Searching Private Data in a Cloud Encrypted Domain

Bernardo Ferreira
FCT/UNL – CITI*
Campus da Caparica
2825-465 Caparica
bernardof@acm.org

Henrique Domingos
FCT/UNL – CITI*
Campus da Caparica
2825-465 Caparica
hj@fct.unl.pt

ABSTRACT
Cloud computing security and reliability are important challenges in the research agenda. For some applications managing sensitive data, cloud security solutions and data-privacy management are the main concerns for organizations that are considering a move to the cloud. The advantages of cloud computing include reduced costs, easy maintenance and re-provisioning of resources, thereby also possibly increasing profits. But the adoption of Cloud Computing solutions applies only if different security concerns are ensured. This article presents a solution for data storage and data management in Internet Storage Clouds, preserving privacy conditions under the control of Cloud users. The proposed solution supports operations over stored encrypted data, including reading, writing and searching based on relevance ranking and multiple keywords. The approach is based on a middleware architecture supported by homomorphic encryption techniques combined with dynamic indexing mechanisms. The solution preserves data-privacy without need to either decipher data during operations in the Cloud or transfer the data during searches. The article further describes an implementation prototype of the solution and its evaluation. The evaluation shows that the solution is viable, offers security and privacy control for the user and does not aggravate conditions of data-access latency and availability.

Categories and Subject Descriptors
C.2.0 [**Computer-Communication Networks**]: General-Security and protection; C.2.4 [**Distributed Systems**]: Distributed applications; E.3 [**Data**]: Data encryption

General Terms
Algorithms, Security

Keywords
Cloud Computing, Internet Storage Clouds, Security and Privacy, Homomorphic Ciphers, Search and Ranking of Data

1. Introduction
The security guarantees of operations and data stored in storage Clouds solutions, offered nowadays by Internet providers (or Internet Storage Cloud Solutions), are decisive criteria in the generalized adoption of those solutions. These solutions present interesting storage and remote data access characteristics, with flexibility of configuration in regard to storage space necessities in each instant, interesting quality of service and reliability and pay-per-use charging models [1]. At first glance they are advantageous solutions, from a technical and operational viewpoints, as well as from an economical analysis, avoiding overloads of management and administration of software solutions or software licensing for data kept by the clients themselves. The existing cloud solutions offer a data storage service with good dependability, accessibility and availability guarantees, in ubiquitous access conditions, independent of geographical location [2]. However, when effective and independent control conditions are required by the users over the availability, security and privacy of outsourced data-storage and operations over sensitive data, the adoption of those solutions can be more problematic.

The undue access or unauthorized disclosure of private data kept in Internet Storage Clouds (ISC) has been referred as a critical problem, not only in the use case of secure data backup solutions but also to preserve security guarantees of sensitive and private data accessed by online applications [3]. This is the case of applications managing and searching medical records; accessing private videos or photographs; managing financial data; or accessing, searching and consulting governmental documents [4]. The dependency of third party trust in ISC outsourced data prevents the control and complete auditing, by the end-users, of possible security vulnerabilities in the computational infrastructure (hardware/software) of the providers, allowing the data to be targeted by possible illicit actions or unattended operation by technical and administrative staff. More specifically, attacks have been verified due to eventual security deficiencies or that explore physical access to the computational and communicational resources used by the providers, including network and communications equipments, software systems, or computational infrastructure devices: memory, hard drives or internal backup solutions [5].

To address the above problems and take the advantages of ISC and Cloud-Computing Solutions as interesting available Internet Services, a dependable solution is needed, conjugating two fundamental dimensions: (i) data privacy management, preserving privacy conditions during data searches or other possible operations; (ii) scalability and performance guarantees, for the possible management of big data sets, supporting large number of operations and possible access by multiple users.

The solutions proposed in recent years are mostly based in the use of cryptographic techniques to encrypt the stored data (ex. [6, 7]). In its majority they advocate the protection of data stored encrypted in the servers, being the encryption done before the data is outsourced [8]. In order to be operated, data must be transferred to the clients that proceed to its decryption and then execute the operations. These solutions are limited when data processing at the cloud server-side is required, for efficiency and low-latency

* Faculty of Sciences and Technology, New University of Lisbon, Center for Informatics and Information Technologies.

This work is supported by FCT-UNL PhD Fellowship Program and FCT-MEC under the PTDC/EIA/113729/2009 Grant (SITAN - Services for Intrusion Tolerance in Ad-Hoc Networks)

requirements. This is the case of applications using storage clouds as remote data-storage backends or applications using Databases as Services, provided by Cloud-Solution Providers. Those solutions are equally limited in applications that have to process or search big data volumes organized in key-value stores, as is the case of many Cloud data processing applications.

In this article we propose a solution that has in sight the conjugation of the two fundamental dimensions, as previously enounced. The solution addresses security and privacy concerns, maintaining the independent control of data-privacy by the end-user, promoting a trustable environment for data storage and data management on Internet provided storage clouds, reducing the role of Cloud-Providers as Trust-Entities. The solution is based on a middleware architecture supported by homomorphic encryption techniques combined with dynamic indexing mechanisms. The mechanisms preserve data-privacy conditions without need to either decrypt data during searching operations in the cloud, also avoiding the need to transfer the data during searches. The middleware is neutral to different Cloud-Solution Providers and may be easily adopted to operate with different Clouds, at the same time. The article presents the middleware system model and architecture, its processing components, explaining the proposed mechanisms conjugating homomorphic encryption and dynamic indexing mechanisms for private data searching based on multiple keywords and relevance ranking. The paper further describes an implementation prototype of the solution and its evaluation. The evaluation shows that the solution is viable, offers security and preserves privacy control for the user. Our observations show that the proposed middleware services not aggravate data-access latency conditions and data availability, comparing with the use of current clouds.

