduling mode used for task assignment.

A. Two-Phase Crossing Mode (Partitionable Reduce)

If a reduce function can be carried out in a "divide-and-conquer" manner, one could first enroll public workers to aggregate the non-sensitive tuples, and then combine them with the sensitive data. Let us first define the following form of distributive property on the reduce function which holds for many regular MR programs:[6]

PROPERTY 1 (PARTITIONABLE REDUCE) *We say that a tagged-reduce $\hat{\rho}$ is partitionable if for all k, L_1 and L_2*

$$\hat{\rho}(k, L_1 \cup L_2) = \hat{\rho}(k, \hat{\rho}(k, L_1) \cup \hat{\rho}(k, L_2)) \quad (1)$$

If $\hat{\rho}$ is partitionable, then it can be performed in two phases:

1) (Phase 1) For each key k, a worker p_k (can be private or public) is selected and assigned to perform a reduce task on all non-sensitive tuples with k. A private worker q_k (possibly different) is selected and assigned to perform a reduce task on all sensitive-tuples with k.
2) (Phase 2) A private worker is selected and assigned to perform a reduce task on the output of p_k and q_k for each key k.

Fig. 3(b) illustrates the above process. Since sensitivity can be downgraded, map tasks running on the private cloud may produce many non-sensitive tuples. This mode allows such tuples to be passed to the public cloud for partial reducing. In general, this mode leads to higher utilization of public servers but may incur increased dataflow from the private to the public cloud during shuffling.

B. Two-Phase Non-Crossing Mode

This mode is a potential improvement of the above two-phase crossing mode, whereby the reduce function is first applied on each map-task's output locally (like the `Combiner` in MR). This local-reduce phase typically can reduce the size of the intermediate results, thus, speeding up the internal shuffling and sorting phase. After the local-reduce phase, the produced intermediate results are first aggregated on the public and the private cloud separately, and then combined on the private cloud, as illustrated in Fig. 3(c). In particular, data downgraded in the local-reduce phase still remains in the private cloud for subsequent processing to prevent unintentional information leakage.

Compared to the two-phase crossing mode, under this mode, the utilization of public servers is expected to be lower, but the volume of inter-cloud data traffic may decrease.

C. Hand-Off Mode (Unique Tag)

In both of the above TPC and TPNC modes, an additional phase (i.e., phase 2) is required to combine the partial reduce outputs produced in the two clouds as a key may be associated with both sensitive and non-sensitive tuples. Now we consider a property of the map function whereby this additional combining phase is not required.

PROPERTY 2 (UNIQUE TAG) *Given a multiset of tagged-tuples U, we say that the keys in U have unique tag if there does not exist a key k such that both $\langle k, v;$ sensitive$\rangle$ and $\langle k, v';$ non-sensitive$\rangle$ are in U for some v and v'.*

We say that a map function meets the unique tag property if, on any input and any execution, completion of the map phase gives a set of tagged output tuples with unique tag.

An example of the unique tag property is a map function that outputs $\langle k, v; t\rangle$ where t is a deterministic function of k. When a map function meets the unique tag property, after the map phase, a key appears either in the sensitive tuples or in the non-sensitive tuples, but never in both. Then, there is an easy way to schedule the reduce tasks – simply assign keys tagged as sensitive to private workers, and keys tagged as non-sensitive to either a public or a private worker. Since

[5] Amazon EC2 does not charge for data transfers in the same Availability Zone but charges for data transferred out to other regions or the Internet.

[6] This is the commutative and associative property that the `Combiner` function satisfies in MR.

no combination or morphing of tasks is required, we call this mode the *hand-off* mode. Fig. 3(d) illustrates this mode.

V. Security Analysis

We consider public servers to be *honest-but-curious*, that is, they will carry out the assigned computation honestly but may retain knowledge derived from the computation for malicious purposes. We assume that the adversary may have control over all the public servers, thus we allow public servers to collude. In addition, we assume that the identities of all private servers, the scheduling algorithms and the map and reduce operations are public information.

