J Supercomput
DOI 10.1007/s11227-012-0829-z

Privacy-aware searching with oblivious term matching for cloud storage

Zeeshan Pervez · Ammar Ahmad Awan · Asad Masood Khattak · Sungyoung Lee · Eui-Nam Huh

Abstract Encryption ensures confidentiality of the data outsourced to cloud storage services. Searching the encrypted data enables subscribers of a cloud storage service to access only relevant data, by defining trapdoors or evaluating search queries on locally stored indexes. However, these approaches do not consider access privileges while executing search queries. Furthermore, these approaches restrict the searching capability of a subscriber to a limited number of trapdoors defined during data encryption. To address the issue of privacy-aware data search, we propose Oblivious Term Matching (OTM). Unlike existing systems, OTM enables authorized subscribers to define their own search queries comprising of arbitrary number of selection criterion. OTM ensures that cloud service provider obliviously evaluates encrypted search queries without learning any information about the outsourced data. Our performance analysis has demonstrated that search queries comprising of 2 to 14 distinct search criteria cost only 0.03 to 1.09 $ per 1000 requests.

Z. Pervez · A.A. Awan · A.M. Khattak · S. Lee (✉)
Ubiquitous Computing Lab, Department of Computer Engineering, Kyung Hee University, Global Campus, 1 Seocheon-dong, Giheung-gu, Yongin-si, Gyeonggi-do 446-701, South Korea
e-mail: sylee@oslab.khu.ac.kr

Z. Pervez
e-mail: zeeshan@oslab.khu.ac.kr

A.A. Awan
e-mail: ammar@oslab.khu.ac.kr

A.M. Khattak
e-mail: asad.masood@oslab.khu.ac.kr

E.-N. Huh
Internet Computing and Network Security Lab, Department of Computer Engineering, Kyung Hee University, Global Campus, 1 Seocheon-dong, Giheung-gu, Yongin-si, Gyeonggi-do 446-701, South Korea
e-mail: johnhuh@khu.ac.kr

Published online: 10 October 2012

Keywords Cloud storage · Data search · Private matching · Private information retrieval

1 Introduction

Cloud computing is an epitome of on-demand and scalable computing. It provides virtualized computing resources as services over the Internet, which can be subscribed on the need basis [1]. Subscribers of cloud services do not need to manage their own computing resources. They can simply harness compute resources of a public cloud, owned by a Cloud Service Provider (CSP). This frees the subscribers from expensive and expertise-intensive in-house IT resource management [2]. CSP provides an abstraction of unlimited processing power and storage facility, which can be subscribed as a pay-as-you-use subscription model; leveraging the subscribers to pay only for what they have consumed.

With the amount of digital data doubling almost every eighteen months, cloud storage provides a cost-effective solution to deal with the ever-increasing demand of storage facility [3]. It provides raw storage as a service, accessible through high-speed network, which can be scaled up or scaled down accordingly. Subscribers of cloud storage, including both individuals and enterprises, can keep their data for much longer time without the concerns of reliability and availability of the outsourced data. The advent of cloud storage has brought forth data backup, synchronization, sharing and collaborative services which have certainly changed the way we manage and interact with the outsourced data [4].

Subscribers of a cloud storage are charged for the consumed storage facility and amount of data accessed on each data access request [5, 6]. To avoid needless data access requests, and to minimize network traffic, subscribers need to access only relevant outsourced data. Thus a cloud storage that leverages its subscribers to search the outsourced data is highly valuable, especially for the enterprises, where huge data are outsourced and relevant data can only be accessed through search. However, cloud storage is owned and managed by an independent CSP, and there is a great risk of privacy infringement when confidential data are stored on such services [7, 8]. Unauthorized subscribers can model search queries that can be used to learn confidential information about the outsourced data. Suppose, subscriber of a cloud storage is a patient who has shared her medical report with a doctor. A malicious subscriber (attacker) can model a search query with keywords (i.e., diabetes mellitus, cerebrovascular accident) and can search for the documents matching the search criteria, consequently learning about the disease patient is suffering with.

To ensure data confidentiality, often encrypted data are outsourced to cloud storage [9]. Certainly, encryption achieves data confidentiality; however, it restrains searching capability of subscribers, as search criteria of a standard lookup query cannot be evaluated for the encrypted data [10]. To access particular data, subscribers either have to download the entire outsourced data,[1] locally decrypt it and then execute

[1] We assume that encrypted data are outsourced to a cloud storage. For simplicity we refer encrypted data residing within untrusted domain of a cloud service provider as outsourced data.

search query over plain text, or rely on CSP to search cloud storage. Clearly, downloading entire outsourced data is not a feasible solution, as it would generate needless data access requests, and unnecessarily consume network bandwidth. Whereas, enabling CSP to search for particular data would compromise privacy of the outsourced data. CSP would need auxiliary information (i.e., keywords, inverted index) about the outsourced data to identify the data, which fulfils the selection criteria. However, this auxiliary information can leverage malicious CSP to learn confidential information about the outsourced data, even though it is in encrypted format.

The problem of searching encrypted data outsourced to an untrusted domain of CSP is manifold. First, the modelling of search query that can be used to search cloud storage. Second, the privacy of standard search query, which do not reveal confidential information about the outsourced data. Third, enforcement of access control policies while searching cloud storage. Considering the issues of conventional searching methodologies, search queries on cloud storage should be executed privately and must conform to access control polices. Private execution of search queries ensures that CSP cannot learn any information about the outsourced data. Conformance to access control policies certifies that search queries of malicious subscribers are never executed successfully even with the assistance of a CSP.

Existing systems tend of achieve searching capabilities within cloud storage by employing methodologies of search over encrypted data [11, 12]. However, these methodologies rely on assumptions which are either not viable in cloud ecosystem or greatly affect the utility of a cloud storage. The most common approach to search encrypted data is to define trapdoors for individual keywords that are associated with the encrypted data [13, 14]. These trapdoors are then distributed among authorized subscribers enabling them to search encrypted data. However, this approach restrains query modelling capabilities of a subscriber to a limited number of valid trapdoors. Besides this, systems utilizing trapdoor-based approach fail to enforce access control policies during the execution of search queries.

Enterprise search products employ slightly different methodology than the systems based on trapdoor-based approach [15, 16]. These products create searchable index from the data that need to be searched. Searchable indexes are stored locally and data are outsourced to cloud storage after employing suitable encryption. Search queries are executed locally over searchable indexes. This ensures that CSP cannot learn any information about the outsourced data, as search queries are executed within the trusted domain of an enterprise. Although, these products ensure data privacy; however, they greatly reduce utility of cloud storage as searchable indexes are need to be managed locally, which requires detected computing resources.

