A Flexible Mechanism for Data Confidentiality in Cloud Database Scenarios

Eliseu C. Branco Jr.[1], Jose Maria Monteiro[2], Roney Reis[2] and Javam C. Machado[2]
[1]*Computer Networks Course, University Center Estacio of Ceara, Fortaleza, Brazil*
[2]*Department of Computer Science, Federal University of Ceara, Fortaleza, Brazil*

Keywords: Data Confidentiality, Cloud Database, Information Decomposition.

Abstract: Cloud computing is a recent trend of technology that aims to provide unlimited, on-demand, elastic computing and data storage resources. In this context, cloud services decrease the need for local data storage and the infrastructure costs. However, hosting confidential data at a cloud storage service requires the transfer of control of the data to a semi-trusted external provider. Therefore, data confidentiality is the top concern from the cloud issues list. Recently, three main approaches have been introduced to ensure data confidentiality in cloud services: data encryption; combination of encryption and fragmentation; and fragmentation. In this paper, we present i-OBJECT, a new approach to preserve data confidentiality in cloud services. The proposed mechanism uses information decomposition to split data into unrecognizable parts and store them in different cloud service providers. Besides, i-OBJECT is a flexible mechanism since it can be used alone or together with other previously approaches in order to increase the data confidentiality level. Thus, a user may trade performance or data utility for a potential increase in the degree of data confidentiality. Experimental results show the potential efficiency of the proposed approach.

1 INTRODUCTION

Cloud Computing moves the application software and databases to large data centers, where data management may not be sufficiently trustworthy. Cloud storage is an increasingly popular class of services for archiving, backup and sharing data. There is an important cost-benefit relation for individuals and small organizations in storing their data using cloud storage services and delegating to them the responsibility of data storage and management (Ciriani et al., 2009). Despite the big business and technical advantages of the cloud storage services, the data confidentiality concern has been one of the major hurdles preventing its widespread adoption.

The concept of privacy varies widely among countries, cultures and jurisdictions. So, a concise definition is elusive if not impossible (Clarke, 1999). For the purposes of this discussion, privacy is "the claim of individuals, groups or institutions to determine for themselves when, how and to what extent the information about them is communicated to others" (Camenisch et al., 2011). Privacy protects access to the person, whereas confidentiality protects access to the data. So, confidentiality is the assurance that certain information that may include a subject's identity, health, lifestyle information or a sponsor's proprietary information would not be disclosed without permission from the subject (or sponsor). When dealing with cloud environments, confidentiality implies that a customer's data and computation tasks are to be kept confidential from both the cloud provider and other customers (Zhifeng and Yang, 2013).

Recently, three main approaches have been introduced to ensure the data confidentiality in cloud environments: a) data encryption, b) combination of encryption and fragmentation (Ciriani et al., 2010), and c) fragmentation (Ciriani et al., 2009). However, in this context, it is in fact crucial to guarantee a proper balance between data confidentiality, on one hand, and other properties, such as, data utility, query execution overhead, and performance on the other hand (Samarati and di Vimercati, 2010; Joseph et al., 2013).

The first approach, denoted by data encryption, consists in encrypting all the data collections. This technique is adopted in the database outsourcing scenario (Ciriani et al., 2010). Actually, encryption algorithms presents increasingly lower costs. Cryptography becomes an inexpensive tool that supports the protection of confidentiality when storing or communicating data (Ciriani et al., 2010). However, deal-

Jr., E., Monteiro, J., Reis, R. and Machado, J.
A Flexible Mechanism for Data Confidentiality in Cloud Database Scenarios.
In *Proceedings of the 18th International Conference on Enterprise Information Systems (ICEIS 2016) - Volume 1*, pages 359-368
ISBN: 978-989-758-187-8

ing with encrypted data may make query processing more expensive (Ciriani et al., 2009; Ciriani et al., 2010). Some techniques have been proposed to enabling the execution of queries directly on encrypted data (remember that confidentiality demands that data decryption must be possible only at the client side) (Samarati and di Vimercati, 2010). These techniques associate with encrypted data indexing information on which queries can be executed. The mainly challenger for indexing methods is the trade off between precision and privacy: more precise indexes provide more efficient query execution but a greater exposure to possible privacy violations (Ceselli et al., 2005; Samarati and di Vimercati, 2010). Besides, the solutions based on an extensive use of encryption suffer from significant consequences due to loss of keys. In the real scenarios, key management, particularly the operations at the human side, is a hard and delicate process (Samarati and di Vimercati, 2010).

The second approach, called combination of encryption and fragmentation, uses encryption together with data fragmentation. It applies encryption only on the sensitive attributes and splits the attributes with sensitive association into several fragments, which are stored by different cloud storage services (Ciriani et al., 2010). In other words, sensitive association constraints are solved via fragmentation, and encryption is limited to those attributes that are sensitive by themselves. Thus, a single cloud service provider cannot join these fragments for responding queries. Therefore, these techniques must also be accompanied by proper query transformation techniques defining how queries on the original data are translated into queries on the fragmented data. Besides, splitting the attributes with sensitive association into some fragment is a NP-hard problem (Samarati and di Vimercati, 2010; Joseph et al., 2013).

The third approach, denoted by fragmentation, does not use cryptography. In this approach, the sensitive attributes remains under the client's custody while the attributes with sensitive association are split into several fragments, which are stored by different cloud storage services (Ciriani et al., 2009). It is important to note that this approach has the same drawbacks discussed previously (for the combination of encryption and fragmentation approach) regarding to query execution (Samarati and di Vimercati, 2010; Joseph et al., 2013).