As different scheduling algorithms can lead to different dataflows during execution, the actual data sent to the public cloud can differ. Hence, from the curious public cloud's viewpoint, the amount and types of information leaked by different schedulers can be different. To illustrate, let us consider the following two examples.

1) Example I: Consider a simple reduce function that on input a list of values with the same key k, outputs $\langle k, (s, m)\rangle$ where s is the sum of the values, and m is the total number of input tuples. The output is tagged as *non-sensitive* iff m is greater than a threshold, say 50. Note that this reduce function is partitionable. An ambitious scheduler might divide the sensitive input tuples into groups of 50 and assign the reduce task on each group to a private worker. Next, the aggregated non-sensitive output from each group is sent to a public worker for further aggregation. Now, let us consider a conservative scheduler which assigns all reduce tasks to private workers. Compared to this conservative scheduler, the ambitious scheduler will reveal the sum of each group to the public cloud. One may argue that the sum of any sufficiently large group is deemed to be non-sensitive by the programmer and thus it is acceptable to reveal the sums of many subgroups. However, that may not be the intention of the programmer and hence we need a clear security model to establish a baseline.

2) Example II: Here is a more subtle example. Consider another ambitious scheduler that dynamically tracks the intermediate tuples generated by the map tasks. If a particular key k has only non-sensitive intermediate tuples, then the scheduler will assign the reduce task on k to a public worker. Since no sensitive tuple is sent to the public cloud, it seems that this scheduler does not leak sensitive information. Now, compared to the conservative scheduler described before, the action of this scheduler reveals an additional piece of information on k: the fact of whether there exists a sensitive intermediate tuple with the key k. Although this piece of information seems to be insignificant, it could be a crucial leakage in certain scenarios. Hence, we need a security model that clearly accepts or disallows such leakage.

The above two examples bring out the subtlety and challenge in formulating the security model. What should be the "baseline" of leakage that is acceptable, and how to compare the leakages incurred by different schedulers? We handle this issue by treating the conservative scheduler described above as the baseline, and propose a security model to compare a scheduler with this conservative scheduler. Essentially, we say that a scheduler S_1 does not leak more than another S_2 iff we can simulate S_1 and generate the information revealed by S_1 based on the information revealed by S_2. Schedulers that do not leak more than the baseline are considered secure. Due to the space constraints, we only describe the essentials of the model without a full formal treatment.

A. Public-View of an Execution

Given an input D, let us consider what information the public cloud can gather during an execution on D. We assume that servers in the public cloud can collude, thus, we treat them as a single entity. During the execution, the public cloud can see the content of each tuple it handles, including the identities of the private workers who send and receive those tuples, and the internal state of all public workers. Let us call the information revealed the *public-view* of an execution on D. Note that on a same instance D, the execution could be different since the scheduler and the job could be non-deterministic. Hence, we are interested in the distribution of the public-views.

B. Baseline - the Conservative Scheduler

Now consider a conservative scheduler. This scheduler assigns map tasks operating on sensitive and non-sensitive tuples to randomly chosen private and public workers respectively, and assigns all reduce tasks to randomly chosen private workers.[7] In addition, all intermediate non-sensitive tuples will be sent to some public workers for temporary storage (i.e., all intermediate non-sensitive tuples are public information). Hence, the public-view of an execution under this conservative scheduler includes the content of all non-sensitive tuples and the identities of the workers who generate and read the non-sensitive tuples. This scheduler is "conservative" since it does not attempt to re-arrange the tasks for better performance.[8]

While there may be still some information leakage by virtue of data going to the public cloud, one assumes by definition that non-sensitive tuples can be disclosed. Since the goal is outsourcing of "some" computations to the public cloud, such leakage is considered to be acceptable. In this sense, the conservative scheduler is reasonable for analyzing the security of scheduling algorithms. Hence, we choose this simple execution model as the baseline and call it the *baseline scheduler*.

[7]The selection of a private worker depends on a few criteria, including the servers' configuration and workload. We assume that such information does not leak useful information to the adversary. Hence, in our security analysis, we consider a selection that randomly picks a private worker from the pool of all private workers.

[8]This conservative scheduler is similar to the SP mode in behavior but exposes all non-sensitive tuples to the public cloud.