To ensure privacy of the outsourced data while executing search queries, we propose oblivious search for cloud storage called Oblivious Term Matching (OTM). It utilizes homomorphic encryption [17] to ensure that search queries are obliviously evaluated, preventing the exploitation of search queries by a CSP. We utilize proxy re-encryption [18] to ensure authorized data search. It restrains malicious subscribers to team up with a CSP and learn any information that can be used to compromise privacy of the outsourced data. OTM is not limited to a number of trapdoors, as it utilizes index-based oblivious term matching. It enables us to associate arbitrary number of keywords with the outsourced data. Authorized subscribers define their own search queries, which can be obliviously evaluated by a CSP.

OTM can be regarded as a value-added privacy-aware search service that enhances the utility of cloud storage. To demonstrate viability of OTM, we implemented a search service for Google's cloud ecosystem (i.e., Google App Engine [19], Google Docs [20] and Google Datastore [21]). The performance analysis of our proposed privacy-aware search for cloud storage demonstrated that search queries comprised of 2 to 14 distinct search criteria can be modelled within 0.001 to 0.037 ms. Besides this, these search queries can be evaluated within 639 to 19554 ms by Google App Engine (CSP).

In general, with OTM we make the following contributions in the area of searchable cloud storage services:

- Oblivious data search for encrypted data to ensure that CSP cannot learn any information from the search query. Moreover, CSP is unable to identify the outsourced data that satisfy the selection criteria.
- Privacy-aware data search,[2] which ensures that only authorized subscriber can search the outsourced data.
- Assorted queries, which ensure that CSP cannot relate search queries submitted by multiple subscribers even if they are searching for similar data contents.
- A search service that enables authorized subscribers to model their own search queries instead of relying on trapdoors defined prior to data outsourcing.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the cryptographic primitives and protocols which are utilized by OTM. Section 4 outlines the models, design goals, and assumptions. Sect. 5 presents OTM. Section 6 discusses the implementation details of our proposed privacy-aware search for cloud storage. Section 7 presents the evaluation results. Section 8 discusses the security aspects of OTM. Finally, Sect. 9 concludes the paper along with the future directions.

2 Related work

We categorize OTM as a service that provides privacy-aware search rather than a cryptosystem, which encrypts the data in such a way that keywords can be matched without the need to decrypt the cipher text. Through OTM, we propose a privacy-aware search for cloud storage that is independent of underlying cryptographic algorithms which ensure confidentiality of the outsourced data. Throughout this section, we focus on the efficacy of the existing systems which provide searching capabilities over the encrypted data. By efficacy of a system we mean that how system affects the utility of cloud storage.

Searchable symmetric key cryptography (SKC) was first proposed by Song et al. [11], making it possible to search for a particular keyword from the encrypted

[2] Privacy-aware data search is realized by distributing appropriate cryptographic keys to authorized subscribers. Inaccessibility to these cryptographic keys restrains capabilities of unauthorized subscribers to search cloud storage and deduce any information about the outsourced data even if they collude with cloud service provider.

data by using trapdoors. Based on SKC various schemes have been developed which utilize it to search encrypted index, instead of the encrypted data [22–24]. Followed by SKC, first practical searchable public key cryptography (PKC) was proposed by Boneh et al. [12]. PKC enables untrusted server to perform search over the encrypted data concealed with public key, without the need to reveal actual decryption key (private key). Both SKC and PKC utilize trapdoors to execute search queries. In cloud-based data sharing and collaborative services both SKC and PKC greatly affect the utility of a service, as search queries are limited to number of trapdoors. In addition, these trapdoors need to be distributed among the entities who want to search the encrypted data. Trapdoors are further subject to change on any modification in the outsourced/shared data; thus they need to be redistributed on each update. From privacy perspective, SKC and PKC fail to enforce access control policies while searching encrypted data.

Li et al. in [13] proposed authorized private keyword search (APKS) on encrypted Personal Health Records (PHR) by using Hierarchical Predicate Encryption (HPE). In their construction of privacy-aware search, a Trusted Third Party (TTP) was responsible for distributing capabilities (trapdoors). Users obtained capabilities from a TTP, according to their access privileges and then submit trapdoors to a CSP. Using trapdoors to model search queries, and relying on TTP to achieve privacy-aware search greatly affects the utility of cloud storage, as authorized subscribers can only model search query using the trapdoors provided by the TTP. Wang et al. [14] proposed a secure ranked search over encrypted data, for data residing within the untrusted domain of CSP. Their proposed system only supports single keyword based search queries. Thus, for an enterprise system outsourcing petabytes of data to cloud storage, their proposed system simply lack realism. Search query with a single keyword cannot define complex selection criteria to achieve efficient searching capabilities. CS2 [25] provided symmetric searchable encryption (SSE) with search authentication (SA). CS2 utilized inverted index to search encrypted data, along with dynamic data updates. However, CS2 was limited to personal storage system and did not allow client to share files.

Enterprise search products, like Google search appliance [15] and Windows enterprise search [16], provide document indexing functionality. Enterprises can use these products to query document repositories within their data centre and over cloud storage (public cloud) as well. These products create a single enterprise wide centralized index, which can be queried and then the results of search queries are filtered out according to the access control policies. As these systems enforce access control policies at query time, they require search services to be hosted within the enterprise, thus obstructing migration of an enterprise to a cloud ecosystem as it would have to manage centralized index by using its compute resources. Research concluded in [26] has shown that by carefully modelling search queries malicious users can learn valuable information from the centralized index, even if they do not have access to the data residing within enterprise's data centre or in a cloud storage.

3 Preliminaries

Before elaborating the proposed oblivious search for cloud storage, we introduce some of the preliminaries used in its development.

3.1 Homomorphic encryption

A cryptographic scheme is said to be homomorphic if its encryption function $\mathcal{E}_H$ holds the property $\mathcal{E}_H(x) * \mathcal{E}_H(y) = \mathcal{E}_H(x + y)$. A homomorphic encryption is said to be *semantically secure* if $\mathcal{E}_H$ reveals no information about x and y, hence it is computationally infeasible to distinguish between the cases $x \neq y$ and $x = y$ [27].

Public key encryption scheme proposed by Pascal Paillier [17] is additively homomorphic, and consists of subsequent fundamental algorithms.

3.1.1 Key generation

Let p and q be two large primes and $n = p.q$. The Euler's totient function is denoted by $\phi(n)$. The Carmichael's function is represented by $\lambda(n)$. For n, the product of two primes, $\phi(n) = (p-1)(q-1)$ and $\lambda(n) = \text{lcm}(p-1, q-1)$. These two functions exhibit the following properties over the multiplicative group $\mathbb{Z}^*_{n^2}$:

$$\left|\mathbb{Z}^*_{n^2}\right| = \phi\left(n^2\right) = n.\phi(n) \tag{1}$$

and for any $\omega \in \mathbb{Z}^*_{n^2}$,

$$\omega^{\phi(n)} = 1 \pmod{n} \tag{2}$$

$$\omega^{n\phi(n)} = 1 \pmod{n^2} \tag{3}$$

Public key $\mathcal{PK}$ is defined as (n, g), where g is an element of $\mathbb{Z}^*_{n^2}$, and a secret key $\mathcal{SK}$ as $\lambda(n)$.

