D. Nuñez, I. Agudo, and J. Lopez, "Delegated Access for Hadoop Clusters in the Cloud", *IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014)*, pp. 374-379, 2014.
http://doi.org/10.1109/CloudCom.2014.50
NICS Lab. Publications: https://www.nics.uma.es/publications

Delegated Access for Hadoop Clusters in the Cloud

David Nuñez, Isaac Agudo, Javier Lopez
Network, Information and Computer Security Laboratory
Universidad de Málaga, Spain
Email: {dnunez, isaac, jlm}@lcc.uma.es

***Abstract*—Among Big Data technologies, Hadoop stands out for its capacity to store and process large-scale datasets. However, although Hadoop was not designed with security in mind, it is widely used by plenty of organizations, some of which have strong data protection requirements. Traditional access control solutions are not enough, and cryptographic solutions must be put in place to protect sensitive information. In this paper, we describe a cryptographically-enforced access control system for Hadoop, based on proxy re-encryption. Our proposed solution fits in well with the outsourcing of Big Data processing to the cloud, since information can be stored in encrypted form in external servers in the cloud and processed only if access has been delegated. Experimental results show that the overhead produced by our solution is manageable, which makes it suitable for some applications.**

I. Introduction

Big Data implies the use of vast amounts of data that makes processing and maintenance virtually impossible from the traditional perspective of information management. Apart from these inherent difficulties, the use of Big Data faces also security and privacy challenges, since in some cases the information stored is sensitive or personal data. These repositories of unprotected and sensitive information are a magnet for malicious agents (insiders and outsiders), which can make a profit by selling or exploiting this data. In most cases, companies and organizations store these repositories in clear, since security is delegated to access control enforcement layers, which are implemented on top of the actual data stores. Nevertheless, the truth is that although enforcement mechanisms that control access to data exist, some technical staff, such as system administrators, are often able to bypass these traditional access control systems and read data at will, e.g., directly in clear on the file system. Therefore, it is of paramount importance to rely on stronger safeguards such as the use of cryptography, as recently noted by the Cloud Security Alliance in [1]: "*[...] sensitive data must be protected through the use of cryptography and granular access control*".

Within the Big Data community, Apache Hadoop [2] stands out as the most prominent framework for processing big datasets. Apache Hadoop is a framework that enables the storing and processing of large-scale datasets by clusters of machines. The strategy of Hadoop is to divide the workload into parts and spreading them throughout the cluster. However, even though Hadoop was not designed with security in mind, it is widely used by organizations that have strong security requirements regarding data protection.

In this paper, we propose a delegated access solution for Hadoop, which uses proxy re-encryption to construct a cryptographically-enforced access control system. The goal of our proposal is to enable Hadoop to achieve data protection while preserving its capacity to process massive amounts of information. This way organizations can securely leverage the value of Big Data for their business, in compliance with security and privacy regulations. Experimental results show that the overhead produced by our solution is manageable, which makes it suitable for some applications.

In order to illustrate the relevance of our proposal, we provide a motivation scenario where a delegated access solution for Hadoop would be extremely useful. Consider the rise of Big Data Analytics, which represents a new opportunity for organizations to transform the way they market services and products through the analysis of massive amounts of data. However, small and medium size companies are not often capable of acquiring and maintaining the necessary infrastructure for running Big Data Analytics on-premise. As it has already happened for multitude of services, the Cloud paradigm represents a natural solution to this problem. The idea of providing Big Data Analytics as a Service [3], [4], [5] is a very appealing solution for small organizations, so they can count on on-demand high-end clusters for analysing massive amounts of data. This idea makes even more sense nowadays, since a lot of organizations are already operating using cloud services, and therefore, it is more sensible to perform analytics where the data is located (i.e., the cloud). Nevertheless, the adoption of the cloud paradigm does not come at no price. There are several

risks, such as the ones that stem for a multi-tenant environment. Jobs and data from different tenants are then kept together under the same cluster in the cloud, which could be unsafe when one considers the weak security measures provided by Hadoop. The use of encryption for protecting data at rest can decrease the risks associated to data disclosures in such scenario. Our proposed solution fits in well with the outsourcing of Big Data processing to the cloud, since information can be stored in encrypted form in external servers in the cloud and processed only if access has been delegated.