In this paper, we present i-OBJECT, a new approach to preserve data confidentiality in cloud storage services. The science behind i-OBJECT uses concepts of the Hegel's Doctrine of Being. The proposed approach is based on the information decomposition to split data into unrecognizable parts and store them in different cloud service providers. Besides, i-OBJECT is a flexible mechanism since it can be used alone or together with other previously approaches in order to increase the data confidentiality level. Thus, a user may trade performance or data utility for a potential increase in the degree of data confidentiality. Experimental results show the potential efficiency of the i-OBJECT.

The remain of this paper is organized as follows. Section 2 presents the proposed approach, called i-OBJECT. Experimental results are presented in Section 3. Next, Section 4 addresses related works. Finally, Section 5 concludes this paper and outlines future works.

2 A DECOMPOSITION-BASED APPROACH FOR DATA CONFIDENTIALITY

The proposed approach for ensuring data confidentiality in cloud environments, denoted i-OBJECT, was designed for transactional data. In this environments, reads are much more frequent than write operations. Thus, i-OBJECT needs to be fast to decompose a file and much faster to recompose a file stored in the cloud.

The i-OBJECT approach was inspired by the German philosopher Hegel's work, according to which an object has three fundamental characteristics (Hegel, 1991): quality, quantity and measure. From this idea, we developed the concept of information object (see Definition 1). From this concept, we have developed the processes to: i) fragment a file in a sequence of information objects and ii) decompose each information object in its properties (quality, quantity and measure).

The i-OBJECT approach has three phases: data fragmentation, decomposition and dispersion, which will be discussed later. Figure 1 shows an overview of the the i-OBJECT approach.

2.1 The Fragmentation Phase

In the fragmentation phase, the basic idea consists in split an input file F in a sequence of n information objects (see Definition 1). Then, we can represent a file F as an ordered set $\{iObj_1, iObj_2, \cdots, iObj_n\}$ of i-OBJECTs.

Definition 1 (i-OBJECT)**.** *An information object, i-OBJECT for short, is a piece of 256 sequential bytes from a file.* ◇

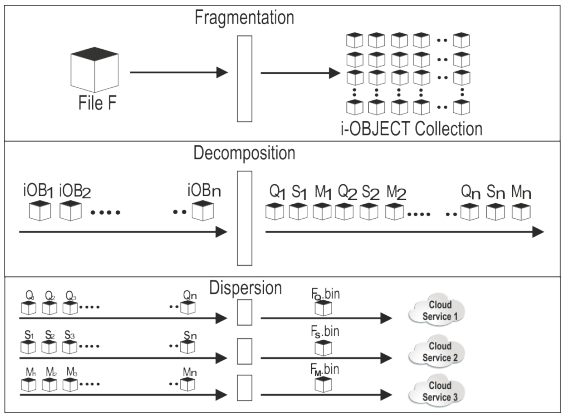

Figure 1: i-OBJECT Approach Overview.

2.2 The Decomposition Phase

The decomposition phase receives as input a file F, represented as an ordered set $\{iObj_1, iObj_2, \cdots, iObj_n\}$ of i-OBJECTs, and using the Hegel's theory (Hegel, 1991), according to which an object has three fundamental characteristics (quality, quantity and measure), decomposes F in three files: Fq.bin, Fs.bin and Fm.bin, which represent, respectively, F's quality, quantity and measure.

In order to understand the decomposition phase, it is necessary to formally define the quality, quantity and measure properties. These definitions are presented next.

Definition 2 (Quality)**.** *Quality is the set of diverse bytes that composes a particular i-OBJECT. Let $iObj_k$ be an i-OBJECT, $Q(iObj_k)$ denotes the quality property of the $iObj_k$. $Q(iObj_k)$ is a ordered vector containing the* m *diverse bytes present in $iObj_k$. More formally, $Q(iObj_k) = \{b_1, b_2, b_3, \cdots, b_m\}$ such that $1 \leqslant b_i \leqslant 256$ and $i \neq j \rightarrow b_i \neq b_j$, where b_i is a byte present in $iObj_k$.* ◇

Definition 3 (Quantity)**.** *Quantity is an array containing the number of times that each distinct byte appears in a specific i-Object. Let $iObj_k$ be an i-OBJECT, $S(iObj_k)$ denotes the quantity property of the $iObj_k$. $S(iObj_k)$ is a vector containing, for each different byte b_j (representing a ASCII Symbol) present in $Q(iObj_k)$ the number of times that b_i appears in $iObj_k$. More formally, $S(iObj_k) = \{s_1, s_2, s_3, \cdots, s_m\}$ such that $1 \leqslant s_i \leqslant 256$, where s_i represents the number of times that b_i appears in $iObj_k$.* ◇

Definition 4 (Measure)**.** *Measure is a two-dimensional array containing, for each diverse byte that composes a particular i-OBJECT, a vector with the positions where this byte occurs in the i-OBJECT. Let $iObj_k$ be an i-OBJECT, $M(iObj_k)$ denotes the measure property of the $iObj_k$. $M(iObj_k)$ is a two-dimensional array containing, for each different byte b_j present in $Q(iObj_k)$, an array m_{b_j} storing the positions in which the byte b_j appears in $iObj_k$. More formally, $M(iObj_k) = \{m_{b_1}, m_{b_2}, \cdots, m_{b_m}\}$, such that, $m = 256$ and $1 \leqslant size(m_{b_i}) \leqslant 256$.* ◇

Given a file F, where $F = \{iObj_1, iObj_2, \cdots, iObj_n\}$. Initially, the decomposition phase consists in extracting, for each i-OBJECT $iObj_k$, where $1 \leqslant k \leqslant n$ and $iObj_k \in F$, its properties: quality ($Q(iObj_k)$), quantity ($S(iObj_k)$) and measure ($M(iObj_k)$).