3.1.2 Encryption

To encrypt a message $m \in \mathbb{Z}_n$, randomly choose $y \in_R \mathbb{Z}^*_{n^2}$ and define an encryption function $\mathcal{E}_H$ such that

$$\mathcal{E}_H : \mathbb{Z}_n \times \mathbb{Z}^*_n \mapsto \mathbb{Z}^*_{n^2} \tag{4}$$

$$\mathcal{E}_H(m, y) = g^m y^n \pmod{n^2} \tag{5}$$

3.1.3 Decryption

To decrypt the ciphertext, L is defined as $(u-1)/n, \forall u \in \{u | u = 1 (\text{mod } n)\}$. Ciphertext c can be decrypted by using secret key $\mathcal{SK} = \lambda(n)$, $\mathcal{D}_g$ as

$$\mathcal{D}_H\left(c, \lambda(n)\right) = \frac{L(c^{\lambda(n)} (\text{mod } n^2))}{L(g^{\lambda(n)} (\text{mod } n^2))} \tag{6}$$

3.1.4 Homomorphic operation

Arithmetic addition between the ciphertexts $c_1 = \mathcal{E}_H(m_1, y_1)$ and $c_2 = \mathcal{E}_H(m_2, y_2)$ is obliviously computed as:

$$\mathcal{E}_H(m_1, y_1) = g^{m_1} {y_1}^n \left(\text{mod } n^2\right)$$
$$\mathcal{E}_H(m_2, y_2) = g^{m_2} {y_2}^n \left(\text{mod } n^2\right)$$

$$\begin{aligned} \mathcal{E}_H(m_1, y_1).\mathcal{E}_g(m_2, y_2) &= g^{m_1+m_2}(y_1.y_2)^n \left(\text{mod } n^2\right) \\ &= \mathcal{E}_H(m_1 + m_2) \end{aligned} \tag{7}$$

3.2 Private matching

Private matching (PM) [28] is a value matching protocol. It assists two interactive entities to compute set intersection over their private set of values, without revealing any element of their private set to each other. It uses homomorphic encryption to identify the commonalities among the private sets, while ensuring privacy of each set.

Suppose there is a client $\mathcal{C}$ and a server $\mathcal{S}$. $\mathcal{C}$ has its own private set of values $\mathcal{X} : \{x_1, x_2 \ldots x_n\}$, and so does $\mathcal{S}$, $\mathcal{Y} : \{y_1, y_2 \ldots y_n\}$. $\mathcal{C}$ wants to compute set intersection with $\mathcal{S}$ over the private set of values (i.e. $\mathcal{X}, \mathcal{Y}$). However, $\mathcal{C}$ does not want to seep out any information about $\mathcal{X}$, with an exception of set cardinality. To identify the commonalities between $\mathcal{X}$ and $\mathcal{Y}$, $\mathcal{C}$ computes a polynomial (see (8)) whose roots are members of $\mathcal{X}$:

$$P(x \in \mathcal{X}) = (x - x_1)(x - x_2) \ldots (x - x_n) = \sum_{i=0}^{n} \alpha_i x^i \tag{8}$$

$\mathcal{C}$ then sends the homomorphically encrypted coefficients $(\hat{\alpha}_{0\ldots n})$ of $P(x)$ to $\mathcal{S}$. By using $\hat{\alpha}$, $\mathcal{S}$ evaluates $P(y)$ for every element of its private set. It then computes oblivious value by multiplying evaluated $P(y)$ with a random number r and adding it to y, i.e. $\mathcal{E}_H(r.P(y) + y)$, where $\mathcal{E}_H$ is a homomorphic encryption algorithm. These oblivious values are then send to $\mathcal{C}$ for decryption. At $\mathcal{C}$, the decryption of an oblivious value results in y, if $P(y)$ computed by $\mathcal{S}$ is evaluated at z, such that $\langle z \subseteq \bigcap | (z \in \mathcal{X}) \wedge (z \in \mathcal{Y}) \rangle$. Otherwise, $\mathcal{C}$ ends up generating a random value. At the end of this protocol, $\mathcal{C}$ learns only the intersection set; whereas, $\mathcal{S}$ ascertains nothing more than the cardinality of $\mathcal{X}$.

3.3 Proxy re-encryption (PRE)

Proxy Re-Encryption (PRE) is a cryptographic primitive, which transforms the ciphertext from one secret key to another without revealing the secret key to a semi-trusted party [18]. Through PRE, ciphertext encrypted with Alice secret key can be transformed to another ciphertext, which Bob can decrypt without revealing any information to the intermediary (*semi-trusted server*). PRE consists of four fundamental algorithms: Key Generation, Encryption, Re-Encryption, and Decryption. Suppose Alice wants to send a message m to Bob through an intermediary server by using PRE; the following are the steps which will be executed.

3.3.1 Key generation

Alice first selects a Bilinear Group $\mathbb{G}$ of prime order q with g generator. Two random numbers a and b of order q are generated. Then a and b are used to generate respective secret keys $\mathrm{SK}_a = a$ and $\mathrm{SK}_b = b$. Consequently, public keys are produced as $\mathrm{PK}_a = g^a$ and $\mathrm{PK}_b = g^b$. Once public keys are defined, Alice selects a random number $r \in \mathbb{Z}^*{}_p$, along with a Bilinear Map of $\mathbb{G}$ as $Z = e(g, g)$. Finally, proxy-key is generated as $\mathrm{RK}_{a\to b} = (g^b)^{1/a}$ and is handed over to a semi-trusted server responsible for ciphertext transformation.

3.3.2 Encryption

In order to encrypt message m, with Alice public key, ciphertext is computed as $C_a = (Z^r.m, g^{ra})$.

3.3.3 Re-encryption

This step is executed by a semi-trusted server. Ciphertext is transformed from $C_a \to C_b$ by using proxy-key $\mathrm{RK}_{a\to b}$:

$$C_b = \left(Z^r.m, e\left(g^{ra}, \mathrm{RK}_{a\to b}\right)\right) = \left(Z^r.m, e\left(g^{ra}, g^{b/a}\right)\right) = \left(Z^r.m, Z^{rb}\right) \tag{9}$$

3.3.4 Decryption

To decrypt the ciphertext C_b, Bob uses his secret key SK_b, communicated to him by Alice through secure means, i.e. *SSL*. Message m can be obtained as $m = \frac{Z^r.m}{(Z^{rb})^{1/b}}$

4 Models, design goals, and assumptions

4.1 System model

To realize a cloud storage with privacy-aware searching capabilities over outsourced data, cloud service provider, data owner, data consumer, and trusted third party are considered as the involved entities. For brevity, we shall refer to them as cloud server, owner, user, and third party, respectively. Cloud server provisions storage and computes facilities on subscription basis. Owner owns the confidential data that need to be shared with users. Authorized users can access and search the cloud storage with respect to their access privileges. Third party transforms the search criteria submitted by a user to an oblivious search query. Cloud server evaluates the oblivious queries without learning any information about the search criteria and the outsourced data. Besides this, cloud server cannot identify the files which contain the keywords user is looking for.

