Data Security and access management in Cloud Computing: capability list-based cryptography

Khalid Aissaoui, Hicham Belhadaoui, Abdelouahed Zakari, Mounir Rifi

Data Security and access management in Cloud Computing: capability list-based cryptography

Khalid Aissaoui*, Hicham Belhadaoui*, Abdelouahed Zakari#, Mounir Rifi*

*CED Engineering Science, ENSEM, Lab. RITM/ESTC Hassan II University Casablanca, Morocco
#LITA University of LORRAINE Metz, France
aissaoui.khalid@gmail.com

***Abstract*--A new paradigm in information technology has emerged this last years and continues to grow. Cloud computing is quite a new concept that has potential advantages; computing capacity provided as needed, much lower cost than in-house infrastructure, better reliability depending on the Service Level Agreement (SLA)…**
However, some drawbacks, mainly data security and access control concerns, remain a real setback to a wider utilization of this technology. In fact, storing sensitive data in a remote server owned by an external entity (the cloud provider) could lead to substantial risks. A malicious user or operator can identify and exploit the vulnerabilities of this system.
Numerous works are being done in order to reinforce the cloud capacities in term of protecting data and managing access control using cryptography, data fragmentation, access control policies...
In this paper, we introduce a new approach in which we exclude the cloud provider from any involvement in the access management with the aim of minimizing the leaks. We developed and tested programs based on a capability-list and using both symmetric and asymmetric cryptography.

Keywords: Cloud computing security; cryptograph; Hadoop DFS; confidentiality; virtualization; access management.

I. INTRODUCTION

During the last few years, new information technologies have made considerable progress; internet became widely used in all domains, offering larger bandwidth and bigger storage capacity. Consequently, a new IS management model was advanced.

According to the NIST "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction…" [1].

Cloud Computing offers many great advantages. The same block of data can be stored separately in more than one server; this provides best availability such as Back up. It provides enhanced and simplified IT management and maintenance capabilities through central administration of resources. Besides, one pays only for the resources that he actually uses (pay as you go) without need for more expenses; there is no invested IT capital. By adopting this new technology, companies, organizations and other entities reduce enormously their costs (purchasing new devices, hiring more engineers, computer maintenance). Finally, data stored on cloud provider's servers can be reached from anywhere at any moment using a simple internet connection. Three different models can be deployed on a provider's servers; a public, a private and a hybrid one [2].

According to Jansen and Grance [3], Cloud providers offer diversified services depending on users requirements.

- Software as a Service (SaaS): Users get access to applications and software using a web browser. SaaS enables providers to control and manage the use of their products.
- Platform as a Service (PaaS): In this model, providers offer a development environment (IDE, toolkits, DBMS…) to developers in order to design, develop and deploy their own solutions.
- Infrastructure as a Service (IaaS): This model provides the infrastructure to run customer's applications; it enables organizations and companies to exclusively use hardware (Servers, routers, connectors…).

In this paper, we aim to describe a novel approach to protect and secure data in Cloud environment; hence we exclude providers from participating in the access management. Only the data owner can manage and update the access rights. As a consequence, providers only offer an IaaS. To explain our concept, this paper is divided as follows: in Section II, we enumerate security problems that face cloud computing. In Section III and IV, we present, according the state of art, some solutions to

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

secure outsourced data. In Section V, we describe an implementation of our model, its features and advantages. Finally, we conclude with some perspectives of evolution.

II. SECURITY IN CLOUD COMPUTING

Cloud Computing is being used increasingly due to its significant advantages. However, there are some concerns about data security that require more attention and need to be resolved. Within this new paradigm, traditional means of protection are deprecated. Important data like financial information, scientific formula, industrial secrets or trading information are moved to remote shared servers managed by a third party who must guarantee security as agreed with clients (data owner or users).

In a cloud environment, sensitive data is stored on shared remote servers assigned by a service provider who has to guarantee confidentiality. Access to this data is forbidden to persons or groups not allowed including the provider. Some techniques are used, like cryptography (data is encrypted before being sent to external servers), or fragmentation which is an interesting alternative too (columns of the relational data table, which together are more sensitive, are stored separately on different nodes). Protecting data queries is also a major issue. When data is encrypted and stored on external servers, users send queries to the provider with the aim of getting the data they need. The number, the type and the date of those requests may be confidential as well, one can deduce sensitive information. Besides, authentication is an important step which allows the server to identify any user in order to enable access to data. Different methods can be implemented to strengthen this process (passwords, strong and other challenge-response authentication). Furthermore, to guarantee data integrity, the owner has to protect his/her files from modification or deletion that may occur (accidentally or deliberately). Users who need access to stored data in cloud environment have different profiles and roles; they are employees, customers or even suppliers. Hence, the system needs an identity access management model to give the right data to the right user (confidentiality). Finally, the non-repudiation provides proof of the integrity and origin of the data. This prevents the user from denying actions performed (purchasing on the net, sending messages…).