II. The Hadoop Framework

Hadoop is a framework for processing massive amounts of data in a large-scale, distributed way. Hadoop adopts the MapReduce programming paradigm, which permits to spread the workload across a cluster of machines, usually hundreds or thousands. In Hadoop, all the operations or tasks are executed by nodes in the cluster. There are two kinds of active elements in Hadoop: (i) the JobTracker, which distributes the workload across the cluster by assigning individual tasks to worker nodes and is in charge of its coordination, and (ii) the TaskTrackers, which simply execute the tasks they are assigned.

In the MapReduce paradigm, each portion of the workload must be independent from the others, in order to leverage the potential of massive parallelization. For this reason, a MapReduce job is designed to be executed in two phases, Map and Reduce. In the Map phase, the input data is splitted and processed parallely by the cluster. For each data split, the JobTracker assigns a Map task to a TaskTracker that has an available slot (a single TaskTracker can handle several tasks). The output of each Map task is a list of key-value pairs. Roughly speaking, each individual data record in a split is used for producing a key-value pair during this phase. Next, this intermediate data is partitioned and sorted with respect to the key, and stored locally. For each partition, the JobTracker assigns a Reduce task to an available TaskTracker. Now the Reduce tasks have to fetch the intermediate data generated in the previous phase; for this reason, this part represents the main communication bottleneck in a MapReduce job. Once intermediate data is retrieved, each Reduce task sorts and merges all his partitions, and the Reduce operation is executed. Finally, the data is combined into one or a few outputs.

The Hadoop framework also defines a special filesystem designed for achieving fault-tolerance and high throughput for large-scale processing, called Hadoop Distributed File System (HDFS); however, Hadoop can be used with other data sources. In HDFS, files are split in large blocks of a determined size (default size is 64MB), which are replicated and randomly distributed across the cluster of machines. One of the most prominent characteristics of Hadoop is that it leverages data locality for reducing communication overhead. In order to do so, Hadoop exploits the topology of the cluster by assigning tasks to nodes that are close to the input data, preferably local to it. Another important aspect is the way it provides fault-tolerance. In the event of task failure, Hadoop handles the failure automatically by re-assigning the task to a different node, taking advantage of the multiple copies of each block.

Hadoop clusters usually store huge amounts of data from different sources, owners and degrees of sensitivity. However, because of its nature, Hadoop can be considered as a multi-tenant service and several jobs from different users can be executed at the same time on the cluster. Also, in the case of HDFS, data is distributed evenly through the cluster, so it is possible that one node stores and process data from different tenants at the same time, which can also introduce security threats, such as accessing to intermediate output of other tenants, to concurrent tasks of other jobs or to HDFS blocks on a node through the local filesystem [6]. Some of these problems could be mitigated using a cryptographically-enforced access control approach, such as our proposal.

III. Proxy Re-Encryption

Our proposal uses proxy re-encryption, a special kind of cryptographic scheme that enables the construction of our access delegation system. In this section, we explain what is proxy re-encryption and how it is used for fulfiling our purposes.

From a high-level viewpoint, a proxy re-encryption scheme is an asymmetric encryption scheme that permits a proxy to transform ciphertexts under Alice's public key into ciphertexts decryptable by Bob's secret key. In order to do this, the proxy is given a re-encryption key $rk_{A\to B}$, which makes this process possible. So, besides defining traditional encryption and decryption functions, a proxy re-encryption scheme also defines a re-encryption function for executing the transformation.