Next, the proposed approach combines the quality values for all i-OBJECTs in the set $\{iObj_1, iObj_2, \cdots, iObj_n\}$ and creates a file called Fq.bin. After this, the i-OBJECT approach combines the quantity values for all i-OBJECTs in the set $\{iObj_1, iObj_2, \cdots, iObj_n\}$ and creates a file denoted by Fs.bin. Finally, the proposed approach combines the measure values for all i-OBJECTs in the set $\{iObj_1, iObj_2, \cdots, iObj_n\}$ and creates a file called Fm.bin (see Figure 1).

In order to illustrated how an i-OBJECT $iObj_k$ is decomposed into its three basic properties (Q, S and M) consider the following example, denoted Example 1. **Example 1:** Consider an i-OBJECT $iObj_k$ containing the following text:

```
"Google dropped its cloud computing prices
and other vendors are expected to follow
suit, but the lower pricing may not be
the key for attracting enterprises to the
cloud. When Enterprises comes to cloud,
they're more concerned about privacy
and security"
```

Note that $iObj_k$ contains 31 diverse bytes, where each byte represents a ASCII symbol. So, the quality property of the $iObj_k$ is a vector with 31 elements, as follows: $Q(iObj_k)$ = { 32(space), 34("), 39('), 44(,) 46(.), 69(E), 71(G), 87(W), 97(a), 98(b), 99(c), 100(d), 101(e), 102(f), 103(g), 104(h), 105(i), 107(k), 108(l), 109(m), 110(n), 111(o), 112(p), 114(r), 115(s), 116(t), 117(u), 118(v), 119(w), 120(x), 121(y) } Thereby, the quantity property of the $iObj_k$ is also a vector with 31 elements: $S(iObj_k)$ = { 40, 2, 1, 2, 1, 1, 1, 1, 8, 3, 13, 10, 28, 2, 4, 6, 11, 1, 7, 4, 12, 21, 9, 18, 10, 21, 8, 2, 2, 1, 5 } It's important to note the relationship between quality and quantity properties. Note that, for example, the character "space" (ASCII 32), the first element in $Q(iObj_k)$, denoted by $Q(iObj_k)_1$, appears 40 times in the $iObj_k$, and the character "y" (ASCII 121), the last element in $Q(iObj_k)$, denoted by $Q(iObj_k)_{31}$, occurs 5 times in the $iObj_k$. In this

scenario, the measure property of the $iObj_k$ is a two-dimensional array containing 31 arrays, as follows: $M(iObj_k)$ = {{ 7, 15, 19, 25, 35, 42, 46, 52, 60, 64, 73, 76, 83, 89, 93, 97, 103, 111, 115, 119, 122, 126, 130, 134, 145, 157, 160, 164, 171, 176, 188, 194, 197, 204, 212, 217, 227, 233, 245, 241 }, { 0, 255}, { 209 }, ⋯, { 114, 129, 208, 253, 240 }} Observe that, for example, the character "y" (ASCII 121), the 31st element in $Q(iObj_k)$, occurs 5 times ($Q(iObj_k)_{31} = 5$) in the $iObj_k$, in the positions 114, 129, 208, 253 and 240, which are represented by the last array in $M(iObj_k)$.

The Algorithm 1 illustrates how a file F is decomposed in the files $F_q.bin$, $F_s.bin$ and $F_m.bin$. The Algorithm 2 shows how the files $F_q.bin$, $F_s.bin$ and $F_m.bin$ are used to recompose the file F.

The decomposition algorithm (Algorithm 1) is performed in two stages. The first step (lines 1 to 17) obtains the information of the bytes ordinal positions (M) of $iObj_k$ (Q) and stores it in a two-dimensional vector temp [256] [256] (line 12), where the first dimension of the vector represents the decimal value of the byte and the second dimension is a list of the positions occupied by the byte's occurrence in $iObj_k$. The second step of the process (lines 18 to 41) generates three vectors containing the Q and S information, where each element of these sets is represented by one bit, and a byte vector M, which contains the positions grouped in ascending order of Q elements. In groups with more than one element, the two last elements are made to reverse its positions to indicate the end of the group.

The recomposition algorithm (Algorithm 2) receives as input the files $F_q.bin$, $F_s.bin$ and $F_m.bin$ and restores the original file F. The files are read sequentially in blocks of 256 bits (Q and S) and 256 bytes (M) (lines 4 to 6) so that the rebuilding of elements Q, S and M starts. The algorithm goes through the sample space of Q elements (0 to 255), identifying bytes exist in $iObj_k$ (line 11) and if they occur one time or more than once (S) (line 15) to then retrieve positions occupied by these bytes and insert them in vector $iObj_k$ (rows 18 and 22).

Algorithm 1: Decomposition function.

```
   input  : File F
   output : File Fq.bin, Fs.bin, Fm.bin
 1 Function − Decomposition(F);
 2 begin
 3     IObj = Read(F,256);
 4     CreateFile(Fq.bin,Fs.bin,Fm.bin);
 5     Temp[256][256];
 6     while IObj not null do
 7         byte = 0; cont = 0; freq[] = 0;
 8         for pos = 0 to 255 do
 9             byte = IObj[pos];
10             freq[byte] ++;
11             cont = freq[byte];
12             Temp[byte][cont] = pos;
13         end for
14         CreateQSM(Temp, freq);
15         IObj = Read(F,256);
16     end while
17 end
18 Function − CreateQSM(Temp[256][256], freq[256]);
19 begin
20     Q[256] = 0;S[256] = 0;M[256] = 0;
21     pos = 0; postemp = 0; cont = 0; cont2 = 0;
22     for byte = 0 to 255 do
23         if freq[byte] ≥ 1 then
24             Q[byte] = 1;
25             if freq[byte] ≥ 2 then
26                 S[cont2] = 1;
27                 postemp = Temp[byte][freq[byte]];
28                 Temp[byte][freq[byte]] =
                     Temp[byte][freq[byte] − 1];
29                 Temp[byte][freq[byte] − 1] = postemp;
30             end if
31             for cont = 1 to freq[byte] do
32                 M[pos] = Temp[byte][cont];
33                 pos ++;
34             end for
35             cont2 ++;
36         end if
37     end for
38     Append(Fq.bin,Q);
39     Append(Fs.bin,S);
40     Append(Fm.bin,M);
41 end
```

2.3 The Dispersion Phase

In the dispersion step, the files $F_q.bin$, $F_s.bin$ and $F_m.bin$ are spread across different cloud storage service providers. So, i-OBJECT requires that these three files are isolated between themselves.