Paper Organization

The rest of the paper is organized in the following way: in section 2 we introduce the relevant related work; section 3 presents a middleware system model and architecture to address secure indexing and searching over private data stored encrypted in the Cloud; section 4 addresses flexibility issues for the implementation of the proposed system model, according to different requirements and tradeoffs related with security, performance and available computational resources; section 5 discusses in more detail the main components of the proposed solution, as well as, the processing algorithms for indexing, privacy protection and secure searching; section 6 addresses the experimental evaluation of the implemented solution and prototype; finally, section 7 concludes the paper, presenting some future research work directions.

2. Related Work

One of the most common types of searches over files or text documents is based on the use of keywords or subsets of searchable metadata. Usually the keywords and the searches are supported over the original documents in plaintext. The need to execute these searches over ciphered data raises new problems. In general, operations on the cipher data require it to possess homomorphic characteristics in relation to plaintext data. This is, apparently, a contradictory aspect regarding the security properties of the cryptographic algorithms themselves. This means that a good homomorphic cryptographic scheme must be able to combine the homomorphic properties required while preserving the security characteristics, in accordance to security analysis criteria as considered for conventional cryptography.

Different homomorphic properties may be needed for different purposes [9, 10]. A cryptographic transformation presents pure and complete homomorphism when any operation on the plaintext data can be transformed into an equivalent operation on the ciphertext data. This vision of full homomorphism does not have today a generic and practical solution based solely on pure homomorphic cryptographic algorithms. However, partially or incomplete homomorphic encryption schemes can be addressed. These schemes are usually based on conventional cryptography and, as such, share the same performance and security levels. A cryptographic transformation is said to be partially homomorphic when the homomorphism is only kept for a subset of the operations, such as: a single operation or a limited group of operations (ex. additions and subtractions, multiplications and divisions, etc.).

Although some conventional encryption schemes (ex. [9, 11, 12]) allow searching over the ciphered data, exploring partially homomorphic properties, a large group of the current practical approaches only address Boolean searches, that is, searches that verify the (in)existence of one or more text terms. Such solutions do not allow the capture of complementary indicators, including relevance scores and related metrics, needed for multi-keyword ranked searches. In this case, the common solutions present limitations, possibly not supporting: (1) searches without complete or partial data transfer to a user's trust base, where they can be decrypted during the search process; (2) searches requiring low processing latency and avoiding network traffic, also affecting the "pay-per-use models" common in most Cloud repositories.

Recent works have demonstrated the practicality of homomorphic encryption schemes in other forms of searching. A good example comes from the encrypted relational database world, where the system CryptDB [10] allows around 99.5% of all SQL queries to be executed over the encrypted data. This is achieved in the system by combining different partially homomorphic schemes. Another example comes from information retrieval on the Cloud with ranked queries [20], where a partially homomorphic scheme is used to allow searching while protecting search, access and rank privacy. More related to our work is [13], which aims at supporting single keyword ranked searching over encrypted Cloud data. The work uses a TF-IDF approach to scoring and achieves security "as strong as possible" with good efficiency conditions, through the use of an order-preserving encryption scheme that allows ordering rank scores while encrypted and stored in the Cloud. In [19], the same authors revisit the topic and address multi-keyword ranked searching through a secure k-nearest neighbor (kNN) technique, combined with a "coordinate matching" ranking function. However this last work does not foresee a dynamic scenario where the index can be updated, and the encryption scheme used requires sequential scanning of the index each time a search is requested. Finally, both works require the index to be built locally, encrypted and only then it can be outsourced to the Cloud. In this paper we follow (for the first time, as far as we know) an approach that allows the ranking and indexing of encrypted documents in the Cloud while preserving their privacy. This approach requires considerably less computational power in the Client and fully explores the potential of the Cloud, with good performance and security conditions as proved by our experimental results.

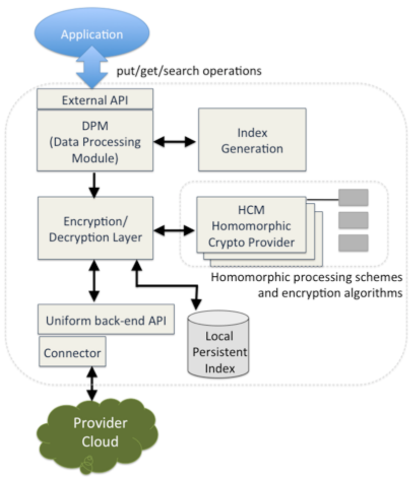
3. System model and architecture

The proposed solution aims to manage data privacy by using a security approach, which allows the storage of sensitive data on Internet Storage Clouds under control of the users who own the data. For this purpose the solution addresses the following requirements:

- Confidentiality, integrity and privacy of the data, with independent control and auditing by the users;
- Extension of the previous guarantees to the search operations over the encrypted data, namely supporting secure ranked multi-keyword searches;
- Promotion of a solution that can operate under full control of the users, independently of trustability services or guarantees offered by Cloud providers;

Figure 1 shows the reference architecture for the proposed solution. The architecture has in sight the use of homomorphic encryption techniques (HCM module) combined with data indexing and dynamic updates, in a middleware solution between the final user applications and the Internet Cloud services.