C. Security Model

Given a particular scheduler S, we want to analyze and determine whether it leaks "more" than the baseline scheduler. For a particular job and input D, let us consider an oracle $\mathcal{O}_D$ that, on request, generates a public-view of the baseline scheduler on D. Note that this is just one sample from the distribution of public-views. Now, let us consider a simulator who has access to the oracle once. Based on the public-view generated by the oracle, this simulator attempts to generate a sample of the public-view of the scheduler S in question. We say that a scheduler S does not leak additional information on a particular job if the following holds:

> *There exists a simulator that, for any D, its output is statistically close to the public-view of S on the input D.*

Note that in the above definition, the simulator can be different for different jobs. A scheduler that does not leak additional information on any job is considered to be secure.

Using this security model, we are able to show that the two scheduling examples given at the beginning of this section, while potentially improving efficiency, indeed leak more information than the baseline scheduler while our proposed scheduling modes do not.

D. Side-Channel Information

During execution, adversaries in the public cloud could measure the size and timing of packets received from each private server. Analysis of such network traffic might provide information on the workload of individual private servers. The workload may depend on the actual content of sensitive tuples, which although unlikely, could potentially leak information of the content. Since the network traffic is heavily influenced by other factors like overall network conditions and time of the day, and these information may still present even if encrypted computation (e.g., homomorphic encryption) is used, we consider these as "side-channel" information and do not capture them in our security model. Nevertheless, there are mechanisms to reduce such leakage such as hiding the identities of the private servers by routing the traffic to a proxy and "translating" the identities (which is similar to NAT (Network Address Translation)), inserting random delays to the traffic, adding noise in the scheduling, etc. The issue of side-channels is orthogonal to this paper. An interesting future work is to study how these mechanisms can be combined to reduce such side-channel leakage.

VI. Implementation and Evaluation

We implement a prototype of tagged-MapReduce based on Hadoop 1.0.1. Essentially, two components of Hadoop are updated: the Distributed File System (HDFS) and the MR execution framework. For the HDFS, the `INodeFile` class is extended with a tag giving the sensitivity of the corresponding file. The `NameNode` recognizes the tags and distributes the files in a privacy-aware manner: files tagged as sensitive are always replicated to the private servers while there is no constraint for non-sensitive files. For the MR execution framework, we add to each task (map and reduce) a sensitivity label whose value is to be determined from the sensitivity of the data to be processed, and modify the `JobTracker` to assign the tasks based on their security labels. We also modify the default `HashPartitioner` and have special treatment for the TPC and TPNC modes where there are two reduce phases.

We experiment with our prototype on Amazon EC2 to evaluate the practicality of the system and the effectiveness of the proposed scheduling strategies in terms of: (i) inter-cloud communication cost; (ii) total job running time; and (iii) computation outsourcing ratio. The experiments are run using both simple (single) and complex (chained) jobs.

A. Experimental Setting

1) Computing jobs and datasets: Many analytical tasks and applications such as target marketing, spam detection and medical processing need to deal with both public and confidential data simultaneously [7]. In this experiment, we run two basic MR jobs where it is natural to have data with mixed sensitivity: *word count* (WC) and *face detection* (FD). The word count job, which is an extension to the classic MR example [10], counts the occurrences of each word in a set of sensitive and non-sensitive text files. For the map function, only words from the sensitive input files and not in the set of stop words are tagged as sensitive (see Fig. 2). For the reduce function, the aggregated count of a word is tagged as sensitive iff at least one of its inputs is sensitive. The input is a dataset of English Wikipedia articles [4] which is partitioned into 10 separate files about the same size. During experiments, we will vary the ratio of sensitive data over the whole input dataset. The sensitivity of each of the 10 input files are randomly assigned to match a required ratio.

The face detection job detects faces from a database of 80,000 images crawled from the web via Google Images. In the experiment, to obtain a particular ratio of sensitive data over the whole image dataset, an appropriate number of randomly selected images are tagged as sensitive. The output of the job are extracted images of successfully detected faces where a face is tagged as sensitive if it is in a sensitive image which contains no more than 3 faces.