3 DATA CONFIDENTIALITY CONSIDERATIONS

The data confidentiality in the proposed approach stems from the fact that the files Fq.bin, Fs.bin and Fm.bin are stored in different cloud providers, which are physically and administratively independent. According to (Resch and Plank, 2011), physical data dispersion in different storage servers, along with the careful choice of the number of servers and the amount of fragments needed for restore the original

Algorithm 2: Recomposition function.

```
   input  : File Fq.bin, Fs.bin, Fm.bin
   output : File F
 1 Function − Recomposition(F);
 2 begin
 3     CreateFile(F);
 4     Q[256] = Read(Fq.bin,256);
 5     S[256] = Read(Fs.bin,256);
 6     M[256] = Read(Fm.bin,256);
 7     while Q not null do
 8         ordem = 0; cont = 0; pos = 0;
 9         for bit = 0 to 255 do
10             if Q[bit] ≠ 0 then
11                 pos ← M[ordem];
12                 IObj[pos] ← CodASCII(Q[bit]);
13                 ordem ++;
14                 if S[cont] = 1 then
15                     while M[ordem] ⩾ pos do
16                         pos ← M[ordem];
17                         IObj[pos] ← CodASCII(Q[bit]);
18                         ordem ++;
19                     end while
20                     pos ← M[ordem];
21                     IObj[pos] ← CodASCII(Q[bit]);
22                     ordem ++;
23                 end if
24                 cont ++;
25             end if
26         end for
27         Append(F,IObj);
28         Q[256] = Read(Fq.bin,256);
29         S[256] = Read(Fs.bin,256);
30         M[256] = Read(Fm.bin,256);
31     end while
32 end
```

files, reduces the chances of an attacker and is enough to make a system safe. So, in i-OBJECT, the degree of data confidentiality is based on the difficulty of the attackers to reconstruct the i-OBJECTS from one of these three files, Fq.bin, Fs.bin or Fm.bin. Besides, i-OBJECT is a flexible mechanism since it can be used alone or together with other previously approaches (such as encryption algorithms like AES, DES or 3-DES) in order to increase the data confidentiality level.

4 EXPERIMENTAL EVALUATION

In order to show the potentials of i-OBJECT, several experiments have been conducted. The main results achieved so far are presented and discussed in this section. Thus, we first provide information on how the experimentation environment was set up. Thereafter, empirical results are quantitatively presented and qualitatively discussed.

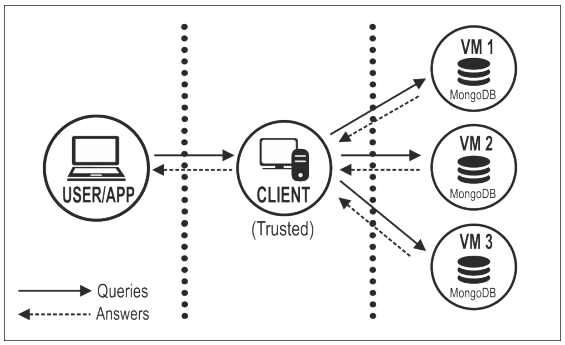

Figure 2: Experimental Architecture.

4.1 Experimental Setup

We implemented i-OBJECT and the other data confidentiality approaches using C and Java. In order to run these approaches we have used a private cloud computing infrastructure based on OpenStack. Figure 2 shows the architecture used in the experiments, which contains two kinds of virtual machines: the client node and the data storage nodes. The client, a Trusted Third Party (TTP), runs the i-OBJECT algorithms: decomposition and recomposition. The data storage nodes (called VM1, VM2 and VM3) emulate three different cloud storage service providers. We assume that the data nodes provide reliable content storage and data management but are not trusted by the client to maintain data privacy.

Each data storage node has the following configuration: Ubuntu 14.04 operating system, Intel Xeon 2.20 GHz processor, 4 GB memory and 50 GB disk. The client is a Intel Xeon 2.20 GHz processor, 4 GB memory and 40 GB disk capacity, running a Windows Server 2008.

Besides this, each data storage node has a MongoDB instance with default settings. MongoDB is an open source, scalable, high performance, schema-free, document oriented database. We opted to use the MongoDB because it is one of the most used database in cloud computing environments and exists opportunities for improvement its security and privacy. MongoDB supports a binary wire-level protocol but doesn't provide a method to automatically encrypt data. This means that any attacker with access to the file system can directly extract the information from the files (Okman et al., 2011).

In order to evaluate the i-OBJECT efficiency, we have used a document collection, synthetically created, which contains files (documents) with different sizes. Each file has four parts (or attributes), which have the same size. These attributes are: curriculum vitae (A1), paper text (A2), author photo (A3) and paper evaluation (A4).

4.2 Test Results

In this section, we present the results of the experiments we carried out. For evaluate the i-OBJECT efficiency, we have used two metrics: the input and output times. The input time is defined as the total amount of time spent to process a file F and generates the data that will be send to the cloud storage service providers. The output time is defined as the total amount of time spent to process the data received from the cloud storage service providers in order to remount the original file F. It is important to highlight that the time spend in the communication process, to send and receive data to the cloud providers do not composes neither the input nor the output time. We have evaluated different file sizes (2^{18}, 2^{20} and 2^{22} bytes). For each distinct file size, we have used 10 files and computes the average time for decomposes and recomposes these files. To validate the i-OBJECT approach, we have evaluated four different scenarios, which will be discussed next.