System Model. The users execute applications that interact in a secure, authenticated way with the middleware. User data or documents (typically sets of files) are processed and indexed by the middleware in order to be securely transferred to the Cloud. Data or documents stored in the Cloud are maintained encrypted even when remote operations are executed. This way, the middleware offers data storage outsourcing while preserving privacy control over that data. A dynamic system is considered, where documents can be inserted and removed at anytime of the middleware execution. Documents are also considered mutable, that is, new versions of already existing documents can be inserted, with repercussion on the metrics stored in the index.



Application
put/get/search operations
External API
DPM (Data Processing Module)
Index Generation
Encryption/ Decryption Layer
HCM Homomorphic Crypto Provider
Homomorphic processing schemes and encryption algorithms
Uniform back-end API
Connector
Local Persistent Index
Provider Cloud

Figure 1. Reference architecture of the envisioned solution

Adversary Model. In the previous model, the trust base for preserving conditions of dependability, availability, security and privacy is restricted to the components of the middleware system. The storage Clouds can be considered not trustable, admitting they may be subject to attacks of intrusion. In the case of having the middleware implemented as a service in the Cloud (more details in section 4), it is considered that the servers of this computational Cloud are not subject of attackers aiming at destroying its data, disrupting their service or *crashing* the software. It is assumed, in the case of this single Cloud provider, that its HW/SW infrastructure is always dependable and available, executing the middleware system correctly and according to specification. Nonetheless, attacks on the communications or on the servers aiming at breaking data privacy, are admitted.

Middleware External API. The middleware provides a web service security API (SSL supported) described in the following table. The API implements data management operations as in key-value storage Clouds. Additionally, a search operation is supported. In the system model implementation this operation implements ranked multi-keyword searching as will be discussed later (section 5.2).

Operation	**Description**
*ObjectId **write** (Object)*	Writes an object to be transferred to the Cloud
*Object **get** (ObjectId)*	Retrieves an object from the Cloud using it's identifier
*Set<ObjectId> **search** (Set<Keyword>)*	Searches the Cloud repository for a given set of keywords, returning a set of object identifiers

Table 1: External Middleware API

Using the above API, applications can access the middleware services in a transparent way, since the provided **write**() and **get**() operations are similar to the usual operations offered by current Cloud service providers.

Middleware Processing. Internally, the different modules implement the required algorithms for indexing, data encryption and searching. When the user writes documents through the external API, the indexing module builds a searchable secure index from all relevant keywords. This index is then encrypted and stored persistently. As we will discuss later, the index may be stored locally, in a data repository component, or remotely in the Cloud. At the same time, documents are also encrypted and (always) remotely stored in the Cloud. As discussed in the next section, this processing allows the middleware to be used in multiple architectural patterns, according to different requirements, trusted computer-base assumptions and tradeoffs.

4. Implementation Options

The system model described in the previous section allows for a generic and flexible solution that can fit different purposes, according to various requirements. By making tradeoffs between security level requirements, performance of operations and local resources available, three implementation scenarios were conceived. It should be noted that in all scenarios the documents themselves are always stored securely in the Cloud and the encryption keys used never leave the user's device. The presented scenarios focus on possible implementations of the solution in a single user perspective. The generalization of the solutions to multi-user environments is in our on-going research work agenda.

Middleware in user's trustable device. This scenario is based on high security and performance requirements. The user wishes to have the best available security and chooses to keep the middleware running inside his trustable device, in an attempt to minimize system exposure. As he also wants to keep search operations fast, the index is built and stored in the device. Likewise, to increase performance even further and since the user's device is assumed to be trustable, the index isn't encrypted. This scenario has high computational power requirements, as the whole system will be running in the user's device.

Middleware as a "proxy service". As in the previous scenario, in this use case the user has high security and performance requirements. However, he wishes to interact with the middleware through a device with low resources (e.g. mobile device, smartphone, etc.). To this end, the middleware's core processing is moved to an auditable proxy service running in a local network. In this case we assume the proxy service is susceptible to passive attacks. As such, the index is encrypted after being built and in-between re-indexing processes that may occur.

Middleware as a Service in the Cloud. In this scenario, the user has very limited resources and cannot afford the deployment and management of a local proxy. As such he whishes to move the middleware's core to a computational Cloud. The user has light performance requirements and can coupe with Internet latency on write and search operations. Security requirements are also lighter and the user acknowledges that the middleware is more vulnerable in a computational Cloud. In order to securely index the private data in the Cloud, a partially homomorphic scheme is applied on the documents. This scheme (more details in section 5.1) allows processing the documents for relevance scoring while preserving their privacy. After the index is built (or updated with dynamic operations), it is ciphered with either traditional encryption or with a homomorphic scheme. The later allows for final ranking scores calculation from the encrypted metrics in the Cloud, but at the cost of performance. The cryptographic keys used to encrypt the index are generated by the client and sent to the Cloud for operations that require encryption/decryption. These keys are periodically be refreshed in order to minimize security risks.

5. Components of the software architecture

This session will discuss in more detail the main components of the reference architecture: the Homomorphic Crypto Module (HCM) and the Indexing and Searching Module. To better understand how these connect and interact with each other, the two final sub-sections will present algorithmic visions of the external API operations, namely the **put()** and **search()** operations.

5.1 Homomorphic Crypto Module

We now present the three schemes conceived and implemented in the homomorphic encryption module (HCM in the reference architecture discussed): random scheme, homomorphic scheme and linear search scheme.