The above two basic jobs, WC and FD, can be each carried out in a single MR job. We also experiment on variations of these two jobs with chained MR: *wordcount+sort* and *face anonymization* as summarized in Table I.

2) Hybrid cloud setting: We build a hybrid cloud on Amazon EC2. The private cloud consists of 3 instances located at Singapore and the public cloud has 3 instances at US West (N. California). All instances (m1.large) run Ubuntu 12.04, and each provides 2 virtual cores with 4 Amazon ECUs, 7.5 GB memory and 850 GB storage. The bandwidth between these instances is not specified. An

Table I
SUMMARY OF THE COMPUTING JOBS AND DATASETS

Application	Jobs	Description of Jobs	Dataset	Description of Dataset
Wordcount+Sort	Wordcount (WC)	Count the occurrence of each word in text files	Wikipedia dataset	English Wikipedia articles up to July 2012 (36.8GB)
	Sorting (ST)	Sort the words by the number of occurrence		
Face Anonymization	Face detection (FD)	Detect human faces from each image	Image dataset	Images of human faces crawled via Google Images (17.2GB)
	Averaging (AG)	Replace each face by the average of 5 neighboring faces		
	Sorting (ST)	Sort the anonymized faces by image name		

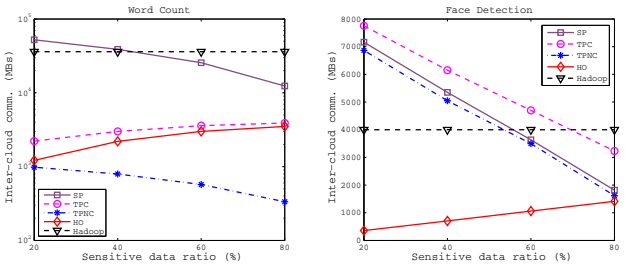


Figure 4. Inter-cloud communication.

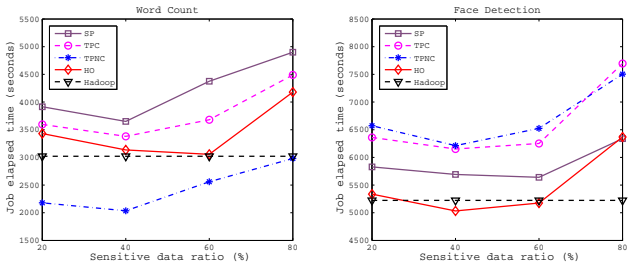


Figure 5. Job elapsed time.

informal test of file transfer using `scp` gives 35–40 MB/s within the same region (e.g., from Singapore to Singapore), and 3–8 MB/s across different regions (e.g., from Singapore to California). Though our cluster is small, our dataset size is quite modest and hence matches the cluster.

B. Experiments on Scheduling Modes

The two basic jobs WC and FD are run under the four scheduling modes presented in Sect. IV and also on original Hadoop over the whole hybrid cloud (6 nodes). The original Hadoop run is meant for performance comparisons although it does not meet the security requirement of keeping sensitive information in the private cloud. For each mode, we vary the ratio of sensitive data over the whole input between 20% to 80%. The total job running time (job elapsed time), the execution time of each task (individual CPU time) and the data transferred across the public and private cloud (inter-cloud communication) can be derived from the Hadoop log files.

The hand-off mode requires the unique tag property which is not met by the word count job as a same word can occur in a sensitive and a non-sensitive file. Fortunately, we can use the following observation. Given a map $\hat{\mu}$ which produces $\langle k, v_1, s\rangle$ and $\langle k, v_2, n\rangle$ where s and n denote sensitive and non-sensitive, a simple transformation to meet the unique tag property is to instead output $\langle 1.k, v_1, s\rangle$ and $\langle 0.k, v_2, n\rangle$ where the operator . denotes bit concatenation. Face detection uses the image name as the key which is either sensitive or not, thus fits the unique tag property.

1) Inter-cloud communication: This measures how much data is transferred across the two clouds during execution, which could be a potential performance bottleneck due to the limited inter-cloud bandwidth. Fig. 4 shows the result.