Proxy re-encryption can be seen as a way to accomplish access delegation; thus, one of the immediate applications of proxy re-encryption is the construction of access control systems, as already shown in one of the first proposed and most well-known schemes [7]. A cryptographically-enforced access control system using proxy re-encryption can be constructed as follows. Let us assume a scenario with three entities: (i) Alice, the data owner, with public and private keys pk_A and sk_A; (ii) Bob, with public and private keys pk_B and sk_B; and

(iii) the proxy entity, which permits access through re-encryption. The data to be protected is encrypted with a fresh symmetric encryption key, the *data key*, and this key is in turn encrypted using the proxy re-encryption scheme and the public key of the data owner pk_A in order to create an *encrypted lockbox*. Encrypted data, together with its associated lockbox, can now be stored in an untrusted repository, even publicly accessible. This process can be made with differents levels of granularity for the data (file-level, block-level, record-level, etc.). Note also that the generation of encryption data can be done by any entity that knows the public key of Alice. Now, the data owner Alice, can generate re-encryption keys for authorized entities and pass them to the proxy entity. In this case, the re-encryption key $rk_{A\to B}$ can be seen as an access delegation token that the owner of the data creates in order to enable Bob access to his data. Each time that Bob wants to access encrypted data, he has to ask the proxy for the re-encryption of the lockbox, so he can decrypt it using his private key sk_B.

We will borrow this idea for constructing a cryptographically-enforced access control scheme for Hadoop. There are plenty of proxy re-encryption schemes that could be used to this end; however, most proxy re-encryption schemes use pairing-based cryptography, which makes them very costly in terms of computation, and hence, less suitable for high-requirement environments like ours, where efficiency is paramount. In this proposal, we will use a scheme from Weng et al [8], which is proven CCA-secure, and unlike most of others proxy re-encryption schemes, it is not based in costly bilinear pairing operations.

Another useful property of this scheme is that it is *transitive*, which means that anyone knowing $rk_{A\to B}$ and $rk_{B\to C}$ can derive $rk_{A\to C}$. This follows from the fact that in this scheme $rk_{A\to B} = sk_B \cdot sk_A^{-1}$, so $rk_{A\to C} = rk_{A\to B} \cdot rk_{B\to C}$. This property will allow us to derive re-encryption keys from a single "*master*" re-encryption key generated by the data owner. This is explained in more detail in Section IV-B.

IV. Re-Encryption-based Delegated Access System for Hadoop

In this section, we describe a cryptographically-enforced access control system for Hadoop, based in proxy re-encryption. By using it, the data is stored in encrypted form and the owner can delegate access rights to the computing cluster for processing. In our proposal, the data lifecycle is composed of three phases:

1) Production phase: during this phase, data is generated by different data sources, and stored encrypted under the owner's public key for later processing.
2) Delegation phase: in this phase, the data owner produces the necessary master re-encryption key for initiating the delegation process; once this phase concludes, the data owner does not need to participate again.
3) Consumption phase: This phase occurs each time a user of the Hadoop cluster submits a job; is in this phase where encrypted data is read by the worker nodes of the cluster. At the beginning of this phase, re-encryption keys for each job are generated.

A. Production phase

This phase comprises the generation of the data by different sources and its storage in encrypted form. We assume a scenario where for each dataset, there are multiple data sources and only one dataset owner. In our proposal, we establish that data of each owner is stored encrypted using his public key pk_{DO}. One advantage of using here a public key cryptosystem is that input data can be generated from disparate sources and still be protected from its origin, without requiring to agree on a common secret key. Let us assume that a data producer (which can be either the dataset owner himself or an external source) stores a file into a cluster with HDFS, and this file is splitted in N blocks $(b_1, ..., b_N)$. Recall that, when a job is submitted to the cluster, Hadoop first splits input data and then assigns a Map task for each split. In the most common case, Hadoop is implemented using HDFS as the underlying filesystem, so each split will usually match a HDFS block. From now on, we will assume then that data is encrypted on a block-by-block basis since this is the more efficient approach, although our solution could be adapted to other levels of granularity and other filesystems.