III. RELATED WORKS:

Protecting and managing access control to outsourced data has been the main issue in some research papers. Ateniese et al [4] propose a proxy re-encryption; the owner encrypts with symmetric content keys blocks of data before sending it to cloud servers, those content keys are encrypted with the owner's master public key. His master private key and users public keys are then combined to generate proxy re-encryption keys which are used to recover plain text intended to a specific user. In this model, any collaboration between the services provider and the user would compromise data security by revealing decryption keys. In their paper, Naor and al [5] use a mechanism of generation and distribution of symmetric keys based on Blom's work [6].

Yu et al [7] propose an encryption based on attributes assigned to data. Every single file is associated to an attribute. The access structure of each user (defining his authorized files) is created according to a logical expression over these attributes. Each file is encrypted with the public key corresponding to his attribute. The user is not able to decrypt a file if its attribute does not match his access structure. In this scheme, the computation of logical expressions becomes more complex as the quantity of data stored grows. Besides, the updating of the access structure after access rights modification or revocation is a real burden as the number of users is extremely high in a cloud environment.

Miklau et al [8] are interested in access control on XML documents by encrypting different portions of an XML tree with different keys and using metadata nodes.

Li et al [9], for their part, developed an access control system also based on Attribute-Based Encryption. Nevertheless, their scheme is based on delegating Key generation and decryption to two different Cloud Service Providers in order to outsource heavy computation. However, this model requires, in addition to the client and final users, two more actors. This presents a risk of colluding.

For securing outsourced data [10], Vermacati and al propose a solution based on a key derivation system. Files are encrypted with symmetric keys; a private key is assigned to each file. A user can only decrypt the files he is allowed to, by using both his private key and a set of public tokens that the owner generates and sends to the server which is responsible for distribution. In this model the complexity of all these operations is linear to the number of users.

Hota and al propose a solution [11] in which the data owner shares with the provider a list of access rights (capability list). It is constantly updated in case of modification or insertion. The provider uses the capability list to manage access requests. Data is encrypted before being sent to the servers with a symmetric key (shared with appropriate users). This approach has some security issues revealed by Gao et al in their paper [12], particularly, Repeat and Man In The Middle attacks. However, the Cloud Service Provider participates directly in the access control which requires an additional encryption layer to secure the owner's data.

Another issue related to outsourced data security is to delegate the ability to process data to the provider. In fact, with encrypted data stored in his servers, the Cloud Service Provider can only handle

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

the storing and ultimately the access. No operation can be performed on data. This is a limitation that Gentry [13] tried to overcome by developing a Fully Homomorphic Encryption (FHE). This technique allows the provider to run programs on ciphertext and obtain results that completely match with plaintext's results. However, executing a program with a FHE produces an encryption of the result, not the result itself. The provider cannot, therefore, make decisions based on the outputs.

In the next section, we propose a new scheme using the capability list mentioned above, but the provider is totally excluded from participation in securing data and access control.

IV. THE PROPOSED APPROACH

A. *Hota's model*

First, we explain here Hota's model on which we based our solution. There are three actors; Data Owner (**DO**), Cloud Service Provider (**CSP**) and User. "Fig. 1" shows the architecture used.

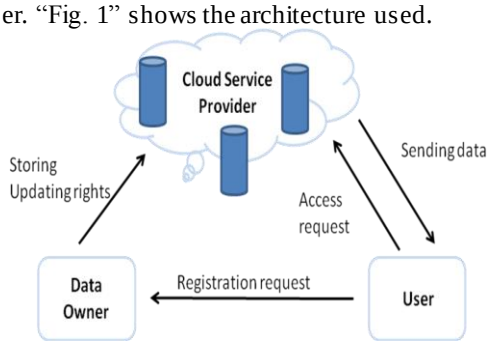

Figure 1. The system architecture.