Random Scheme (AES). The most secure encryption scheme used in our solution and with best performance is the random scheme. In this scheme, two equal plaintexts originate different ciphertext with overwhelming (pseudo-random) probability, protecting against *Adaptive Chosen Plaintext Attacks* (IND-CPA). However, due to its probabilistic model, the scheme does not allow any computations over the domain of the encrypted data. These properties make it suitable for encrypting data that will not be subject of further operations, such as the documents or the index when it is being encrypted for persistent storage. In our solution the random scheme is built using a block symmetric ciphering algorithm. In the implemented prototype, discussed in more detail in section 6, the implementation was based on AES on CBC mode, with an adequate ciphering key and random initialization vector.

Homomorphic Scheme (Paillier). The solution uses a partially homomorphic encryption scheme that allows addition over ciphertext blocks. The scheme, which implements the Paillier Cryptographic System [14], is based on demonstrable modular arithmetic properties where the multiplication of the ciphertexts $\varepsilon(x1)$ and $\varepsilon(x2)$ is equivalent to the encryption of the modular addition of their plaintexts, x1 and x2. Equation (1) exemplifies the homomorphic properties of the scheme:

$$\mathbf{E}(X_1)\,.\,\mathbf{E}(X_2) = (g^{X1}r_1^{\ m})\,.\,(g^{X2}r_2^{\ m}) = g^{X1+X2}(r_1r_2)^m = \mathbf{E}(X_1+X_2 \bmod m) \quad (1)$$

In summary, the Paillier cryptosystem is an asymmetric (public key) cryptographic algorithm, with provable security properties, reaching a security level equivalent to Padded RSA and El-Gammal asymmetric cryptosystems (IND-CPA), but providing additive homomorphic encryption properties. In [14] and [15], detailed information on the security and efficiency of the algorithm is discussed, at the level of key-pairs generation and encryption/decryption operations. The expected computational complexity of the encryption operation with the Paillier cryptosystem is proportional to $g^m\ r^n \bmod n^2$, with g the public (encryption) key, n an integer value (obtained from a product of two large prime numbers of equivalent length) and r a random value chosen in Z^*_n. For security observations and in our implementation, the values g, r and n are values with sizes represented with 1024 bits.
Despite that the computational cost seems to be considerable (depending on the size of the integer values used and computed), we will show later, in our evaluation experiments, that the application of the Paillier cryptosystem doesn't aggravate the latency conditions of data searching in the Cloud, even when using the current computational power as provided by nowadays common computers.

Linear Search Scheme (LSS). This scheme shows homomorphic properties in relation to textual data. In more detail, it allows linear scans and searches for patterns over the encrypted data. The cipher text is calculated as authentication codes based on HMACs and a secret used as a Master Key. Using this scheme, a client can produce synthesis as HMAC over the various terms of a document, with the expression identified in (3). Afterwards, the existence of a term in the document can be verified by comparing a transformation over it (2) with the previously generated synthesis.

WordKey(word**)** := HMAC<MasterKey>(word) (2)

E(word**)** := Hash (**WordKey**(word)) XOR
(Salt + HMAC< **WordKey**(word)> (Salt)) (3)

This scheme was first proposed in [9] to support linear scans and is also used in [10]. The expressions (2) and (3) represent the cryptographic functions of the scheme as implemented in [10]. In the presented scheme two equal plaintexts originate different ciphertexts. Although this property is desirable from a security viewpoint, it limits the computations over the ciphertext. For instance, to support the construction of a secure index over the encrypted terms of a collection of documents, allowing the middleware architecture to be run on an uncontrolled environment such as the Cloud while preserving the document's privacy, we need the ability to directly compare two ciphertexts for equality. Following this requirement we propose an adapted scheme that consists on removing the random vector *Salt* and HMAC synthesis from expression (3).

E(word**)** := Hash (**WordKey**(word)) (4)

The proposed scheme (4) is deterministic, as two equal plaintexts generate the same ciphertext. However, the level of security achieved can still be acceptable for the purpose in sight. More specifically, the scheme's security is inherited by the underlying security of the hash function used and by the protection of the MasterKey (which is kept protected in the user's device). The performance of the proposed scheme is also guaranteed by the hash function used. In the resulting scheme, the

Hash function is kept over the HMAC function, as a way to increase the ciphertext distribution and to limit it's length.

5.2 Indexing and Searching Module

A key component in the middleware architecture is the indexing module for support of secure ranked multi-keyword searches over the data encrypted and stored at the Cloud. The indexes built by this model are used to guarantee practical and usable response times for online "real-time" applications.

To search the documents for a set of keywords, an authenticated and authorized user submits a **search()** operation. The system can execute the operation locally (if the index is stored at the middleware) or remotely (if the index is secured at the Cloud). The searched keywords can be encrypted, working as "trapdoors". These trapdoors are then used to directly search on the index (also possibly encrypted), returning as answer the set of relevant files and ranking information. The result of a search (may also come encrypted) is ordered according to the relevance scorings. Despite variations in the possible implementation scenarios (introduced in section 4), no Cloud server ever learns or has knowledge of the metrics used, the searched keywords, the data kept in the index or the documents themselves. To optimize the use of the network, financial costs and latency of accessing the Cloud, it's possible to parameterize the searches in order to return only a certain number of most relevant documents or their identifiers. The following sub-sections will further detail how the indexing and dynamic updates are managed in the proposed solution.