For each data block b_i, the data producer generates a fresh symmetric key r_i that is used for encapsulating the data through a symmetric encryption scheme E^{sym}, such as AES. Encrypted data is then of the form $E^{sym}_{r_i}(b_i)$. The data key r_i is in turn encapsulated using the encryption function E^{pre} of the proxy re-encryption scheme with the public key of the dataset owner, pk_{DO}, obtaining an *encrypted lockbox* $E^{pre}_{pk_{DO}}(r_i)$. Thus, for each block b_i, we obtain a pair of the form $(E^{pre}_{pk_{DO}}(r_i); E^{sym}_{r_i}(b_i))$, which is the data that is finally stored.

B. Delegation phase

The goal of this phase is that the dataset owner produces a master re-encryption key mrk_{DO} to allow the delegation of access to the encrypted data. This master re-encryption key is used to derive re-encryption keys in the next phase. The delegation phase is done only once

for each computing cluster and involves the interaction of three entities: (ii) Dataset Owner (DO), with a pair of public and secret keys (pk_{DO}, sk_{DO}), the former used to encrypted generated data for consumption; (ii) Delegation Manager (DM), with keys (pk_{DM}, sk_{DM}), and which belongs to the security domain of the data owner, so it is assumed trusted by him; and (iii) Re-Encryption Key Generation Center (RKGC), which is local to the cluster and is responsible for generating all the re-encryption keys needed for access delegation during the consumption phase.

In order to create the master re-encryption key mrk_{DO}, which is actually $mrk_{DO} = rk_{DO \to DM} = sk_{DM} \cdot sk_{DO}^{-1}$, these three entities follow a simple three-party protocol, so no secret keys are shared, as depicted in Figure 1. The value t used during this protocol is simply a random value that is used to blind the secret key. At the end of this protocol, the RKGC possesses the master re-encryption key mrk_{DO} that later will be used for generating the rest of re-encryption keys in the consumption phase, making use of the transitive property of the proxy re-encryption scheme.

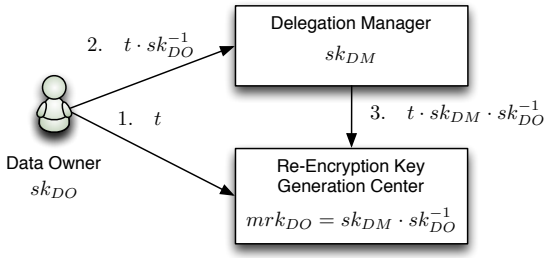

Fig. 1. Delegation protocol

C. Consumption phase

This phase is performed each time a user submits a job to the Hadoop cluster. First, a pair of public and private keys for the TaskTrackers is initialized in this step; these keys will be used later during the encryption and decryption process. For simplicity, we assume that a common pair of public and private keys (pk_{TT}, sk_{TT}) is shared by all the TaskTrackers; however, each TaskTracker could have a different pair if necessary and the process would be the same. Next, re-encryption keys for each TaskTracker are generated, in our case only one, as we assumed only one pair of public and secret keys. In this step, the Delegation Manager, the Re-Encryption Key Generation Center, the JobTracker and one of the TaskTrackers with public key pk_{TT} interact in order to generate the re-encryption key $rk_{DO \to TT}$, as depicted in Figure 2; in this case, u is the random blinding value. The final output is a re-encryption key $rk_{DO \to TT}$ held by the JobTracker, who will be the one performing re-encryptions. This process could be repeated in case that more TaskTrackers' keys are in place.