4.2 Security model

Privacy of cloud storage is ensured if cloud server cannot learn or extract any information about the outsourced data, from the data itself, and the index that leverages authorized users to search the cloud storage. In short, cloud server should obliviously execute search queries submitted by a user. Oblivious query execution ensures that privacy of the search query is preserved along with the outsourced data. In addition, it ensures that cloud server cannot identify the files which contain the keywords specified by the search criteria. To compromise privacy of the outsourced data, cloud server can team up with malicious users. By teaming up, cloud server and malicious users try to learn confidential information about the outsourced data, from the queries submitted by an authorized user.

4.3 System design goals

Searching is an important feature of storage systems. When data is stored in an untrusted domain (i.e., cloud server), search queries can reveal confidential information about the outsourced data. Privacy of outsourced data can be effortlessly ensured by encrypting it with an appropriate encryption algorithm. However, searching outsourced data is of major concern, as cloud server cannot map standard search queries to encrypted data. Besides this, cloud server can learn confidential information about the outsourced data by manipulating the search query submitted by a user. The pivotal design goal of our proposed oblivious search for cloud storage is to assure that authorized users can search outsourced data without revealing search query to the cloud server. Additionally, the cloud server should not be able to learn the result of search query execution and relate search queries of different users searching for identical data contents.

4.4 Assumption and notation

Oblivious Term Matching (OTM) focuses on enabling privacy-aware data search for cloud storage systems—we intentionally neglected the details of data sharing from security and privacy point of view. Readers may refer to [29] for more details on efficient and secure data sharing in cloud storage. Table 1 illustrates the notation that we use to explain the core concept of our proposed oblivious search for cloud storage.

$\mathcal{F}$ represents the file outsourced to cloud storage, searchable to authorized subscribers. $\mathcal{I}$ represents the inverted index over $\mathcal{F}$. It contains frequently occurring keywords in $\mathcal{F}$ along with their frequency count. $\mathcal{H}$ is a publicly known encoding function, which encodes keywords into integer value of arbitrary size. $\mathcal{E}_P$, $\mathcal{D}_P$ and $\mathcal{T}_P$ are proxy re-encryption functions. These functions realize privacy-aware data search and ensure that cloud service provider need to persist single copy of encrypted inverted index for each outsourced file. $\mathcal{E}_H$ and $\mathcal{D}_H$ are homomorphic encryption functions—for realizing secure computation on encrypted data, i.e., inverted index and search query. $\mathcal{E}_A$ and $\mathcal{D}_A$ are asymmetric encryption functions that ensure that only authorized subscribers can search cloud storage. $\alpha_{0\ldots n}$ are polynomial coefficients that are used to privately match encrypted search query and inverted index obliviously. $\Delta_{y_{0\ldots n}}$ represent oblivious values that are obtained as a result of private matching protocol. $\psi_{0\ldots n}$ represent the query evaluation result retrieved by trusted third party.

Table 1 Notation used in the descriptive detail of OTM

Notation	Description
$\mathcal{F}$	Confidential file that needs to be shared.
$\mathcal{I} = \{(kw_0, f_0), \ldots, (kw_n, f_n)\}$	Inverted index file that contains n keywords along with their frequency in $\mathcal{F}$.
$\mathcal{H}$	Public encoding function which encodes an arbitrary string to an integer value of q modulo, where q is a large prime.
$\mathcal{E}_P, \mathcal{D}_P, \mathcal{T}_P$	Proxy re-encryption, decryption, and ciphertext transformation algorithms.
$\omega_o, \omega_u, \omega_{o \to u}$	Proxy re-encryption keys for the owner, user and cloud server, respectively.
$\mathcal{E}_H, \mathcal{D}_H$	Homomorphic encryption and decryption algorithms.
$\sigma_{\text{pk}}, \sigma_{\text{sk}}$	Public and secret keys for homomorphic encryption algorithm.
$\mathcal{E}_A, \mathcal{D}_A$	Asymmetric encryption and decryption algorithms.
$k_{\text{pub}}, k_{\text{pri}}$	Public and private key pair for asymmetric encryption algorithm.
$\alpha_{0\ldots n}$	List of coefficients of a polynomial P that defines the search query.
$\Delta_{y_{0\ldots n}}$	Oblivious values: query execution result by a cloud server provider.
$\psi_{0\ldots n}$	Query execution result computed by the user from the oblivious values ($\Delta_{y_{0\ldots n}}$).
sk	Third party secret to conceal keyword frequency in inverted index.

5 Proposed system

We first briefly present the main idea of privacy-aware data search for cloud storage. Then we describe the details of searching cloud storage with oblivious term matching.

5.1 Main idea

Suppose, Datamine is an advisory firm which provides market analysis and trend discovery services to its customers. Alice is a director of research at Datamine. She is working on two distinct projects. One of the project mines social networks' data to devise effective advertisement campaigns. Whereas, the other project processes the healthcare data to identify early signs of an epidemic. Alice's clients exchange their data in text files, which she stores on cloud storage, owned by Eve. For each project, Alice maintains a separate directory, which contains the data along with the index (i.e. keywords). Alice does not trust Eve as the data stored on the cloud storage contains confidential information, which Eve can exploit. To ensure privacy of the data, Alice first encrypts the data and corresponding index, and then outsources them to the cloud storage.