Indexing and Scoring. An index for text data can be efficiently represented as a set of inverted lists [16]. Figure 2 shows an example implemented over a *HashMap*.

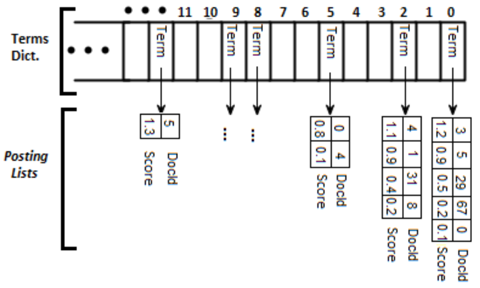


Figure 2. Inverted List Index implemented on *HashMap*

Summarizing, the index is composed by a dictionary that maps each searchable term (that appears in one or more documents) to a list of integer pairs, known as a *PostingList* [16]. Each pair in this list represents a document where the term appears and a score for the corresponding term/document relation that quantifies an indicator of relevance. In our case, we store as score the direct frequency, that is, how many times the term appears in the document. As text is characterized as being sparse data, meaning that a term generally only appears in a fraction of all the documents, each *PostingList* contains only the required size in each case (zero frequency document entries are not stored).

As direct frequency (*tf*) alone may not truly represent the value of a document when part of a whole repository, it's necessary to collect additional, more general metrics to access the effective relevance in a query. Among them we can count the size of each document (L_d), the average size of the collection (L_{ave}), total number of documents (N) and document frequency (*df*, the number of documents that a term appears on). When a search is done, these statistics are grouped in a scoring function that allows us to assess the final ranking of the query. In our solution we used, from the various functions published to this date, the Okapi BM25 scoring function [17] as indicated in (5).

$$\text{BM25(q,d)} = \sum_{t \in q} \log\left[\frac{N}{\text{df}_t}\right] \cdot \frac{(k_1+1)\text{tf}_{td}}{k_1((1-b)+b\times(L_d/L_{\text{ave}}))+\text{tf}_{td}} \quad (5)$$

BM25 was chosen for being a probabilistic function that conjugated the different metrics described and for being one of the most analyzed and used in the scientific community. In the expression, besides the already stated metrics, k_1 and b are tuning parameter constants (in the implemented prototype of section 6, k_1*=1.2* and *b=0.75*). When a query possesses multiple terms, all the corresponding PostingLists are fetched from the index, transformed into final scores using the function and then merged into the final result.

The construction of the index is done using the SPIMI algorithm [18], chosen for its performance, scalability and low memory requirements. Since this algorithm does frequent searches over the PostingLists as it processes the documents, we chose to implement them also as HashMaps. If the index does not fit entirely in memory, during its construction we use the support of secondary memory (as described in [18]) and in the end we use a technique called *ChampionLists* [16]. This technique consists in keeping the whole index in persistent storage and leaving in memory only the top scoring documents for each term, ordered by their relevance. This way searches can be done without the overhead of accessing hard disks. With this technique some precision may be lost, however it is not expected to be substantial nor relevant for the final result.

Dynamic Insertions and Removals. The proposed solution offers a fully dynamic system where documents can be inserted, removed and updated at anytime. However, the indexing module must be able to deal with document insertions and removals while minimizing: (i) costs in the searching functionality and its precision; (ii) threats to the privacy of the extracted metrics and ranking scores.

Dynamic changes can occur in two circumstances: when the index still fits in memory and when it has already been written to disk. While the whole index is still in memory the update is trivial. New documents are processed as usual and resulting *PostingsLists* are merged with the index; removed documents are deleted from the index, as well as its references and metrics. In case the index is stored in disc, the possible retrieval to memory of various partitions of the index in order to update them becomes too expensive. As such, the solution consists in creating an auxiliary index in memory that will hold the new *PostingLists*, as well as a list with identifiers of the removed documents. These structures will co-exist in memory in parallel with the *ChampionLists* referenced previously. Searches are done having in consideration both indexes and filtering the removed documents from the results. Periodically the auxiliary index is merged with the full index in disc, documents referenced in the removal list are effectively deleted from the Cloud (and from the index) and the *ChampionLists* are re-calculated. This "re-indexing" can, however, be carried in background without interrupting executing searches.

To ensure the privacy of the metrics, both the *ChampionLists* and the auxiliary index are kept encrypted at all possible times. In more detail, *ChampionLists* are encrypted immediately after construction and only decrypted at the client when a search is done, while the auxiliary index is kept encrypted between insertions. The maximum size of the auxiliary index affects how much scoring data may be exposed while a new insertion or update is being processed. Cryptographic keys are only persistently stored at the client app and are refreshed periodically.

5.3 Indexing and Writing

In this section we describe the processing of indexing and writing operations when the middleware solution is used as a Cloud service. In this case, previously to sending the documents to the Cloud for storage and indexation, there is a local processing executed by the client application. The client operations are represented in Algorithm 1.