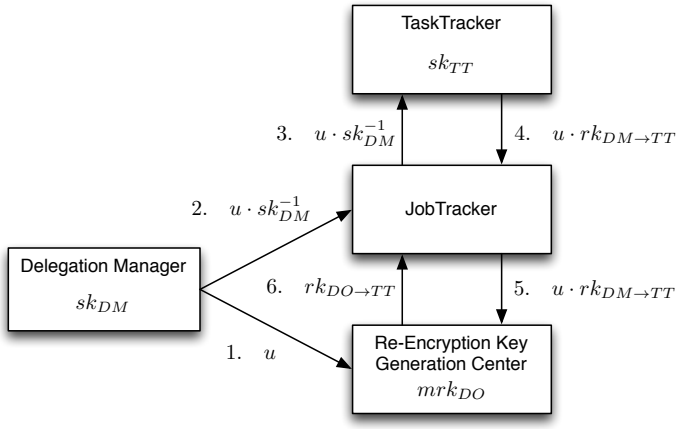

Fig. 2. Re-Encryption Key Generation protocol

Now that re-encryption keys have been generated, the JobTracker determines the input set, which is specified by the job configuration, in order to find the number of input splits. Recall that the number of map tasks depends on the number of input splits. Following Hadoop's data locality principle in order to save network bandwith, the JobTracker will select a set of TaskTrackers that are close to the input data in terms of network proximity, and will send the task requests to this set of TaskTrackers. Before each TaskTracker being able to do any processing, encrypted blocks must be deciphered. In order to do so, each TaskTracker needs to request the re-encryption of the encrypted lockbox for each block to the JobTracker. When the re-encryption is done, the JobTracker sends back the re-encrypted lockbox, which is next deciphered by the TaskTracker for extracting the symmetric key of the content. Once the block is decrypted, data is ready for being extracted by the TaskTracker. The map process now continues in the same way than in regular Hadoop: each TaskTracker invokes the map function for each record in the input split, producing a set of key-value pairs. This intermediate data is sorted and partitioned with respect to the key and stored in local files, one for each reducer. These intermediate files are also encrypted, but this time with the public key of the Reducer TaskTrackers. Since we assume that all TaskTrackers share the same pair, this key will be pk_{TT}; however, a different set of keys could be used.

When the map task finishes, its TaskTracker notifies the JobTracker about the completion, and once all the TaskTracker complete their map tasks, the JobTracker will select a set of TaskTrackers for performing the