The model is based on three main algorithms. First, DO stores encrypted data and the capability list (which describes users' access rights) on CSP servers. The second algorithm explains how User sends a registration request. Finally, CSP provides, on demand and after checking the capability list, encrypted data to appropriate users along with the decryption key. To achieve their solution they assume that public keys are available for all participants, without using any Public Key Infrastructure (PKI). Besides, DO does not have to be permanently connected. *Table 1* shows the notations used in this model.

TABLE I. NOTATIONS USED IN HOTA'S SCHEME

Notation	Description
PUSP	CSP public key
PRSP	CSP private key
PUUSR	User public key
PRUSR	User private key
PUOWN	DO public key
PROWN	DO private key
Fi	ith File
Ko	DO symmetric key
MD5	Hash Algorithm
CapList	Capability List (UID, FID, AR)
AR	Access Right (0 r, 1 w, 2 r/w)
UID	User Identity
FID	File Identity

First, the DO sends both encrypted data and the Capability List (UID, FID and AR) "Fig. 2". The ith file and its message digest (generated with a hash function) are encrypted with DO's symmetric key (Ko), and then a signature with PROWN is performed to proof sender's identity. Finally, the package is encrypted with PUSP for confidentiality purpose. The CapList is also sent after a signature with PROWN and an encryption with PUSP. We note that the function insert allows for adding a new line to the CapList related to the file uploaded (UID, FID and AR).

$DO \rightarrow CSP: ENC_{PUSP}\{SIG_{PROWN}\{ENC_{K0}\{|Hash(fi),fi|\}\}\}$
$DO \rightarrow CSP: ENC_{PUSP}\{SIG_{PROWN}\{insert(CapList, UID, FID, AR)\}\}$

Figure 2. Sending encrypted data and capability list to CSP.

On the other hand, the User needs to send a registration request to DO containing his UID, a FID and the access rights AR "Fig. 3". DO updates the CapList and sends it with all decryption parameters to CSP (those parameters are necessarily encrypted with the User's public key) "Fig. 4".

$USER \rightarrow DO : ENC_{PUOWN}\{SIG_{PRUSR}\{UID.FID.AR.N1.TimeStamp\}\}$

Figure 3. Sending registration request to the DO.

$DO \rightarrow CSP : ENC_{PUSP}\{SIG_{PROWN}\{ENC_{PUUSR}\{|K0, Hash, TimeStamp, N1+1|\}\}\}$
$DO \rightarrow CSP : ENC_{PUSP}\{insert(CapList, UID, FID, AR)\}$

Figure 4. Updating and Sending the CapList and decryption parameters to CSP.

CSP updates the CapList stored on his servers and sends decryption parameters to User "Fig. 5". To get access to stored data, User sends a request to CSP who verifies User's identity and access rights before authorizing.

$CSP \rightarrow USER: ENC_{PUUSR}\{SIG_{PROWN}\{ENC_{PUUSR}\{|K0, Hash, TimeStamp, N1+1|\}\}\}$

Figure 5. Sending encrypted data and decryption parameters to USER.

B. *The revised model*

As described above in the scheme proposed in [11], the CSP participates actively in the data access control, since it verifies the CapList and sends data to users. This requires an additional encryption layer (public and private key, session key to secure communication with users). Furthermore, the capability list, which is shared with CSP in plain form, represents a risky source of information (requests flow, number of users…). Finally, a

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

malicious collaboration between CSP and any user (e.g. key disclosure) would bring a considerable damage to the systemsecurity.

In this paper, we propose to remove CSP from the architecture, "Fig. 6" so we can avoid any exchange between the provider and users. We have implemented the model proposed below on a local platform. In the following paragraph we explain its technical aspect. The new approach is based on the development and integration of programs (discussed below) on servers in order to address the CapList management and respond to access requests.

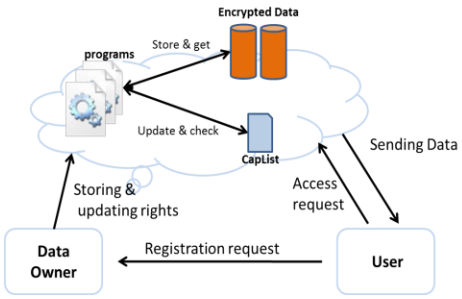

Figure 6. The novel architecture

We assume that all public keys used are available to participants, PKI is not the concern. Besides, DO and users are enabled to execute programs on remote servers provided by CSP. The execute rights of the programs will depend on the use case (User or/and DO).