Algorithm 1 – Client processing for Writing/Indexing Documents in the Cloud

```
write (documents):
1  forall (document in documents)
2       encrypted_object = encrypt_AES (document);
3       document_id = put (encrypted_object);
4       searchable_terms = process_document (document);
5       encrypted_terms = encrypt_LSS (searchable_terms);
6       index (document_id, encrypted_terms);
7  reindex();
```

Algorithm 1 is executed when the client wishes to write a group of documents to the Cloud. The algorithm starts by encrypting the documents with the Random (AES) scheme **(line 2)** and transferring them to the Cloud for storage **(3)**. Each document is processed and sent to the Cloud in a serialized way **(1)**, as in this scenario the client's resources are assumed to be small. After a document is securely stored, the client's application "pre-processes" it for indexing **(4)**, that is, stop words are filtered, searchable words are stemmed, punctuation is removed and so on. After this "pre-processing", the resulting terms are ciphered with the LSS scheme **(5)**, one at a time, and the resulting set is transferred to the Cloud for indexing **(6)**. After all the documents have been processed, stored and indexed, the Client issues a re-indexing operation **(7)** that merges all index partitions (built during the previous steps) and builds the final top ranking *ChampionLists* (more details on these two steps will be presented next). In the presented algorithm, the **put()**, **index()** and **reindex()** operations are remote procedures supported by the middleware service running in the Cloud. These operations are processed according to Algorithm 2.

Algorithm 2 – Service in the Cloud: support for Writing and Indexing

```
ObjectId put (encrypted_object):
1  object_id = store_object (encrypted_object);
2  return object_id;

index (document_id, encrypted_terms):
3  relevance_metrics = process_document (encrypted_words);
4  insert_index (relevance_metrics);
5  if (memory_full)
6      ciphered_index_partition = encrypt_AES (index);
7      store_object (encrypted_index_partition);
8      clean_index();

reindex():
9  clean_index();
10 forall (index_partition_id)
11     encrypted_index_partition = get (index_partition_id);
12     index_partition = decrypt_AES (encrypted_index_partition);
13     champion_list = build_champion_list (index_partition);
14     merge_champion_list_with_index(champion_list);
15 encrypt_Paillier (index);
```

In the Cloud, the procedure used for persistent storage is the **put()** operation **(1, 2)**. The **index()** operation is used to index a document from the set of it's searchable terms. This set (encrypted in this case) is processed for relevance metrics extraction **(3, 4)** and periodically, if the memory is full, an index partition is encrypted and persistently stored **(5-8)**. After all the documents of a collection have been processed, stored and indexed, the Client can then issue the **reindex()** operation to merge all the index partitions and build the *ChampionLists* from the top ranking documents **(10-14)**. This operation ends with the encryption of the index with the Homomorphic (Paillier) scheme **(21)**.

This writing procedure, represented by Algorithms 1 and 2, characterizes an initial operation after which the middleware will then start accepting dynamic insertions and updates. It should be noted however that this initial procedure is not mandatory, that is, the middleware can start operating in dynamical mode from an empty index and accept the same group of documents dynamically, although the costs in performance will be higher.

5.4 Searching

After the insertion of documents, ranked searches can be executed by issuing a set of keywords. Algorithm 3 exemplifies the operation executed in the client side.

Algorithm 3 – Client searching by keywords

```
search (keywords):
1  query_terms = process_keywords (keywords);
2  encrypted_keywords = encrypt_LSS (query_terms);
3  encrypted_final_scores = search (encrypted_keywords);
4  ranked_scores = decrypt_Paillier (encrypted_ranked_scores);
5  ordered_ranked_scores = order_ranked_scores (ranked_scores);
```

The client starts by processing and encrypting the keywords with the LSS scheme, in order to preserve the privacy of the query **(1, 2)**. Then the encrypted keywords (or terms) are sent to the Middleware service running in the Cloud, in order to perform the search **(3)**. The Cloud service returns encrypted ranked scores, which the client decrypts and sorts to obtain the ids of the top-ranking document for the issued search **(4, 5)**. Algorithm 4 represents the searching processing as executed by the middleware solution running in the cloud.

Algorithm 4 – Service in the Cloud: support for the searching operation

```
IndexEntries search (encrypted_keywords):
1  relevant_index_entries = access_index (encrypted_keywords);
2  encrypted_ranked_scores = calculate_ranking_scores
(relevant_index_entries);
3  return encrypted_ranked_scores;
```

To execute a client search operation, the middleware service at the Cloud consults the index to find the relevant entries **(1)**, calculates the final scores from the encrypted entries **(2)** and returns them to the Client **(3)**.

6. Evaluation

We have implemented a prototype of the proposed solution, in order to evaluate different aspects of the middleware system. The preliminary evaluation we now present focused primarily on performance criteria and latency considerations. We will extend the observations of other important dimensions, such as reliability, resilience and scalability conditions, as future-work.

The implementation of the prototype is based on the Java language and libraries. The results were obtained using a *OpenJDK 6 JVM*[1], with 6 GB of initially reserved and limited memory, *Concurrent Garbage Collector*[2] activated and executing

[1] http://openjdk.java.net/projects/jdk6/

[2] http://docs.oracle.com/javase/6/docs/technotes/guides/vm/cms-6.html

in a PC with *Intel Core i3 3.4Ghz* processor. The chosen Cloud provider was Amazon (AWS)[1], Ireland data-center. Amazon *S3* is used as storage service and Amazon EC2 as computational Cloud service, configured with a Large Instance. The version of the Amazon Java SDK used is 1.3.2. The connection to the Cloud was limited to 30 Mb/s for downloads and 3 Mb/s for upload. We evaluated the system with a dataset comprised of different versions of Wikipedia pages. The versions chosen were the English Wikipedia dump of May and Spanish and Portuguese dumps of October 2012[2], with uncompressed sizes of 37.9, 7.2 and 4.4 GB respectively.

In order to experimentally validate the conceived solution and the implemented prototype, different load tests were carried. Figure 3 shows the first group of tests.