Bob and Mallory are research analysts at Datamine, working with Alice. Bob is an expert in dealing with data related to social networks. Mallory is good in processing

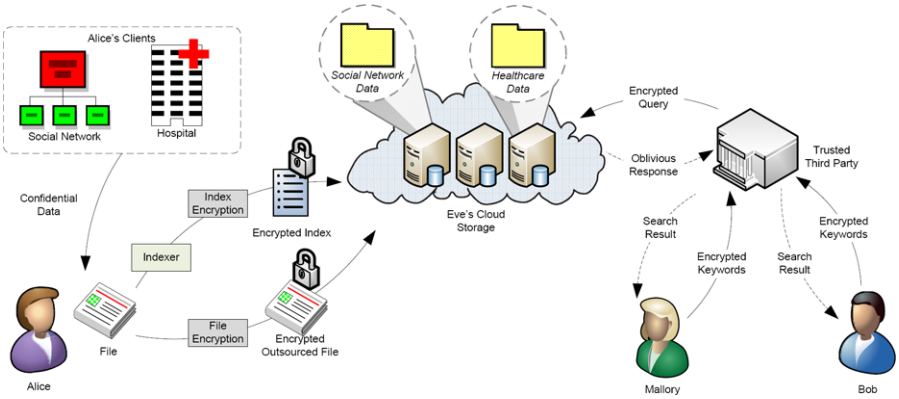

Fig. 1 Abstract model of privacy-aware oblivious data search in a cloud storage

healthcare data. Alice has granted access to both of the research analysts on their respective directories by exchanging data decryption and secret key to conceal search criteria. Whenever Bob and Mallory need to search for a file containing particular keywords, they define selection criteria by encoding keywords with publicly known encoding function. Selection criteria are then encrypted with secret key and submitted to the trusted party, which models oblivious query. Oblivious query is then submitted to the cloud server as a search query. Cloud server then obliviously evaluates the query on encrypted index, and replies back the response. Trusted party processes the cloud server response and sorts the results according to the concealed selection criteria. Query execution result is then sent back to the respective user.

At Datamine, Alice is dealing with confidential data. She does not allow Bob to query directory, which persists the healthcare data. Similarly, Mallory cannot query the social network data. Even if one of them would behave maliciously and team up with Eve, still they would not be able to successfully query the encrypted index. For an attacker, the result of a malicious query is always a randomized response. Oblivious evaluation of the query ensures that Eve cannot learn the keywords, which Bob and Mallory are looking for. Nevertheless, Eve manages to accurately evaluate the query, without compromising privacy of the query and outsourced data. Figure 1 presents an abstract model of our proposed oblivious search for cloud storage.

5.2 Searching cloud storage with oblivious term matching (OTM)

Oblivious query evaluation in a cloud storage system is achieved by uniquely combining homomorphic encryption and proxy re-encryption. The amalgam of these cryptographic primitives ensures that cloud server cannot learn any information about the outsourced data. Most importantly, cloud server evaluates the query submitted by a user, without learning the search criteria and the result of query execution. Besides this, it also ensures that unauthorized users cannot query the outsourced data on which access is not granted by the owner. To achieve privacy-aware data search in an untrusted domain of cloud server, the proposed privacy-aware search for cloud storage is divided into five steps, namely: Setup, Data Outsourcing, Query Generation, Searching and Response Extraction.

5.2.1 Setup

Privacy-aware data search in a cloud storage system is achieved by searching an inverted index ($\mathcal{I}$) associated with the outsourced data ($\mathcal{F}$). For each $\mathcal{F}$, the owner generates $\mathcal{I}$ by utilizing an indexing algorithm. $\mathcal{I}$ contains a list of keywords, along with the frequency of each keyword ($\mathcal{I} = (kw_0, f_0), \ldots, (kw_n, f_n)$). Search queries are evaluated against these keywords. Once $\mathcal{I}$ is generated, the owner initializes proxy re-encryption by generating owner key (ω_o), user key (ω_u), and transformation key ($\omega_{o \to u}$). The owner key ω_o ensures the privacy of keywords within $\mathcal{I}$, whereas keyword frequencies are concealed with third party's secret key (sk). The user key ω_u is used by the user to encrypt search criteria. The owner only shares ω_u with the authorized users. The transformation key $\omega_{o \to u}$ is used by the cloud server to transform ciphertext (encrypted inverted index). Transformation of ciphertext ensures that the owner does not need to outsource separate encrypted index for each authorized user.

5.2.2 Data outsourcing

To ensure that cloud server can obliviously evaluate the search query submitted by an authorized user, owner encodes $\mathcal{I}_{kw_{0\ldots n}}$ by using a publicly known encoding function, i.e. $\mathcal{H}(\mathcal{I}_{kw_{0\ldots n}}) \to \hat{\mathcal{I}}_{kw_{0\ldots n}}$. The encoded keywords ($\hat{\mathcal{I}}_{kw_{0\ldots n}}$) are then encrypted with proxy re-encryption algorithm by using ω_o, i.e. $\mathcal{E}_P(\hat{\mathcal{I}}_{kw_{0\ldots n}}, \omega_o) \to \hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$. To ensure that the cloud server cannot learn any information from the inverted index, an owner encrypts $\mathcal{I}_{f_{0\ldots n}}$ with third party's secret key, i.e. $\mathcal{E}_S(\mathcal{I}_{f_{0\ldots n}}, \text{sk}) \to \mathcal{I}^{sk}_{f_{0\ldots n}}$. After that, the owner encrypts ω_u with the public key of the user to whom it wants to grant searching capabilities over the outsourced data, i.e. $\mathcal{E}_A(\omega_u, k_{\text{pub}}) \to \omega_u^{k_{\text{pub}}}$.

In a cloud storage system, outsourced data can be shared with multiple users – each having its own access privileges over the outsourced data. With proxy re-encryption owner does not need to encrypt $\mathcal{I}_{kw_{0\ldots n}}$ separately to permit each authorized user to query $\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$. An authorized user can submit its query encrypted with its proxy re-encryption secret key (ω_{u_i}). Cloud server then transforms $\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$ to $\hat{\mathcal{I}}^{\omega_{u_i}}_{kw_{0\ldots n}}$ by using an appropriate transformation key ($\omega_{o \to u_i}$) provided by the owner. Thus, the owner only needs to encrypt $\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$ once, and n authorized users can query it, without compromising privacy of the outsourced data.

Once the owner secures $\mathcal{I}_{kw_{0\ldots n}}$, $\mathcal{I}_{f_{0\ldots n}}$, and ω_u, it outsources $\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$, $\mathcal{I}^{\text{sk}}_{f_{0\ldots n}}$ and $\omega_u^{k_{\text{pub}}}$ to the cloud server, along with the outsourced data. After that, availability of the owner is no longer required. Multiple users can engage in an oblivious query evaluation protocol with the cloud server. However, only authorized users can successfully query $\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$. For others, the cloud server obliviously generates a randomized response from which they cannot learn any information about the outsourced data.

5.2.3 Query generation

In order to privately search the cloud storage, a user obtains its proxy re-encryption secret key from the cloud server and deciphers it by using the private key, i.e.