4.2.1 Scenario 1: Encryption Algorithms

The first scenario was running with the aim of compare the most popular symmetric cryptographic algorithms: AES, DES and 3-DES. It is important to note that, in this experiment, the Input Time matches the Encryption Time (the spent time to encrypt a file F) and Output Time matches the Decryption Time (time necessary to decrypt a file F).

So, in this scenario, the client receives a file F from the user, encrypt it, generating a new file F_e, and sends F_e to VM1. Figure 3 shows the encryption time for the algorithms AES, DES and 3-DES. Next, the client receives the encrypted file F_e from VM1, decrypt it, generating the original file F and sends F to the user. Figure 4 shows the decryption time for the algorithms AES, DES and 3-DES. Note that, for files with size of 2^{16} bytes these three algorithms presented the same encryption time, while AES and DES presented the same decryption time. However, for files with sizes of 2^{18}, 2^{20}, and 2^{30} bytes AES outperforms DES and 3-DES, for both encryption and decryption.

4.2.2 Scenario 2: Data Confidentiality Approaches

In the second scenario, we compared the i-OBJECT approach with the three main approaches to ensure the data confidentiality in cloud environments: a) data encryption, b) combination of encryption and fragmentation, and c) fragmentation (see Section 1) (Samarati and di Vimercati, 2010; Joseph et al., 2013).

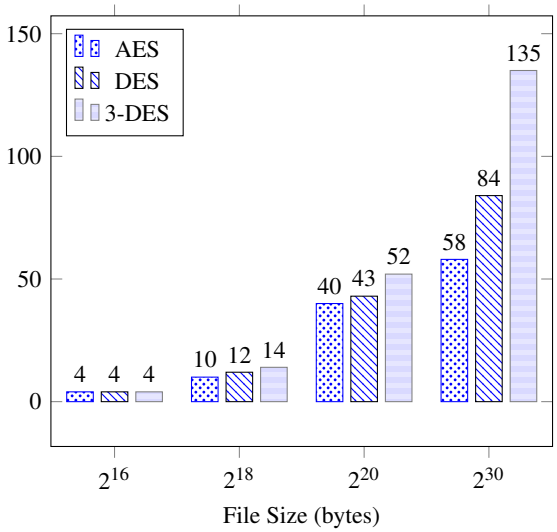

Figure 3: Scenario 1: Encryption Time.

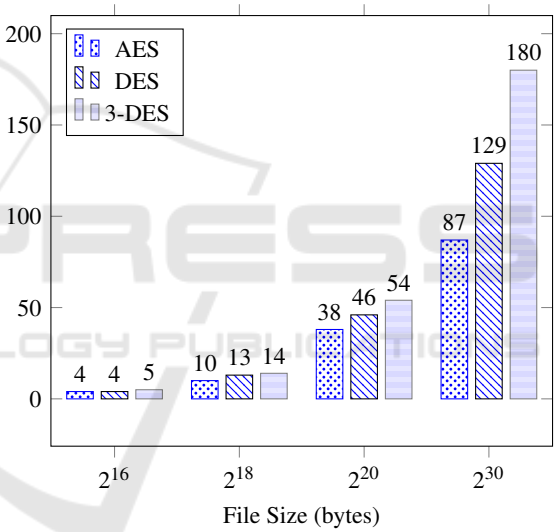

Figure 4: Scenario 1: Decryption Time.

In order to run the approaches (b), combination of encryption and fragmentation, and (c), fragmentation, it is necessary to define which attributes are sensitive, besides to identify the sensitive association between attributes. Moreover, splitting the attributes with sensitive association into some fragment is a NP-hard problem (Samarati and di Vimercati, 2010; Joseph et al., 2013).

Thus, we have assumed that each document has four attributes: curriculum vitae (A1), paper text (A2), author photo (A3) and paper evaluation (A4). Besides, we supposed that there is a set C of sensitive association constraints, with the following constraints: C_1={A1}, C_2 = {A2}, C_3 = {A2, A4}, C_4 = {A1, A3}. So, the attributes A_1 and A_3 are considered sensitive and must be encrypted in approaches

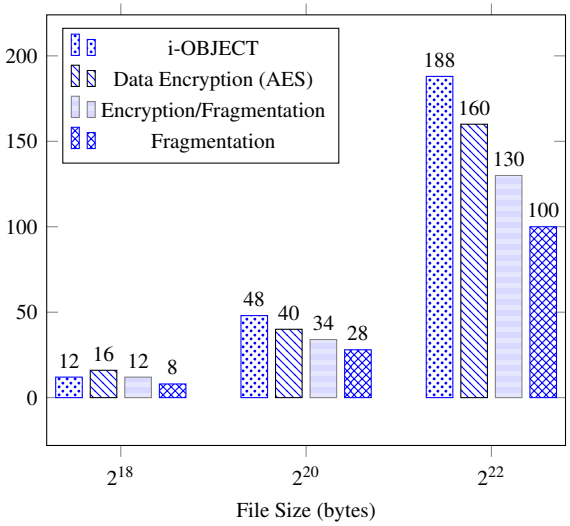

Figure 5: Scenario 2: Input Time.