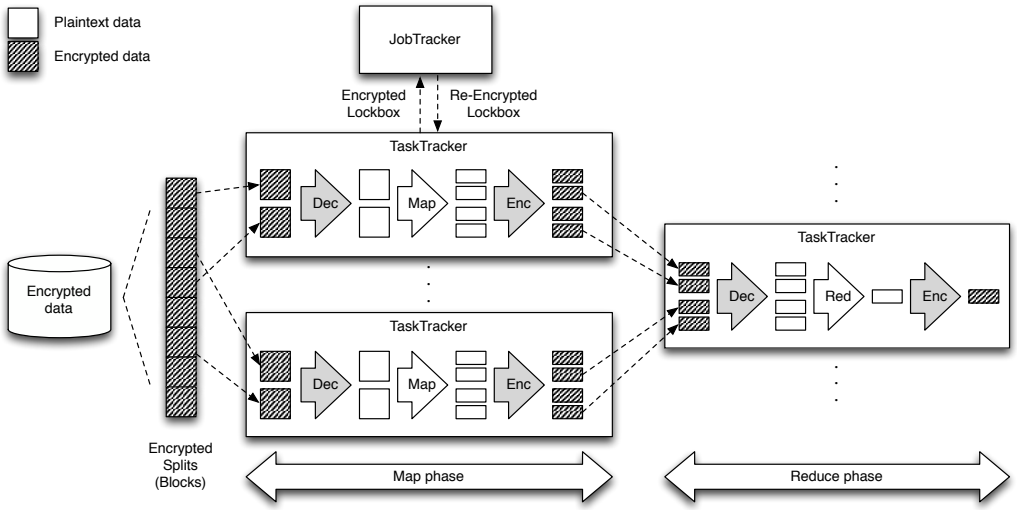

Fig. 3. Operation of our proposal during the consumption phase

Reduce tasks. Each reduce task will first read the intermediate output files remotely and decrypt them using their secret key sk_{TT}. Now that the intermediate files are in clear, they sort and merge the output files and execute the reduce function, which produces an aggregated value for each key; the results are written in one output file per reduce task. The final output can be encrypted using the public key of the client; for simplicity, we can assume that the client in this case is the data owner, so the public key is pk_{DO}. Finally, output files are stored in HDFS. The full procedure is depicted in Figure 3.

V. Experimental results

For our experiments, we have executed the main part of the consumption phase of a job, where the processing of the data occurs. From the Hadoop perspective, the other phases are offline processes, since are not related with Hadoop's flow. Our experiments are executed in a virtualized environment on a rack of IBM BladeCenter HS23 servers connected through 10 gigabit Ethernet, running VMware ESXi 5.1.0. Each of the blade servers is equipped with two quad-core Intel(R) Xeon(R) CPU E5-2680 @ 2.70GHz. We set up a cluster of 17 VMs (the master node, which contains the JobTracker and the NameNode, and 16 slave nodes, each of them holding a TaskTracker and a DataNode). Each of the VMs in this environment is provided with two logical cores and 4 GB of RAM, running a modified version of Hadoop 1.2.1 that implements a prototype of our proposal. As for the cryptographic details, the proxy re-encryption scheme is implemented using elliptic curve cryptography over a prime field. In particular, we implemented the proxy re-encryption scheme from Weng et al. using the NIST P-256 curve, which provides 128 bits of security and is therefore appropriate for encapsulating 128 bits symmetric keys [9]. With respect to the symmetric encryption algorithm we chose AES-128-CBC. We will also make use of the built-in support for AES included in some Intel processors through the AES-NI instruction set.

The experiment consisted on the execution of one of the sample programs included in Hadoop, the WordCount benchmark, a simple application that counts the occurence of words over a set of files. In the case of our experiment, the job input was a set of 1800 encrypted files of 64 MB each; in total, the input contains 28.8 billions of words and occupies aproximately 112.5 GB. The size of each input file is sligthly below 64 MB, in order to fit HDFS blocks.

We executed two runs over the same input: the first one using a clean version of Hadoop and the second one using a modified version with a prototype of our proposal. The total running time of the experiment was 1932.09 and 1960.74 seconds, respectively. That is, a difference of 28.74 seconds, which represents a relative overhead of 1.49% for this experiment. Table I shows the measured time cost associated to the main cryptographic operations of our solution.

The most critical part of the execution is at the beginning of each Map task, when for each encrypted split the TaskTracker asks for the corresponding re-encrypted lockbox to the JobTracker, decrypts it and performs a symmetric decryption of the data block. The duration of this process is more or less constant, as it mostly depends on the size of the encrypted data block. Thus, relative overhead will depend drastically on the duration of the map phase. On the one hand, if the map phase is very

TABLE I
TIME COST FOR THE MAIN CRYPTOGRAPHIC OPERATIONS

Operation	Time (ms)
Block Encryption (AES-128, 64 MB)	214.62
Block Decryption (AES-128, 64 MB)	116.81
Lockbox Encryption (PRE scheme)	17.84
Lockbox Re-Encryption (PRE scheme)	17.59
Lockbox Decryption (PRE scheme)	11.66

intensive, then the overhead introduced by our solution will be relative small, in comparison with the processing of the input splits. On the other hand, if the map phase is light, then the overhead will be very significant. In the case of our experiment, where the processing of input splits in each map task takes approximately 34 seconds, the overhead introduced by our solution is very small, as the duration of the cryptographic operations is within the order of milliseconds.