$\mathcal{D}_A(\omega_u^{\text{pub}}, k_{\text{pri}}) = \omega_u$. The user then defines a search criteria ($\mathcal{C}_{kw_{0\ldots l}}$) that consist of a list of keywords $kw_0 \ldots kw_l$. Then $\mathcal{C}_{kw_{0\ldots l}}$ is encoded by using a publicly known encoding function, i.e. $\mathcal{H}(\mathcal{C}_{kw_{0\ldots l}}) \rightarrow \hat{\mathcal{C}}_{kw_{0\ldots l}}$, where $\mathcal{H}$ is the same as used by the owner during data outsourcing. To ensure confidentiality of the keywords, $\hat{\mathcal{C}}_{kw_{0\ldots l}}$ is encrypted with proxy re-encryption by using the proxy re-encryption secret key, i.e. $\mathcal{E}_P(\hat{\mathcal{C}}_{kw_{0\ldots l}}, \omega_u) = \hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots l}}$.

Once privacy of the search criteria is assured, it is send to third party who uses it to model oblivious search query. On receiving $\hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots l}}$ the third party defines a polynomial ($P(x)$), such that each element of $\hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots l}}$ is a root of $P(x)$, i.e. $P(x \in \hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots l}}) = \sum_{i=0}^{l} \alpha_i x^i = 0$; see Sect. 3.2 for more details on defining a polynomial with multiple roots.

Once $P(x)$ is defined in accordance to $\hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots l}}$, the third party then initializes homomorphic encryption by generating a public key (σ_{pk}) and secret key (σ_{sk}). The third party then encrypts the coefficients ($\alpha_{0\ldots l}$) of $P(x)$ with homomorphic encryption algorithm by using σ_{sk}, i.e. $\mathcal{E}_H(\alpha_{0\ldots l}, \sigma_{\text{sk}}) = \alpha^{\sigma_{\text{sk}}}_{0\ldots l}$. After that, $\alpha^{\sigma_{\text{sk}}}_{0\ldots l}$ and σ_{pk} are transferred to the cloud server. Encrypted coefficients ($\alpha^{\sigma_{\text{sk}}}_{0\ldots l}$) are used to execute search query over encrypted inverted index ($\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$). Section 3.2 illustrates that coefficients ($\alpha_{0\ldots n}$) of a polynomial ($P(x)$) can be used to compute set intersection between two private sets. In the context of search over encrypted data, set intersection can be used to execute search query by matching search criteria with the inverted index.

5.2.4 Searching

Cloud server hosts the encrypted inverted index as an encrypted keywords ($\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$) and their concealed frequencies ($\mathcal{I}^{\text{sk}}_{f_{0\ldots n}}$) along with the outsourced data ($\mathcal{F}$). Encrypted query ($\alpha^{\sigma_{\text{sk}}}_{0\ldots l}$) submitted by the third party is evaluated against $\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$. On receiving $\alpha^{\sigma_{\text{sk}}}_{0\ldots l}$, cloud server transforms $\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}$ with the respective user's transformation key ($\omega_{o\rightarrow u}$), provided by the owner, i.e. $\mathcal{T}_P(\hat{\mathcal{I}}^{\omega_o}_{kw_{0\ldots n}}, \omega_{o\rightarrow u}) \rightarrow \hat{\mathcal{I}}^{\omega_u}_{kw_{0\ldots n}}$. Once the encrypted index is transformed, cloud server defines a polynomial ($P(y)$), by using each element of $\alpha^{\sigma_{\text{sk}}}_{0\ldots l}$ as a coefficient of $P(y)$. It then computes oblivious value (Δ_{y_i}), by evaluating $r.P(y_i)$, where $y_i \in \hat{\mathcal{I}}^{\omega_u}_{kw_{0\ldots n}}$ and r is a random number, i.e. $\Delta_{y_i} = r.P(y_i)$.

As the query is concealed by using homomorphic encryption, cloud server cannot learn any information from $P(y_i \in \hat{\mathcal{I}}^{\omega_u}_{kw_{0\ldots n}})$. Once the cloud server has evaluated $P(y_{0\ldots n} \in \hat{\mathcal{I}}^{\omega_u}_{kw_{0\ldots n}}) = \Delta_{y_{0\ldots n}}$, it replies back the query evaluation result—list of oblivious values along with the concealed keyword frequencies to the third party, i.e. $\Delta_{y_{0\ldots n}}$, $\mathcal{I}^{\text{sk}}_{f_{0\ldots n}}$.

5.2.5 Response extraction

On receiving the cloud server's response ($\Delta_{y_{0\ldots n}}$, $\mathcal{I}^{sk}_{f_{0\ldots n}}$), third party decrypts the oblivious values by using the homomorphic secret key, i.e. $\mathcal{D}_H(\Delta_{y_i}, \sigma_{\text{sk}}) = \psi_i$, where ψ_i can be zero or a random number. As the search query was modelled as a polynomial having roots equal to the concealed search criteria, i.e. $P(x \in \hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots l}}) =$

$\sum_{i=0}^{l} \alpha_i x^i$, query evaluation at cloud server can result either in a zero or a non-zero value shown in (10).

$$P(y_i) = \begin{cases} \psi_i = 0 & \text{if } \{y_i | y_i \in \hat{\mathcal{I}}^{\omega_u}_{kw_{0\ldots n}} \wedge y_i \in \hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots l}}\} \\ \psi_i \neq 0 & \text{if } \{y_i | y_i \in \hat{\mathcal{I}}^{\omega_u}_{kw_{0\ldots n}} \wedge y_i \notin \hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots l}}\} \end{cases} \tag{10}$$

Zero value reveals that inverted index contains keyword that matches with the concealed search criteria specified by the user, i.e. $\hat{\mathcal{C}}^{\omega_u}_{kw_i} \in \hat{\mathcal{I}}^{\omega_u}_{kw_{0\ldots n}}$, whereas non-zero reveals that concealed search criteria do not match with any of the keyword in inverted index; consequently, third party recovers $\tilde{r}$. Once encrypted keywords are identified, third party deciphers the corresponding frequency index by using the secret key, i.e. $\mathcal{D}_S(\mathcal{I}^{\text{sk}}_{f_i}, \text{sk}) \rightarrow \mathcal{I}_{f_i}$. After that, the third party sorts the identified encrypted keywords according to the frequency count. The third party then replies back the oblivious query evaluation result to the user.

On receiving the third party's response, a user deciphers the search criteria by using its proxy re-encryption secret key. Through decryption the user learns the keyword that matches with the encrypted index, i.e. $\mathcal{D}_P(\hat{\mathcal{C}}^{\omega_u}_{kw_{0\ldots k}}, \omega_u) = \hat{\mathcal{C}}_{kw_{0\ldots k}}$, where k is the number terms that are identical between $\hat{\mathcal{C}}_{kw_{0\ldots l}}$ and $\hat{\mathcal{I}}^{\omega_u}_{kw_{0\ldots n}}$. During the query, evaluation cloud server learns nothing about the inverted index or the search criteria. However, it accurately evaluates the search query and replies back the oblivious response. Whereas, the third party only learns frequencies of the concealed keyword that matches the search criteria.