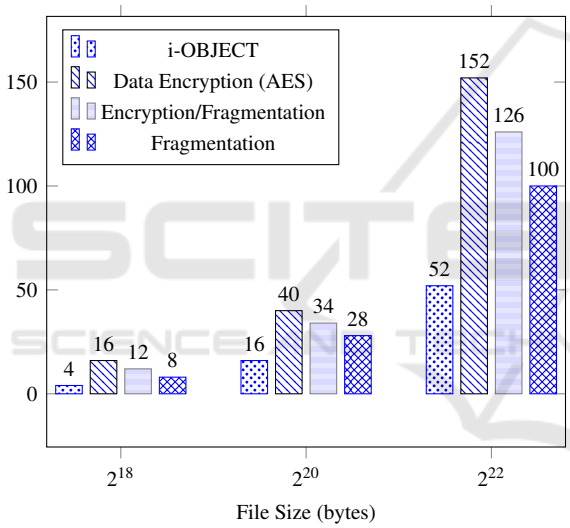

Figure 6: Scenario 2: Output Time.

(b) and (c). The constraint $C_3 = \{A2, A4\}$ indicates that there is a sensitive association between A2 and A4. The constraint $C_4 = \{A1, A3\}$ indicates that there is a sensitive association between A1 and A3, and these attributes should be stored in different servers in the cloud. Based on the set C of confidentiality constraints, a set P of data fragments was generated, as following: i) the approach (b), combination of encryption and fragmentation, produced the fragments P_1={A1,A4} and P_2 = {A2,A3}; and ii) the approach (c), fragmentation, formed the fragments P_3 = {A1, A3}, P_4 = {A2} and P_5 = {A4}.

It is important to emphasize that the time necessary to define the set of fragments (fragments schema) for splitting the attributes with sensitive association, that is a NP-hard problem, was not considered in this experiment. Furthermore, for the approach (a), data encryption, we have used the AES algorithm, since it presented best results in the first scenario.

In this experiment, we considered performance with respect to the following metrics: (i) Input Time and (ii) Output Time. These metrics change a little according to the used data confidentiality approach.

Input Time is computed as following:

- Approach (a), data encryption: time to encrypt a file F using AES, generating a file F_e. The file F_e will be send to VM1;
- Approach (b), combination of encryption and fragmentation: time to encrypt A_1 and A_3, plus the time to generates P_1 and P_2. Where, A_1 and P_1 will be send to VM1, while A_3 and P_2 will be send to VM2.
- Approach (c), fragmentation: the time to generates P_3, P_4 and P_5. Where, P_3 will be send to VM1, P_4 to VM2 and P_5 to VM3.
- i-OBJECT Approach: time to decompose a file F into $F_q.bin$, $F_s.bin$ and $F_m.bin$. Where, $F_q.bin$ will be send to VM1, $F_s.bin$ to VM2 and $F_m.bin$ to VM3;

Output Time is computed as following:

- Approach (a), data encryption: time to decrypt a file F_e using AES, generating the original file F;
- Approach (b), combination of encryption and fragmentation: time to decrypt A_1 and A_3, plus the time to join A_1, A_3, P_1 and P_2 in order to remount the file F;
- Approach (c), fragmentation: the time to join P_3, P_4, P_5 and the sensitive attributes stored in the client;
- i-OBJECT Approach: time to recompose a file F from $F_q.bin$, $F_s.bin$ and $F_m.bin$.

Figure 5 shows the input time for the evaluated approaches. Note that i-OBJECT approach has a performance slightly worse than Data Encryption (AES). Fragmentation approach outperforms the other approaches, for all file sizes. On the other hand, the last two approaches, Encryption/Fragmentation and Fragmentation, need to define the set of fragments (fragmentation schema) for splitting the attributes with sensitive association. However, how we have used a fixed example, the time necessary to define the fragmentation schema was not computed. In part, this explains the better results obtained by these two approaches.

Figure 6 shows the output time for the evaluated approaches. Note that i-OBJECT outperforms all the other approaches, for all file sizes. It is important to

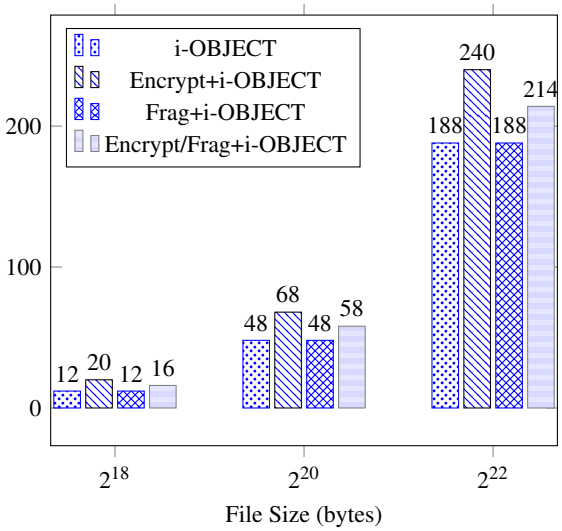

Figure 7: Scenario 3: Input Time.

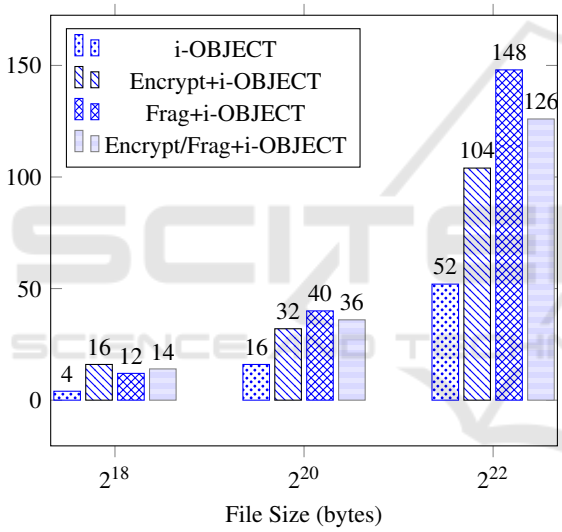

Figure 8: Scenario 3: Output Time.