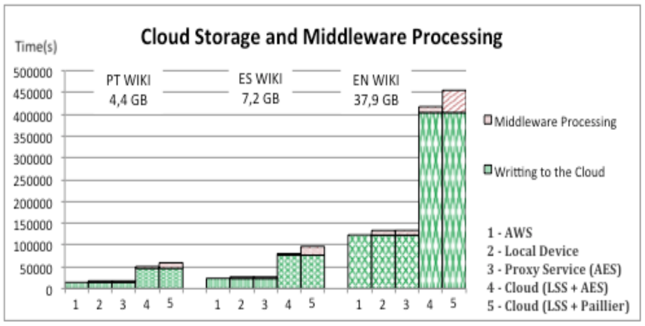


Figure 3. Performance of writing and indexing in the different scenarios

This first group of tests aimed at measuring the performance of the middleware processing when compared to the direct storage in the Cloud. To increase the research interest of the comparison, a secure solution from Amazon, based on *Client-Side Encryption*[3], was used (referred to in the graphs as AWS). At the time of writing, the solution is based on encrypting the data at the client with AES before sending it to Amazon Cloud servers, while the mater encryption key is stored and managed by the client. As far as we know, this is the only solution promoted by the provider to secure the client's data. On the other hand, the solution's only concern is to protect the data, and no operations are possible on the data after it's encryption (including searching).

In the figure, results are divided between datasets and between implementation scenarios. The implementation scenarios are in the same order as described in section 4 and the last use case (middleware as service in the Cloud) is divided in two, according to a small implementation option. That option consists in either encrypting the index with the Random (AES) scheme or with the Homomorphic (Paillier) scheme. However, this small variation has high impacts on the obtained results, as will be seen next.

The results presented by Figure 3 show that the overhead introduced by the middleware is minimal when compared to the cost of transferring the same collection of documents to the Cloud. On the other hand, if we analyze in more detail the two use cases entirely based on the Cloud (the rightmost bars in each of the 3 datasets), we can conclude that they require writing more data to the Cloud than the other tests. This requirement comes from the need to send the collection of documents twice to the Cloud: once ciphered with the Random scheme, for storage purposes, and a second time ciphered with the Line Search Scheme, for indexing and scoring. This is necessary because the LSS scheme is irreversible, as it is based on Hashing and HMAC techniques. As a future research direction we intend to conceive a LSS scheme based on symmetric encryption, that is, a reversible scheme. Such a scheme, while preserving the performance of the current implementation, wouldn't require duplication of data and consequently would save on Cloud storage costs and increase the overall performance of the presented solution.

Entering in detail into the middleware processing, we can analyze and compare the performance of its main processes. Figure 4 represents these tests. In the figure, document processing refers to the act of processing a document for the first time and is only done once for each document; indexing refers to the re-indexing operation and building of *ChampionLists* from persistently stored index partitions; and Security refers to all processes of encryption and decryption with the different schemes.

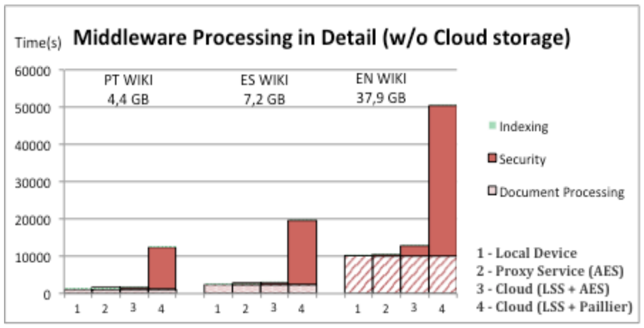


Figure 4. Performance of the middleware processes in the different scenarios

Analyzing Figure 4 in detail and the performance of the middleware processes, we conclude that using conventional security (Random scheme) to protect the privacy of the index adds relatively little latency to the overall performance. This is shown by the security cost in the "Local Proxy Service" scenario. On both Cloud scenarios, security overhead is slightly increased due to the necessity of ciphering each word of each document independently with the LSS scheme. Additionally, using the Paillier scheme to encrypt the index has a very high penalty on performance. For example, the Portuguese data set test in the Cloud AES scenario takes around 19 minutes, while in the Cloud Paillier scenario it takes around 3 hours. However, the cost of using the Paillier scheme seems to amortize with the increase in data size. Comparing the English and Spanish Wikipedias (7.2 and 37.9 GB respectively), the data size increases in around 5 times but the performance cost in this scenario only increases in around 2.3 times. The third and final group of tests aimed at comparing the performance of searching in the different implementation and operation scenarios. Figure 5 presents the results.

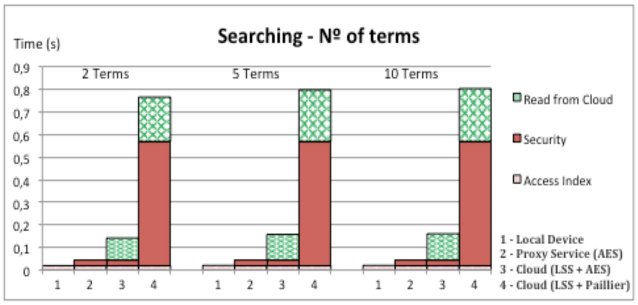


Figure 5. Search performance in the different scenarios

The results shown were obtained using the 37,9 GB collection of English Wikipedia documents. In all scenarios searches are very fast and always done under the maximum threshold in usability for online applications, which is considered to be 1 second. However, there are visible differences between the

[1] http://aws.amazon.com/
[2] http://en.wikipedia.org/wiki/Wikipedia:Database_download
[3] http://aws.amazon.com/articles/2850096021478074

scenarios. Comparing the first two use cases, we can see that direct access to the index is very fast and that traditional security has very low cost on the search performance. Also, in both scenarios searches are always done under the 100 milliseconds mark. When comparing these results to the scenarios that use the Cloud, latency is increased as access to the middleware has to be done through the Internet. This increase is due to Internet latency and network traffic, and should be expected in any implementation scenario where a search has to request information from a remote site.