6 Implementation

To demonstrate viability of our proposed privacy-aware search for Google's cloud ecosystem, we implement a data search, key management, and intermediate services as standard Java web services. Data search and key management services are deployed on Google App Engine. Intermediate service is deployed on secure local server. Google Docs hosts the outsourced documents. Google Datastore is utilized to store encrypted inverted index associated with the documents stored in Google Docs. Data search service is responsible for executing search queries over the encrypted inverted index. For each user, data owner outsources user's proxy re-encryption key to key management service, whereas each user generates its own RSA key pair. Public key is persisted by the key management service, and the user securely stores private key. Intermediate service is utilized to model oblivious query and process the response of data search service.

To create inverted index we utilize Apache Lucene [30], a high-performance, full-featured text search engine library. With Apache Lucene we associate arbitrary number of keywords with the outsourced documents. Although, there is no restriction on the length of inverted index; however, we restrict Lucene from indexing keywords that are smaller than a four characters. However, data owner is allowed to manually add or remove keywords in inverted index. SHA-512 hashing algorithm is utilized to hash keywords in inverted index. Hashed value of individual keyword is encoded as a BigInteger of arbitrary length. To achieve oblivious query evaluation, we utilize Pascal Paillier cryptosystem.

We have developed owner and client applications as standard Java SE 7.0 desktop applications. The owner application is responsible for generating inverted index, encrypting it with proxy re-encryption and outsourcing it to Google Datastore. The client application is utilized to encode search criteria and encrypting it with proxy re-encryption.

7 Evaluation

We evaluate our proposed privacy-aware search for cloud storage on Google's cloud ecosystem (i.e., Google App Engine [19] and Google Datastore [21]). Data search and Key management services are individually deployed on Google App Engine by using F4 frontend instance class having 2.40 GHz processor and 512 MB main memory [31]. The performance analysis of owner and client applications is carried out on 32-bit Windows 7 machine having 2.60 GHz Dual Core processor with 2 GB main memory. We test execution overhead of intermediate service on 64-bit Windows 7 machine having 3.30 GHz Core i5 processor with 4 GB main memory.

For evaluation, initially we analyse owner and client applications by measuring the execution time required to generate encrypted inverted index, encrypt search criteria, and decrypt response of intermediate service. We then present execution overhead to model and generate oblivious query, and time required to learn result of oblivious query evaluation. Finally, we present the execution time and cost analysis of oblivious query evaluation on Google App Engine. The core purpose of this evaluation is to measure the execution overhead of enabling oblivious query evaluation on cloud storage and modelling of privacy-aware search query. We intentionally neglect the details of key exchange between the key management service and authorized subscribers.

7.1 Inverted index and search criteria generation and processing

We utilize Apache Lucene to generate inverted index. Indexed keywords are hashed by using SHA-512 hashing algorithm. Individual hashed value is then encoded as a BigInteger of arbitrary length. Finally, the encoded values are encrypted with proxy re-encryption by using 1248-bit key and frequencies of indexed keywords are encrypted with AES by using 256-bit key. Figure 2 shows the time required to index file of varied sizes ranging from 1 to 30 MB. It delineates the execution time to encrypt indexed terms within inverted index. Figure 3 presents the time exerted by client application in generating encrypted search criteria comprising of 2 to 14 keywords and decrypting the response of third party.

The evaluation of owner application reveals that the time required to generate inverted index by using Apache Lucene is linear to file size. Besides this, execution time to conceal indexed terms with proxy re-encryption by using 1248-bit key also shows linear behaviour. Although, inverted index can be generated and encrypted in linear time by the owner application; however, size of inverted index (i.e., number of indexed terms) affects the computational time and cost required to evaluate obliviously queries on Google App Engine. Data owner should only select those indexed

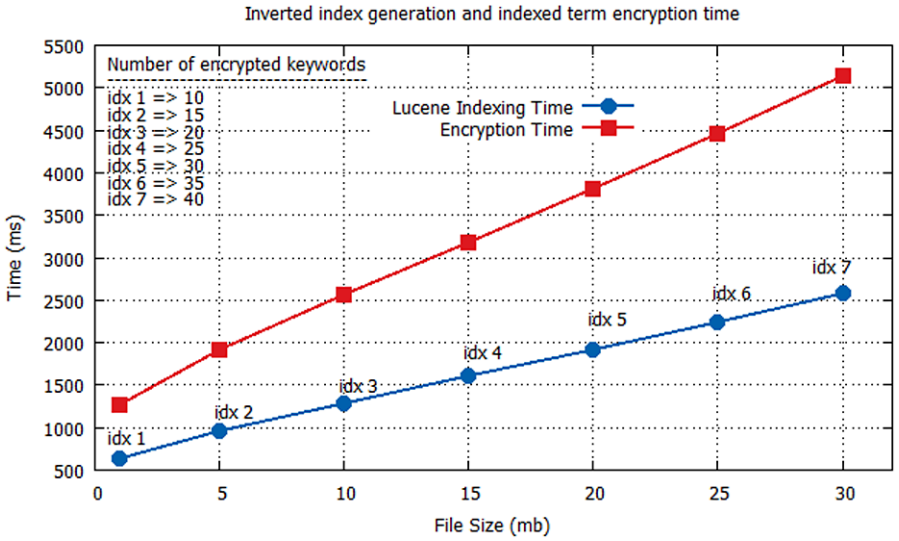

Fig. 2 Inverted index generation and indexed term encryption time

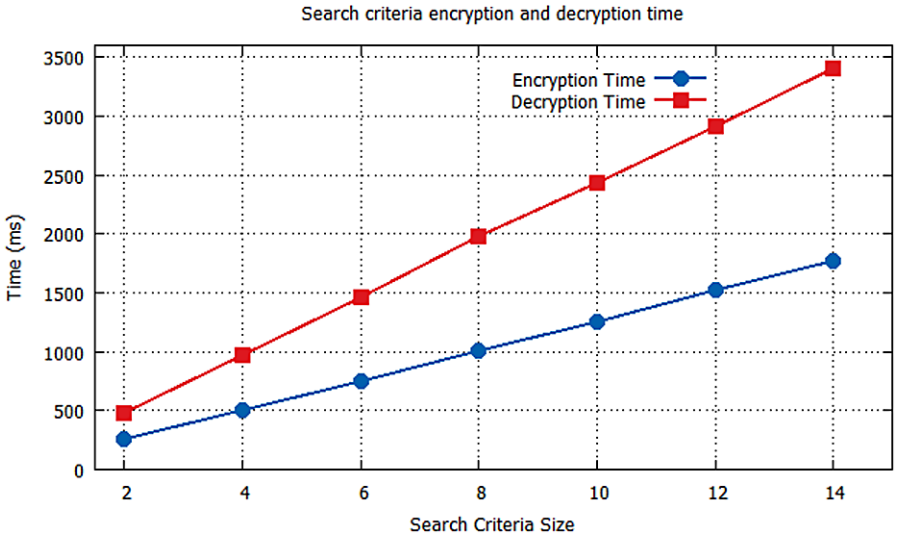

Fig. 3 Search criteria encryption and decryption time

terms that are relevant to a document. For client application, we utilized 2 to 14 different keywords to define search criteria. Our evaluation results highlight the fact that client application can conceal search criteria with proxy re-encryption within fairly response time, considering the level of secrecy achieved with 1248-bit key. Besides this, the decryption of intermediate service's response shows linear behaviour with respect to the number of keywords that comprises the search criteria.