In order to provide a cloud computing environment, we set up a cluster with 3 virtual machines that host a Linux OS using the XenServer hypervisor. Then, we install the Apache Hadoop framework.

Hadoop is an open-source software platform based on Google File System [14] that provides a massive parallel processing and storage tools using commodity hardware. The complexity of distributed processing is handled by Hadoop. This simplifies developers' work and makes them concentrate on their programs. The storage part is handled by HDFS (Hadoop Distributed File System) which is responsible for storing blocks of data in a distributed way [15].

V. IMPLEMENTATION

In order to implement our model, we developed a web service in Java and exported some programs as jar files to the cloud so it can be executed on a Hadoop environment. There are five main algorithms:

A. Publish Data

This code encrypts and uploads sensitive Data to the Cloud. It also updates the capability list on DO's local machine. Only the DO can execute this program "Fig. 7(a, b &c)".

DO → Cloud: ENC_{K0} {zip(Hash(fi),fi)}

Figure 7(a): Sending encrypted data along with its message digest to the Cloud.

Algorithm 1 PUBLISH DATA
Input: fi **output**: Ko, dfi
1. Ko ← generate()
2. dfi ← Hash(fi)
3. send(encrypt(zip(fi,dfi), Ko))
4. insert(CapList, dfi)

Figure 7(b): Publish data - algorithm

Ko: is a symmetric key which is generated by the program and used to encrypt both the file and its message digest as explained before.
dfi: The file's messages digest.
Hash: is obviously a function that calculates a message digest using a hash algorithm.
zip: this function creates a zip archive that contains the file and its message digest.
Insert: this function allows adding the file's message digest to the capability list.

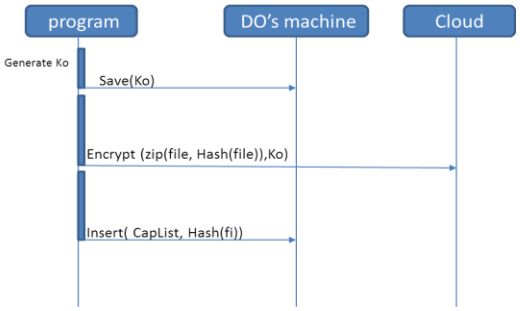

Figure 7(c): Publish data - Sequence diagram

B. Registration

This step allows User to send a registration request to cloud in order to get access to encrypted data. To do so, he has to specify the file he is interested in, his identity and the access rights.

USER → Cloud: zip (SIG_{PRUSR}{ Hash (UID.Filename.AR) }, ENC_{Ks}{UID.Filename.AR}), ENC_{PUOWN}{Ks}

Figure 8(a): Sending registration request to the Cloud

Algorithm 2 REGISTRATION REQUEST
Input:UID, filename, AR **output**: Ks
1. Ks ← generate()
2. req ← concat(UID, Filename, AR)
3. Ereq ← encrypt(req, Ks)
4. Dreq ← Hash(req)
5. Sreq ← Sign(Dreq, PRUSR)
6. EKs ← encrypt(Ks, PUOWN)
7. send(zip(Sreq,Ereq),EKs)

Figure 8(b): Registration request - Algorithm

Ks: a session key used to secure communication. An encrypted version of Ks is naturally sent to Cloud.

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

req: a request string containing the user's id, the file name which the user is interested on and the intended access right.
Ereq: the encrypted form of req (with Ks).
Dreq: the req's message digest.
Sreq: the signature of Dreq with PRUSR.
EKs: the encrypted form of Ks (with PUOWN).

This algorithm uploads two files to the cloud, an encrypted request along with its message digest for integrity check purpose and a session key to decrypt the request.

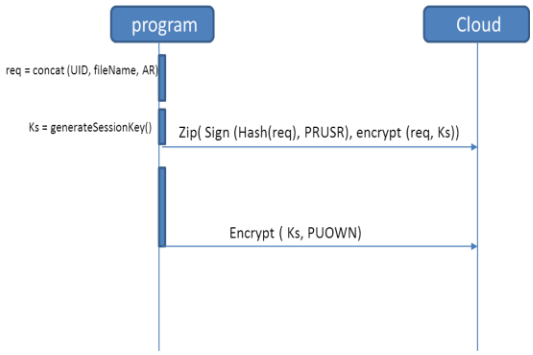

Figure 8(c): Registration request – Sequence diagram

C. *Response*

This bloc program addresses User's request by allowing or denying access to data. Response's program needs as a parameter the User's login or Id. This action is clearly performed by the data owner.