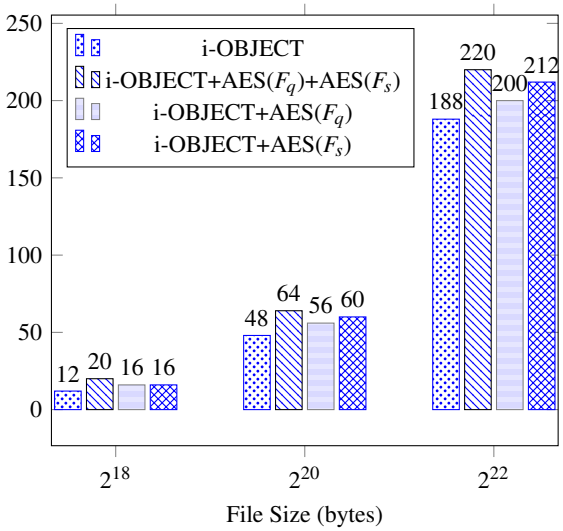

Figure 9: Scenario 4: Input Time.

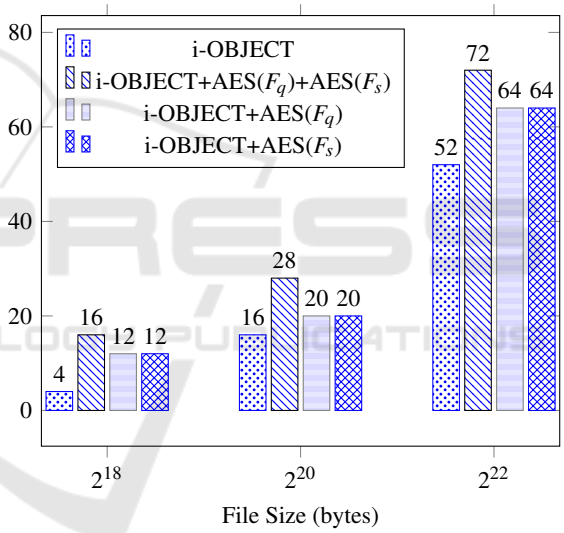

Figure 10: Scenario 4: Output Time.

highlight, for the file size of 2^{22} bytes, i-OBJECT is 88s slower than the Fragmentation approach in the input phase, that is, to process and a file F before sending it to the cloud storage service provider. However, for the same file size, i-OBJECT is 48s faster than the Fragmentation approach. So, for a complete cycle of file write and read, i-OBJECT is just 40s slower than the Fragmentation approach. Then, if the user writes F one time and reads F two times, i-OBJECT is 8s faster than Fragmentation approach. Thus, i-OBJECT outperforms all the other previous approach in scenarios where the number of reads is at least twice larger than the number of writes, which is expected real databases and cloud storage environments.

4.2.3 Scenario 3: Using i-OBJECT Together with Previous Approaches

In the third scenario, we evaluated the use of i-OBJECT together with previous approaches. We believe that i-OBJECT can be used together with other data confidentiality approaches in order to improve their data confidentiality levels.

Figure 7 shows the input time for the evaluated approaches. In this chart, the first bar shows the input time to i-OBJECT (that is, the time to decompose a file F); the second bar represents the input time to apply the Data Encryption approach and, after that, the i-OBJECT (that is, the time to encrypt a file F, producing a new file F_e, plus the time to decompose

F_e); the third bar indicates the input time to apply the Encryption/Fragmentation approach and, next, the i-OBJECT; finally, the fourth bar denotes the input time to apply the Fragmentation approach and then the i-OBJECT. Then, we can argue that i-OBJECT is a flexible approach, in the sense that it can be used together with previous approaches, in order to improve their data confidentiality level. The results presented in Figure 7 show that this strategy provides a small increase in the input time, while providing a high gain in the data confidentiality. Figure 8 shows that the output time overhead has a similar behavior that input time.

4.2.4 Scenario 4: Improving Data Confidentiality in i-OBJECT

In the fourth scenario, we evaluated some strategies to improve the data confidentiality in the i-OBJECT approach. The first strategy consists in encrypt the files $F_q.bin$ and $F_s.bin$, the second strategy consists in encrypt only the file $F_q.bin$ and the third strategy consists in encrypt just the file $F_s.bin$.

Figure 9 and Figure 10 show, respectively. the input and output time with and without the use of these strategies. Note that the strategy of encrypting just the file $F_q.bin$ provides a low overhead, while greatly increases the data confidentiality level of the i-OBJECT approach.

4.2.5 Storage Space Considerations

In the i-OBJECT approach, a file F is decomposed into three files (Fq.bin, Fs.bin and Fm.bin), which are dispersed (sent) to three different cloud providers. The experimental results showed that the size of these files represent, respectively, 12.5%, 12.5% and 90% of the original file size. So, adding these values, the propposed approach provides an overhead of 15% in the disk space utilization. This drawback is minimized since the cloud storage services are designed to store a large quantity of data.

5 RELATED WORK

A significant amount of research has recently been dedicated to the investigation of data confidentiality in cloud computing environment. Most of this work has assumed the data to be entirely encrypted, focusing on the design of queries execution techniques (Ciriani et al., 2010). In (Ceselli et al., 2005) the authors discuss different strategies for evaluating the inference exposure for encrypted data enriched with indexing information, showing that even a limited number of indexes can greatly favor the task for an attacker wishing to violate the data confidentiality provided by encryption.