The result of the word count job shows the effectiveness of our proposed modes in reducing the inter-cloud communication. Note that the y-axis is in logarithmic scale. The TPC, TPNC and HO modes can reduce the total amount of inter-cloud data transfers by orders of magnitude compared to the original Hadoop or the SP mode. In this job, the TPNC mode incurs the least inter-cloud communication overhead as the reduce output is much smaller in size than the map output, hence the data (i.e., partial reduce output) transferred from the public to the private cloud is smallest.

The result of the face detection job is different. Surprisingly, the TPC mode incurs larger inter-cloud data traffic than the SP mode or original Hadoop. We observed that this is due to the behavior of the reduce function in the face detection job, which simply converts detected faces to images and outputs them. This means that the output from reduce is similar in size to the map output. The TPC mode moves all the intermediate results produced on the public cloud, together with those downgraded from the private cloud, back to the private cloud for final reduce, and hence incurs the largest inter-cloud data movement. In contrast, the SP and TPNC modes only involve one step of data movement, giving less inter-cloud data traffic. The HO mode has the least inter-cloud communication cost indicating that very few sensitive images are downgraded to be non-sensitive.

2) Job elapsed time: This measures how long it takes to compute a MR job. The result is illustrated in Fig. 5. It is not surprising that most of our modes require a longer time to compute a same MR job than original Hadoop mainly due to the security requirements. When the sensitive data ratio is lower than 50%, there are roughly equal data processed on the public and private cloud, hence the time is low. As the ratio increases, the private nodes bear the burden of increased data so the time increases. Our modes can outperform the basic SP mode in the word count job. However, in the face detection job, both the TPC and TPNC modes incur longer time than SP. Again this is due to the reduce behavior of the face detection job, which does not

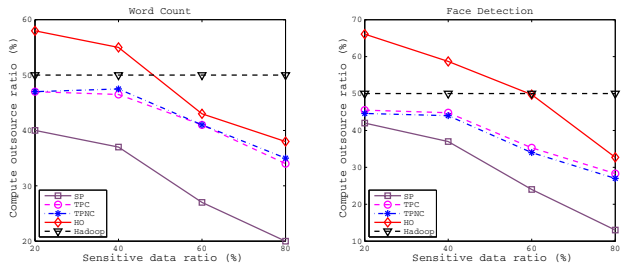


Figure 6. Computation outsourcing ratio.

Table II
MODE ASSIGNMENT

Application	Job	Mode
Wordcount+Sort	Wordcount	Two-Phase Non-Crossing
	Sorting	Single-Phase
Face Anonymization	Face detection	Hand-off
	Averaging	Hand-off
	Sorting	Single-Phase

decrease the data size but incurs computational overhead. The HO mode in both jobs can approach the performance of Hadoop at a ratio of around 50% due to the optimized inter-cloud data traffic. The TPNC mode in word count even outperforms Hadoop. This indicates that inter-cloud data movement is indeed a significant performance bottleneck.

3) Computation outsourcing ratio: The computation outsourcing ratio gives the percentage of total CPU time used in the public cloud over the total CPU time. It measures how much compute is outsourced to the public cloud. The result is given in Fig. 6. The Hadoop baseline is around 50% which is expected as it randomly assigns tasks to all the nodes. The SP mode assigns all reduce tasks to the private cloud and thus serves as a lower bound of the other modes. The HO mode achieves the best outsourcing as it saves the final reduce phase in the private cloud. The TPC and TPNC modes are close to each other. Further, as the sensitive data ratio increases, naturally the outsourcing ratio decreases since less work is available to be outsourced.

In summary, the above results demonstrate that, our proposed scheduling modes (i.e., TPC, TPNC and HO) in a hybrid cloud setting can effectively reduce the inter-cloud communication and job execution time while being able to outsource more computation to the public cloud, as compared to the general SP mode. The time overheads are also reasonable as compared to original Hadoop runs on the same hybrid cluster.