Algorithm 3 RESPONSE
Input: UID **output**: Dreq2, EKo
1. Ks ← decrypt(EKs, PROWN)
2. {Ereq, Dreq} ← unzip(Zreqfile)
3. req ← decrypt(Ereq, Ks)
4. Dreq2 ← Hash(req)
5. if Dreq = Dreq2 then
   Update(capList)
   EKo ← encrypt(Ko, PUUSR)
   Send(EKo)
   End if

Figure 9(a): Response – Algorithm

Zreqfile: a zipped file that contains both the encrypted form of the request and its message digest.
Dreq: is a message digest that calculate the DO in order to verify the integrity of the user's request.
EKo: Encrypted form of Ko

Before the DO allows or denies access to data, the program compares the request's message digest retrieved from the Cloud with a new calculated one (Dreq2). If there is a matching, the granting process (updating capability list on local machine and sending symmetric key Ko to User) can begin, if not, the program terminates.

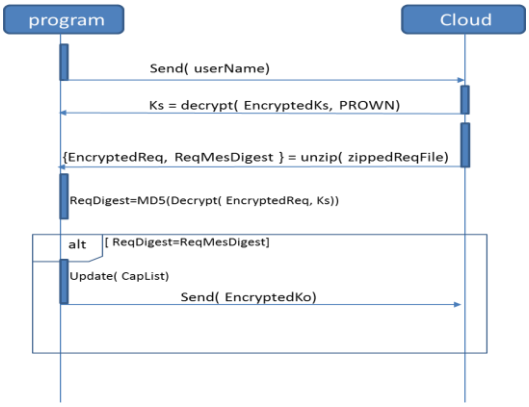

Figure 9(a): Response – Sequence diagram

D. *Load capability list*

In this bloc program, also executed by DO, the capability list and its signed message digest (with DO private key PROWN) are uploaded to cloud.

$$DO \rightarrow Cloud: SIG_{PROWN}\{Hash(CapList)\}, CapList$$

Figure 10(a): Sending the CapList and its message digest to the Cloud

Algorithm 1 LOAD CAPABILITY LIST
Input: CapList **output**: DCList, SDCList
1. DCList ← Hash(CapList)
2. SDCList ← Sign(DCList, PROWN)
3. send(CapList, SDCList)

Figure 10(b): Sending the CapList – Algortihm

DCList: a message digest of the CapList
SDCList: a signed message digest with the DO's private key PROWN

Consequently, the user can send an access request by executing the Access Data program which uses the updated the CapList to check the access rights.

E. *Access Data*

Finally, the User executes the last bloc program, enters his identity and the encrypted file name to ask access. He downloads the encrypted file along with the symmetric key if he's allowed to (according to the capability list).

Algorithm 1 ACCESS DATA
Input: CapList, SDCList, EO **output**:
1. DCList2 ← Hash (CapList)
2. DCList ← decrypt (SDCList, PUOWN)
3. If DCList = DCList2 then
   If verify (CapList)=ok then
   Ko ← decrypt (EKo, PRUSR)
   O ← decrypt (EO, Ko)
   {fi, dfi}← unzip (O)
   If Hash (fi) = dfi then
   //task successfully completed
   End if
   End if
   End if

Figure 11: Access data – Algortihm

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

DCList2: a message digest of the CapList used to verify the integrity of the CapList.
EO: an encrypted object containing both the requested file and its message digest
verify(CapList): is a function that checks the user access rights upon the noticed file.

The process is obviously aborted if any integrity test returns a false result or if the User does not have the right access to the encrypted file. To run our bloc programs on Hadoop, we have to export a jar file of each source file to Hadoop folder. A user can execute a jar file only if he has a read right on it.

F. Execution tests

In order to simulate the proposed model, we created a group of users in addition to a data owner profile. We also developed a service web to run our tests.

First, we need to upload an encrypted data to the Cloud. In order to do so, the DO must authenticate. The file name is naturally specified in the browser.