The first proposal proposing the storage of plaintext data, while ensuring a series of privacy constraints was presented in (Aggarwal, 2005). In this work, the authors suppose data to be split into two fragments, stored on two honest-but-curious service providers, which never exchange information, and resorts to encryption any time these two fragments are not sufficient for enforcing confidentiality constraints. In (Ciriani et al., 2009; Ciriani et al., 2010), the authors address these issues by proposing a solution that first split the data to be protected into several (possibly more than two) different fragments in such a way to break the sensitive associations among attributes and to minimize the amount of attributes represented only in encrypted format. The resulting fragments may be stored at different servers. The proposed heuristic to design these fragments present a polynomial-time computation cost while is able to retrieve solutions close to optimum. In (Xu et al., 2015), the authors propose an efficient graph search based method for the fragmentation problem with confidentiality constraints, which obtains near optimal designs.

The work presented in (Ciriani et al., 2009) proposes a novel paradigm for preserving data confidentiality in data outsourcing which departs from encryption, thus freeing the owner from the burden of its management. The basic idea behind this mechanism is to involve the owner in storing the sensitive attributes. Besides, for each sensitive association, the owner should locally store at least an attribute. The remainder attributes are stored, in the clear, at the server side. With this fragmentation process, an original relation R is then split into two fragments, called F_o and F_s, stored at the data owner and at the server side, respectively. (Wiese, 2010) extends the "vertical fragmentation only" approach and proposes use horizontal fragmentation to filter out confidential rows to be securely stored at the owner site. (Krishna et al., 2012) proposes an approach based on data fragmentation using graph-coloring technique wherein a minimum amount of data is stored at the owner. In (Rekatsinas et al., 2013) the authors present SPARSI, a theoretical framework for partitioning sensitive data across multiple non-colluding adversaries. They introduce the problem of privacy-aware data partitioning, where a sensitive dataset must be partitioned among k untrusted parties (adversaries). The goal is to maximize the utility derived by partitioning and distributing the dataset, while minimizing the total amount of sensitive information disclosed. Solving

privacy-aware partitioning is, in general, NP-hard, but for specific information disclosure functions, good approximate solutions can be found using relaxation techniques.

In (Samarati and di Vimercati, 2010) the authors discuss the main issues to be addressed in cloud storage services, ranging from data confidentiality to data utility. They show the main research directions being investigated for providing effective data confidentiality and for enabling their querying. The survey presented in (Joseph et al., 2013) addressed some approaches for ensuring data confidentiality in untrusted cloud storage services. In (Samarati, 2014), the authors discuss the problems of guaranteeing proper data security and privacy in the cloud, and illustrate possible solutions for them.

6 CONCLUSION AND FUTURE WORK

Experimental results showed the efficiency of i-OBJECT, which can be used with any kind of file and is more suitable for files larger than 256 bytes, files with high entropy and environments where the number of read operations exceeds the number of writes. As a future work, we intend realize a detailed analysis of the i-OBJECT security and evaluate i-OBJECT performance with other data types and using different cloud configurations, including public and mixed clouds.

REFERENCES

Aggarwal, C. C. (2005). On k-anonymity and the curse of dimensionality. In *Proceedings of the 31st international conference on Very large data bases*, pages 901–909. VLDB Endowment.

Camenisch, J., Fischer-Hbner, S., and Rannenberg, K. (2011). *Privacy and identity management for life*. Springer.

Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Jajodia, S., Paraboschi, S., and Samarati, P. (2005). Modeling and assessing inference exposure in encrypted databases. *ACM Transactions on Information and System Security (TISSEC)*, 8(1).

Ciriani, V., De Capitani Di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. (2009). Keep a few: Outsourcing data while maintaining confidentiality. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, pages 440–455, Berlin, Heidelberg. Springer-Verlag.

Ciriani, V., Vimercati, S. D. C. D., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. (2010). Combining fragmentation and encryption to protect privacy in data storage. *ACM Trans. Inf. Syst. Secur.*, 13(3):22:1–22:33.

Clarke, R. (1999). Introduction to dataveillance and information privacy, and definition of terms.

Hegel, G. (1991). The encyclopedia logic (tf geraets, wa suchting, hs harris, trans.). *Indianapolis: Hackett*, 1.

Joseph, N. M., Daniel, E., and Vasanthi, N. A. (2013). Article: Survey on privacy-preserving methods for storage in cloud computing. *IJCA Proceedings on Amrita International Conference of Women in Computing - 2013*, AICWIC(4):1–4. Full text available.

Krishna, R. K. N. S., Sayi, T. J. V. R. K. M. K., Mukkamala, R., and Baruah, P. K. (2012). Efficient privacy-preserving data distribution in outsourced environments: A fragmentation-based approach. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, pages 589–595, New York, NY, USA. ACM.

Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., and Abramov, J. (2011). Security issues in nosql databases. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 541–547.

Rekatsinas, T., Deshpande, A., and Machanavajjhala, A. (2013). Sparsi: Partitioning sensitive data amongst multiple adversaries. *Proc. VLDB Endow.*, 6(13):1594–1605.

Resch, J. K. and Plank, J. S. (2011). Aont-rs: blending security and performance in dispersed storage systems. In *Proceedings of FAST-2011: 9th Usenix Conference on File and Storage Technologies,February 2011*.

Samarati, P. (2014). Data security and privacy in the cloud. In *Information Security Practice and Experience - 10th International Conference, ISPEC 2014, Fuzhou, China, May 5-8, 2014. Proceedings*, pages 28–41.

Samarati, P. and di Vimercati, S. D. C. (2010). Data protection in outsourcing scenarios: Issues and directions. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 1–14, New York, NY, USA. ACM.

Wiese, L. (2010). *Horizontal fragmentation for data outsourcing with formula-based confidentiality constraints*, pages 101–116. Springer.

Xu, X., Xiong, L., and Liu, J. (2015). Database fragmentation with confidentiality constraints: A graph search approach. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, CODASPY '15, pages 263–270, New York, NY, USA. ACM.

Zhifeng, X. and Yang, X. (2013). Security and privacy in cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(2):843–859.