Analysis of the Cloud Paillier scenario shows that the increase in security latency is very high. This is due to the natural cost of decrypting index entries encrypted with the Homomorphic scheme. The time it takes to retrieve results from the Cloud also increases as the ciphertext size of this scheme is bigger than the ciphertext size of the Random scheme. To conclude results show that, on one hand, performance is preserved as query size increases and, on the other hand, results are maintained as data set size increases (due to the use of *Champion Posting Lists*).

7. Conclusions

The article presents a solution that has in sight the conjugation of dependability, security and privacy requirements of data stored in Internet Storage Clouds. The solution is addressed as a middleware system for intermediation of secure storage services for private data in storage Clouds. The presented system supports the management and storage of private data, under full control of the user, and allows searching over the encrypted data, independently of different Clouds that may be used. A relevant contribution of the proposed solution focuses on the support of secure searches over the data, addressing a solution for ranked multi-keyword searches. The solution achieved allows the use of effective mechanisms for searching over the private documents with multiple keywords and accessing the encrypted information in the Cloud, based on scoring/ranking operations on the relevance of the data. During the search operations, privacy conditions are preserved under full control of the users. The presented approach uses cryptographic schemes that explore homomorphic encryption techniques combined with dynamic indexing mechanisms. The implementation of the proposed system and its evaluation shows that the solution is viable, offers more security and greater user control (comparing to a solution promoted by Amazon AWS) and does not aggravate conditions of access latency and data availability.

References

1. P. Mell and T. Grance, "Draft nist working definition of *Cloud* computing," Referenced on Jan. 23rd, 2010 Online at http://csrc.nist.gov/groups/SNS/*Cloud*-computing/index.html, 2010.
2. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of *Cloud* Computing," Communications of the ACM 53 (4) 50-58, April 2010.
3. Privacy Rights Clearinghouse. Chronology of data breaches. http://www.privacyrights.org/data-breach.
4. S. Kamara and K. Lauter, "Cryptographic *Cloud* storage," in Proceedings of Financial Cryptography: Workshop on Real-Life Cryptographic Protocols and Standardization 2010, January 2010.
5. J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Proceedings of the 17th Usenix Security Symposium, San Jose, CA, July–August 2008.
6. J. Li, M. Krohn, D. Mazi`eres, and D. Shasha. Secure untrusted data repository (SUNDR). In Proceedings of the 6th Symposium on Operating Systems Design and Implementation, pages 91–106, San Francisco, CA, December 2004.
7. A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten. "SPORC: Group collaboration using untrusted *Cloud* resources". In Proceedings of the 9th Symposium on Operating Systems Design and Implementation, Vancouver, Canada, October 2010.
8. *Cloud* Security Alliance, "Security guidance for critical areas of focus in *Cloud* computing," 2009, http://www.*Cloud*securityalliance.org.
9. D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy'00*, 2000.
10. R. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan. *CryptDB: Protecting Confidentiality with Encrypted Query Processing*. SOSP '11, October 23–26, 2011, Cascais, Portugal.
11. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYP'04, volume 3027 of LNCS*. Springer, 2004.
12. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS'06*, 2006.
13. Wang, C., Cao, N., Ren, K., & Lou, W. (2012). Enabling secure and efficient ranked keyword search over outsourced cloud data. Parallel and Distributed Systems, IEEE Transactions on, 23(8), 1467-1479.
14. P. Paillier. *Public-key cryptosystems based on composite degree residuosity classes*. In Proceedings of the 18th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Prague, Czech Republic, May 1999.
15. P. Paillier, D. Pointcheval. "Efficient Public-Key Cryptosystem Provably Secure against Active Adversaries". In *Asiacrypt '99*, LNCS 1716, pages 165-179. Springer-Verlag, Berlin, 1999.
16. C. Manning, P. Raghavan, H. Schütze. "An Introduction to Information Retrieval", Cambridge University Press, 2009
17. Spärck Jones, Karen, S.Walker, and Stephen E. Robertson. 2000. A probabilistic model of information retrieval: Development and comparative experiments. *IP&M* 36(6): 779–808, 809–840.
18. Heinz, Steffen, and Justin Zobel. 2003. Efficient single-pass index construction for text databases. *JASIST* 54(8):713–729. DOI: dx.doi.org/10.1002/asi.10268
19. Cao, N., Wang, C., Li, M., Ren, K., & Lou, W. (2011, April). Privacy-preserving multi-keyword ranked search over encrypted cloud data. In INFOCOM, 2011 Proceedings IEEE (pp. 829-837). IEEE.
20. Liu, Qin, Chiu C. Tan, Jie Wu, and Guojun Wang. "Efficient information retrieval for ranked queries in cost-effective cloud environments." In INFOCOM, 2012 Proceedings IEEE, pp. 2581-2585. IEEE, 2012.