C. Experiments with Chained MapReduce

Next, we experiment on more complex MR jobs that involve chained MR: *wordcount+sort* and *face anonymization*. We choose the most appropriate mode, as summarized in Table II, for each individual job in the chain according to the properties of the computation. The ratio of sensitive data over the whole input is around 50% in both jobs. The result is compared with two columns, running with the (default) SP mode and original Hadoop.

Table III
EXPERIMENTAL RESULTS OF CHAINED MAPREDUCE

App.	Job	Hadoop		SP Mode		Assigned Modes	
		Time (sec)	Traffic (MBs)	Time (sec)	Traffic (MBs)	Time (sec)	Traffic (MBs)
Wordcount + Sort	WC	3020	36551	4082	32150	2275	685
	ST	278	505	307	621	307	621
	Total	**3298**	**37056**	**4389**	**32771**	**2582**	**1305**
Face Anony-mization	FD	5220	4109	5776	4521	5093	847
	AG	2158	1020	2534	1453	2064	0
	ST	483	1007	528	1407	528	1407
	Total	**7861**	**6136**	**8838**	**7381**	**7685**	**2254**

Table III gives the overall job execution time and inter-cloud data traffic for each complex job. The result shows that the total amount of inter-cloud data traffic can be significantly reduced with the appropriate modes. For example, knowing that the first two jobs of face anonymization meet the unique tag property, we can assign to them the HO mode, where data are separately processed in the public and the private cloud in parallel. The sensitivity information is then naturally handed over to the next job, so there is no need to transfer the output of each job back to the private cloud. Such avoidance of unnecessary data movement leads to further optimization in inter-cloud communication across multiple MR jobs. Overall, we can reduce the inter-cloud data traffic by more than 90% for the wordcount+sort job and around 70% for the face anonymization job as compared to the SP mode or original Hadoop.

The total elapsed time also has significant improvements with correctly chosen modes compared to the SP mode. The times are also comparable to the original Hadoop runs. We remark that choosing the mode can be done automatically by the system if the properties of the MR job are specified. In summary, the total overheads for the hybrid framework are reasonable for realistic complex MR jobs in the hybrid cloud setting with data confidentiality constraints. We believe that in many cases, the conditions for the non-single phase modes can be met which lead to further optimizations.

VII. RELATED WORK

1) Comparison with Sedic: Sedic [21] is closely related to our work but with fundamental differences. It takes a different sanitization approach whereby data are duplicated to both clouds, but with sensitive portions sanitized in the public cloud. However, Sedic is less flexible for complex MR computation with chained or iterative MR. We address this problem by explicitly tagging. With tagging, data directly carry sensitivity information which can be fed to the next job, and thus multiple MR computation can be carried out naturally. This flexibility also allows legacy MR code to be easily supported. Besides flexibility issues, the sanitization approach also reveals relative locations and length of sensitive data, which potentially could leak important information [14]. In contrast, data in our framework are segregated according to their sensitivity and distributed to the two clouds separately. Since the segregation of data does not explicitly reveal the locations of sensitive data, the

proposed approach is arguably more secure. Furthermore, tagged-MR is also more expressive. Sedic can be expressed as a special case of our model (with a single-phase mode and default tagging policy) but tagged-MR programs with expressive security policies and sensitivity downgrading are not catered to in Sedic. In addition, Sedic does not consider the problem of a general security framework for analyzing of data leakage on a hybrid cloud which we do.

2) Data security in hybrid clouds: The idea of partitioning data and computation to preserve data-privacy has been discussed in various settings [5], [8], [22]. With the emergence of cloud computing, there are growing research interests of applying the idea on hybrid clouds. Ko et al. [13] propose HybrEx which partitions MR data and computations over a hybrid cloud according to some data labels similar to tagging. An outline of HybrEx is proposed but without any details or implementation. Bugiel et al. [6] propose using the private cloud to encrypt and verify the intensive computation performed in the untrusted public cloud. On the issues of secure query processing, Curino et al. [9] and Oktay et al. [15] similarly investigate partitioning of relational databases between the public and private clouds, taking into account of both the security requirements and efficiency. Other work includes distributing human genomic computation to hybrid clouds so as to protect sensitive DNA information [7], [19].