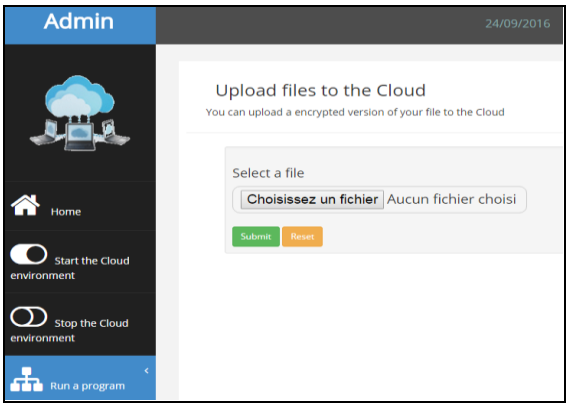

Figure 12: Upload file to Cloud

The file is sent to the Cloud as a result of which a line is added to the CapList. "Fig. 13" shows all the files encrypted and stored in a HDFS folder named "*EncryptedData*".

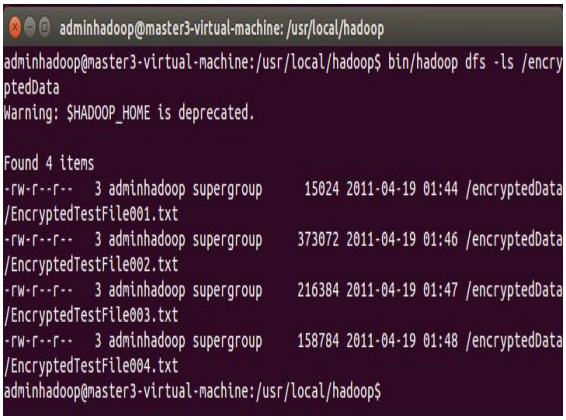

Figure 13: encryptedData folder in Hadoop environment

Henceforward, a final user can send a registration request in which he specifies the file name to which he requires access. Systematically all symmetric keys which DO used to encrypt files are encrypted and saved into his local machine.

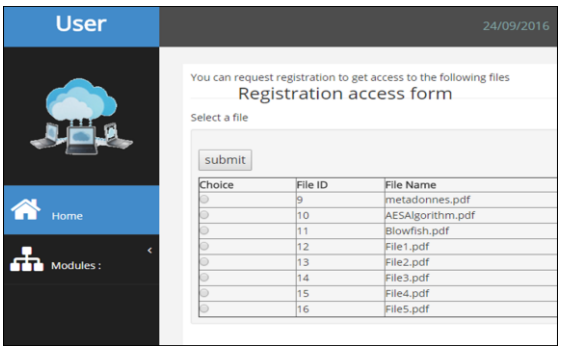

Figure 14: Registration request from a user

Request details are edited in a file and stored into a HDFS folder */requests* so that DO can address it through his account page.

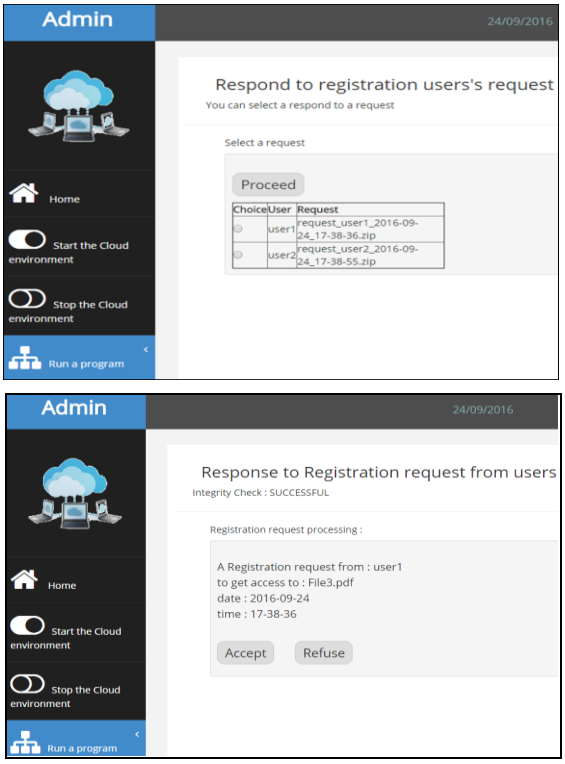

Figure 15: Response to users' Registration request

Finally, if the access to the specified file is authorized according to the updated capability list then the user will receive both the encrypted file and the symmetric key (which is encrypted with his public key) to perform the decryption in a secure way.