VI. Related work

The integration of encryption technologies in Hadoop is a topic that is being explored recently. Park and Lee present in [10] a modification of the Hadoop architecture in order to integrate symmetric encryption in HDFS. They also perform an experimental evaluation of their solution and claim that the overhead is less than 7%. However, as they only consider the use of symmetric encryption, the secret keys used for encrypting have to be shared with the computing nodes. In a similar work, Lin et al [11] show the impact of integrating RSA and pairing-based encryption on HDFS. In their experiments, performance is affected by between 20% and 180%. In other related areas, such as cloud computing, the use of cryptography for enforcing access control and protecting data confidentiality is a hot topic [12]. In particular, the use of proxy re-encryption for constructing cryptographically-enforced access control systems has already been explored in other works [7], [13].

VII. Conclusions

In this paper we address the problem of how to integrate data confidentiality and massive processing in the Big Data scenario; widely accepted solutions, such as Hadoop, do not offer the proper means to protect data at rest. Security issues are even more worrisome in multi-tenant environments, such as when Hadoop is executed in the cloud and provided as a service. For these reasons, there is an imperative need for technical safeguards using cryptography, beyond simple access control layers.

We propose a cryptographically-enforced access control system for Hadoop, based on proxy re-encryption. In our solution, stored data is always encrypted and encryption keys do not need to be shared between different data sources. Nevertheless, the use of proxy re-encryption allows stored data to be re-encrypted into ciphered data that the cluster nodes can decrypt with their own keys when a job is submitted. In order to permit this, the data owner has to first grant access rights (in the form of decryption capabilities) to the Hadoop cluster.

Experimental results show that the overhead produced by the encryption and decryption operations is manageable, so our proposal is suitable for some applications. In particular, the main insight we extract from this experiment is that our proposal will fit well in applications with an intensive map phase, since then the overhead introduced by the use of cryptography will be reduced.

Acknowledgments

This work was partly supported by the Junta de Andalucía through the projects FISICCO (P11-TIC-07223) and PISCIS (P10-TIC-06334). The first author has been funded by a FPI fellowship from the Junta de Andalucía.

References

[1] Expanded Top Ten Big Data Security and Privacy Challenges. Technical report, Cloud Security Alliance, 2013.
[2] Apache Hadoop. http://hadoop.apache.org/.
[3] Google Big Query. https://cloud.google.com/products/bigquery/.
[4] Hadoop on Google Cloud Platform. https://cloud.google.com/solutions/hadoop/.
[5] MapR in the Cloud. http://www.mapr.com/products/mapr-cloud.
[6] Devaraj Das, Owen O'Malley, Sanjay Radia, and Kan Zhang. Adding Security to Apache Hadoop. Technical Report 1, Hortonworks, 2011.
[7] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.
[8] Jian Weng, Robert H Deng, Shengli Liu, and Kefei Chen. Chosen-ciphertext secure bidirectional proxy re-encryption schemes without pairings. *Information Sciences*, 180(24):5077–5089, 2010.
[9] E. Barker, L. Chen, A. Roginsky, and M. Smid. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST special publication 800-56A (Revision 2), NIST, May 2013.
[10] Seonyoung Park and Youngseok Lee. Secure Hadoop with Encrypted HDFS. In *Grid and Pervasive Computing*, pages 134–141. Springer, 2013.
[11] Hsiao-Ying Lin, Shiuan-Tzuo Shen, Wen-Guey Tzeng, and B.-S.P. Lin. Toward data confidentiality via integrating hybrid encryption schemes and Hadoop distributed file system. In *Advanced Information Networking and Applications (AINA), IEEE 26th International Conference on*, pages 740–747, 2012.
[12] S. Kamara and K. Lauter. Cryptographic cloud storage. *Financial Cryptography and Data Security*, pages 136–149, 2010.
[13] David Nuñez and Isaac Agudo. BlindIdM: A privacy-preserving approach for identity management as a service. *International Journal of Information Security*, 13(2):199–215, 2014.