3) Differential privacy: Differential privacy [11] provides strong assurance in protecting individuals' privacy. The Airavat platform [18] incorporates differential privacy mechanisms into the MR execution framework by automatically adding noise to the output data. In comparison, we focus on the hybrid cloud setting and protect data-privacy by dataflow control. Since no noise is added to the output, computation accuracy is maintained.

VIII. Conclusion

The hybrid cloud is a practical approach for scaling computation and data processing needs. However, the seamless inter-cloud dataflow also increases the risk of information leakage if sensitive data can flow freely to the public cloud without proper protection. We present tagged-MapReduce, a secure and practical solution for computing in the hybrid cloud by extending MR with sensitivity tags. Our goal is to give a simple conceptual framework for programmers who are already familiar with MR and want to have data privacy awareness in the hybrid cloud setting. It also allows complex MR programs which can have expressive security policies and multiple chained MR jobs. We also pair this with a general security framework which is suitable for analyzing what kind of information a scheduler can leak through execution in the hybrid cloud against a baseline scheduler. Our experiments demonstrate the effectiveness of the scheduling modes in outsourcing computation and reducing the inter-cloud bandwidth usage. Tagged-MapReduce in the hybrid cloud only incurs small overheads compared to a Hadoop run which ignores the data confidentiality and security constraints, and thus is fairly practical. The framework can also handle legacy MR code.

References

[1] Forecast for 2010: The Rise of Hybrid Clouds. http://gigaom.com/2010/01/01/on-the-rise-of-hybrid-clouds/, 2010.

[2] 2012 Cloud Computing Survey. Online at http://northbridge.com/2012-cloud-computing-survey, 2012.

[3] Dropbox: Yes, we were hacked. Online at http://gigaom.com/2012/08/01/dropbox-yes-we-were-hacked/, 2012.

[4] English wikipedia dumps. Online at http://dumps.wikimedia.org/enwiki/, 2012.

[5] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *CIDR*, 2005.

[6] S. Bugiel, S. Nürnberger, A. Sadeghi, and T. Schneider. Twin clouds: An architecture for secure cloud computing. In *Workshop on Cryptography and Security in Clouds*, 2011.

[7] Y. Chen, B. Peng, X. Wang, and H. Tang. Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds. In *NDSS*, 2012.

[8] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng. Secure web applications via automatic partitioning. In *ACM SIGOPS Operating Systems Review*, 2007.

[9] C. Curino, E. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational cloud: A database service for the cloud. In *CIDR*, 2011.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[11] C. Dwork. Differential privacy. In *ICALP*, 2006.

[12] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[13] S. Y. Ko, K. Jeon, and R. Morales. The HybrEx model for confidentiality and privacy in cloud computing. In *USENIX HotCloud*, 2011.

[14] D. Lopresti and A. L. Spitz. Quantifying information leakage in document redaction. In *ACM Workshop on Hardcopy Document Processing*, 2004.

[15] K. Y. Oktay, V. Khadilkar, B. Hore, M. Kantarcioglu, M. Mehrotra, and B. Thuraisingham. Risk-aware workload distribution in hybrid clouds. In *CLOUD*, 2012.

[16] K. Ren, Y. Kwon, M. Balazinska, and B. Howe. Hadoop's adolescence: An analysis of Hadoop usage in scientific workloads. In *VLDB*, 2013.

[17] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *CCS*, 2009.

[18] I. Roy, S. T. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for MapReduce. In *NSDI*, 2010.

[19] R. Wang, X. Wang, Z. Li, H. Tang, M. K. Reiter, and Z. Dong. Privacy-preserving genomic computation through program specialization. In *CCS*, 2009.

[20] C. Zhang, E. C. Chang, and R. H. C. Yap. Towards a general framework for secure MapReduce computation on hybrid clouds. In *SOCC*, 2013.

[21] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan. Sedic: Privacy-aware data intensive computing on hybrid clouds. In *CCS*, 2011.

[22] L. Zheng, S. Chong, A. C. Myers, and S. Zdancewic. Using replication and partitioning to build secure distributed systems. In *S&P*, 2003.