VI. EVALUATION

Three aspects of the data security are guaranteed in this model. Data integrity; the hash function algorithm is used to calculate message digest by DO. The User does the same and compares. Since there is very strong algorithms like SHA-2 and specially SHA-3 which creates up to 512 bits output word. The use of asymmetric cryptography is beneficial in two ways; namely, confidentiality is being secured with public key and signature with private key. Besides, authentication is an important

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

step which strengthens security. It is essential that the system avoids any intrusion to servers by an unauthorized entity, even if data is already encrypted. Hadoop provides an efficient platform to run programs in a distributed environment. However, it shows some weaknesses in terms of authentication. Although we have put in place an authentication step in our process, an intrusion can occur with ease. A strong authentication would be a good answer to this issue.

On the other side, we conducted some performance tests related to execution duration of both encryption and decryption based on files' sizes. The results show that up to 64 MB the duration of encryption and decryption slightly exceeds 1 second which may be tolerable from users' point of view. The system's behavior in regard to data size is important since Hadoop environment is specially used for big data storing purpose. "Fig. 16" and "Fig. 17" show the results.

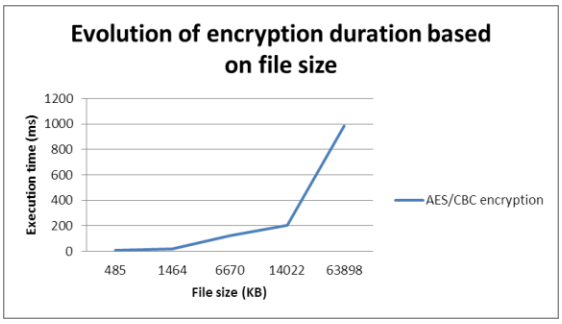

Figure 16: Encryption duration based on file size

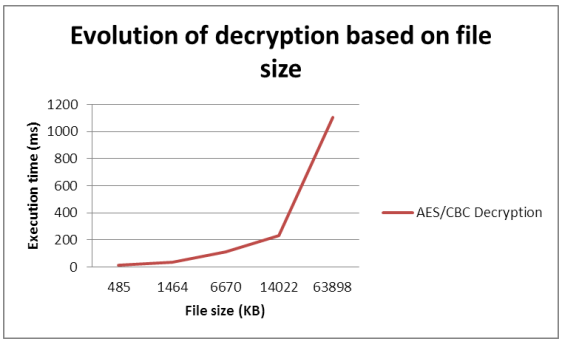

Figure 17: Decryption duration based on file size

More detailed comparison between the most used cryptographic algorithms has been conducted in specific studies [16], [17].

VII. CONCLUSION

In this article, we provide an overview of the literature regarding the protection of the outsourced data in a cloud environment, then we propose and implement a novel approach using a web service and a Hadoop environment with the aim of improving the system security. It seems that the revised model has a number of advantages. As cited above, it improves security of outsourced data since contrary to some other solutions the cloud provider does not participate in the process. On the other hand, using Hadoop allows disburdening bloc programs of distributed programming.

However, some concerns remain to be addressed such as enhancing authentication, revoking rights, deleting and clearing data from cloud provider servers. Furthermore, in our model, the DO has to be connected to the system in order to update the access rights and respond to users' requests witch can be a real constraint.

To this effect, we aim to refine our model by integrating a strong authentication mechanism, disburdening the owner from the access management by getting the model more self-contained and addressing the data remanence issue.

REFERENCES

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing. NIST Special Publication 800-145 (Draft)", Retrieved September 10, 2011, from http://csrc.nist.gov/publications/drafts/800-145/Draft-SP- 800-145 cloud definition.pdf

[2] T Mather, S Kumaraswamy, S Latif, Cloud Security and privacy, O'REILLY, September 2009

[3] W Jansen, T Grance, Guidelines on Security and Privacy in Public Cloud Computing, NIST (National Institut of Standards and Technology) U.S. Department of Commerce, December 2011

[4] G. Ateniese, K. Fu, M. Green and S. Hohenberger, Improved Proxy Re-Encryption Schemes with Application to Secure Distributed Storage, ACM Transactions on Information and System Security, Vol. 9 No. 1, Feb 2006 pp. 1-30.

[5] Dalit Naor, A. Shenhav and A. Wool, Toward Securing Untrusted Storage Without Public-key Operations, Proc. 2005 ACM Workshop on Storage Security and Survivability (StorageSS), Virginia, USA, Nov 2005, pp. 51-56.G.

[6] R. Blom, An Optimal Class of Symmetric Key Generation Systems, Proc. EUROCRYPT 84 Workshop on Advances Cryptology: Theory and Application of Cryptographic Techniques, Springer Verlag, NY, USA, 1985, pp. 335-338.

[7] Shucheng Yu, Cong Wang, Kui Ren and Wenjing Lou, Achieving Secure, Scalable and Fine-grained Data Access Control in Cloud Computing, Proc. ACM Workshop on Computer Security Architecture (CSAW'07), Nov 2007, USA.

[8] Miklau and D. Suciu, Controlling Access to Published Data Using Cryptography, Proc. 29th VLDB, Germany, Sept 2003, pp. 898-909.C.

[9] L. Jin, C. Xiaofeng, L. Jingwei, J. Chunfu, M. Jianfeng, L. Wenjing, New access control system based on outsourced attribute-based encryption. Journal of Computer Security, vol. 23, no. 6, pp. 659-683, 2015

[10] S. D. C. di Vimercati, S; Foresti, S Jajodia, S Parabocshi and P. Samarati, Over-encryption : Management of Access Control Evolution on Outsourced Data, Proc. 33th International Conference on Very Large Databases (VLDB'07), Vienna, Austria, 2007, pp. 123-134.X.

[11] Hota, S. Sanka, M. Rajarajan, S. K. Nair, Capability-Based Cryptographic Data Access Control in Cloud Computing. Int J. Advanced Networking and Applications 03, 1152-1161 (2011)

[12] Gao, Z. Jiang, R. Jiang, A Novel Data Access Scheme in Cloud Computing, International Conference on Computer and Information Application (ICCIA 2012)

https://sites.google.com/site/ijcsis/
ISSN 1947-5500

[13] C Gentry, Fully Homomorphic Encryption Using Ideal Latices, Symposium on the Theory of Computing (STOC), 2009, pp. 169-178

[14] S Ghemawat, H Gobioff, and S Leung, the Google File System. SOSP'03, October 19–22, 2003, Bolton Landing, New York, USA. Copyright 2003 ACM 1-58113-757-5/03/0010

[15] T White, Hadoop the definite guide, O'reilly

[16] O P Verma, R Agarwal, D Dafouti, S Tyagi, Performance analysis of data encryption algorithms, Electronics Computer Technology (ICECT), 2011 3rd International Conference

[17] J Thakur, N Kumar, DES, AES and Blowfish: Symmetric Key Cryptography Algorithms Simulation Based Performance Analysis. International Journal of Emerging Technology and Advanced Engineering (ISSN 2250-2459, Volume 1, Issue 2, December 2011)

AUTHORS PROFILE

Khalid AISSAOUI currently a PhD student, University Hassan II /ENSEM Casablanca Morocco. Received an engineering degree in 2008 at ENSSAT-Lannion France. Research: Security in Cloud Computing
Working as professor at higher institute of applied engineering IGA, Casablanca Morocco

Hicham BELHADAOUI currently working as a Professor Ability in University Hassan II /ESTC, Casablanca Morocco. Received his Phd degree at the National Polytechnic Institute of Lorraine/France.
Research: Reliability, Automatic Signal Processing and Computer Engineering.

Mounir RIFI currently working as a Professor in University Hassan II /ESTC, Casablanca Morocco.
Obtained his PhD Physical Sciences: ElectroMagnetic Compatibility, October 1996 (University Mohamed V of Rabat - Morocco) and PhD in Electronics, May 1987 (University of Lille - France)
Director of the Research Laboratory: RITM (Networks, Computer, Telecom and Multimedia)
Research: Propagation of electromagnetic waves, ElectroMagnetic Compatibility, RFID, Microwave, Transmission Lines Theory, Antennas, Sensors, Networks.

Abdelouahed ZAKARI is Associate Professor at the University of Lorraine, LITA laboratory, Metz, France. He received his Ph.D degree (1984), in Computer Sciences from the Nancy-I University, France.
Research: first in Architecture Description Language, as he worked at the DAST Renault, Paris (Nancy1 university-industry collaboration). He worked after at the University of Metz and he defined, for CAD/CAM systms, a Multiple Modelling concept based on AI/OOP concepts and tools. His current main research interests concern the applications of complex systems, mainly: (1) modelling and

https://sites.google.com/site/ijcsis/
ISSN 1947-5500