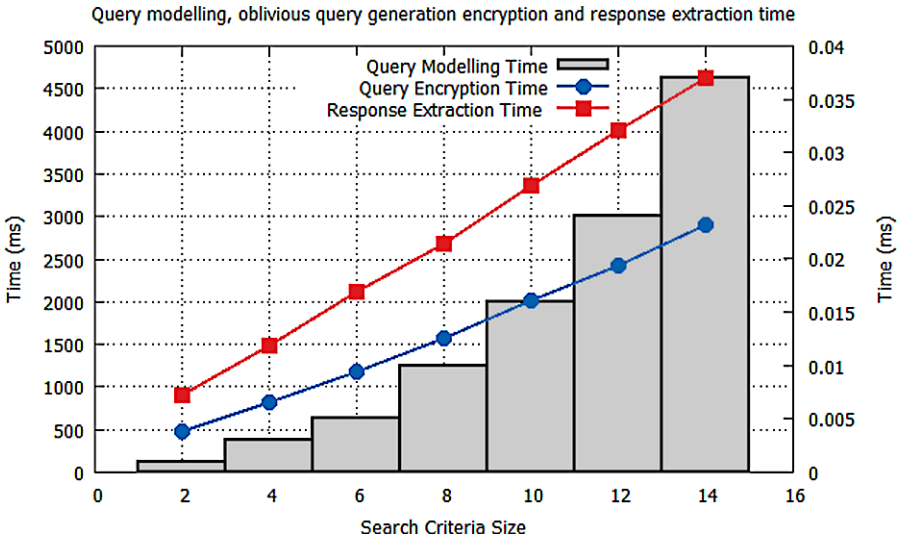

Fig. 4 Query modelling, oblivious query generation encryption and response extraction time

7.2 Oblivious query modelling and response extraction

To model oblivious query a polynomial is defined by the third party, such that the concealed keywords constituting the search criteria are roots of that polynomial. We call this process a query modelling. After that, the third party initializes a 1248-bit key pair of Pascal Paillier cryptosystem and encrypt each coefficient of the defined polynomial. We refer to this process as query encryption. The encrypted query is then transmitted to the cloud storage along with the public key of Pascal Paillier cryptosystem.

Data search service obliviously searches the cloud storage, and responds back the query evaluation result. Third party then learns the keywords that match with the encrypted inverted index by deciphering the query evaluation result. Matched keywords are sorted by the third party and sent back to the user. We refer to the process of learning matched keywords as response extraction. Figure 4 presents the execution time of oblivious query modelling, query encryption, and response extraction.

Search query comprised of higher number of keywords can be effortlessly modelled by the user. However, execution time of query encryption linearly increases with the increase in size of search query. It applies for the response extraction, as well. Although number of terms affect the encryption and response extraction time, still it remains fairly amicable and never increases form 2901 and 4627 milliseconds respectively for query having 14 distinct keywords comprising the search criteria.

7.3 Oblivious query evaluation on Google App Engine

We measure the execution time and cost of oblivious query evaluation on Google App Engine. For the performance analysis we consider the execution time of a billable

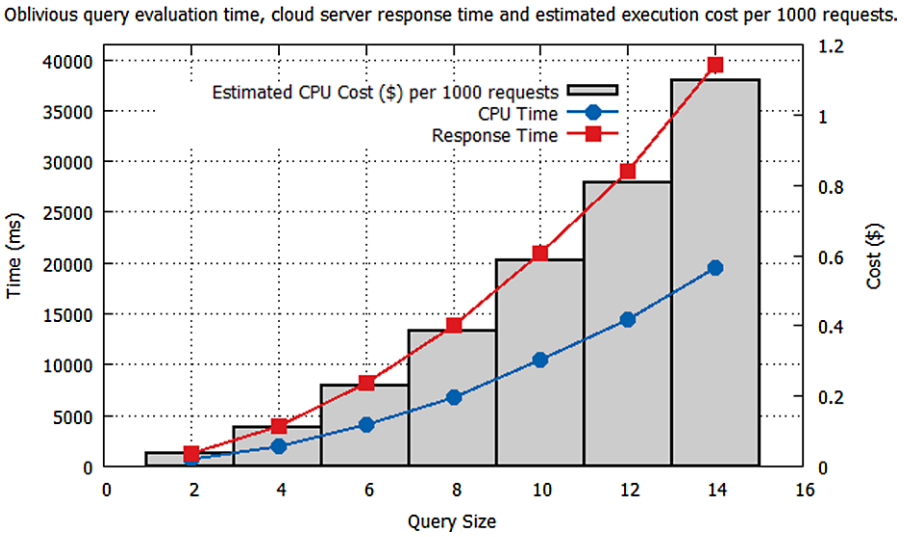

Fig. 5 Oblivious query evaluation time, cloud server response time and estimated execution cost for 1000 requests

CPU (CPU Time), time taken to complete the request (response time) and estimated CPU usage cost for 1000 identical requests (cpm_usd). To obliviously evaluate the search queries, data search service takes encrypted query from the third party. It then executes the oblivious search query for each value in the encrypted inverted index. The result of oblivious query evaluation is then sent back of the third party. Figure 5 shows computational and cost analysis of oblivious query evaluation on Google App Engine for a single encrypted inverted index entry.

It is clear that execution time and cost greatly depend on number of keywords comprising the selection criteria. However, the performance analysis has shown a linear relation among the size of query, CPU Time and its estimated cost. Through the predictive cost analysis, we have shown that the estimated execution cost of oblivious query evaluation having 2 to 14 keywords remains between 0.035 and 1.09 dollars for 1000 requests of similar computational requirements. We have highlighted the fact that oblivious data search can be realized by a cloud storage system with amicable computational load and fairly reasonable cost, without compromising privacy of the outsourced data and search queries as well.

8 Discussion

Data search is an integral part of cloud storage service. Searching methodologies ensure that subscriber of a cloud storage can access relevant data without generating needless data access requests. However, when confidential data are outsourced to these services in encrypted format, subscribers can no longer use standard search queries to look for a particular file or data content, mainly because comparison operators cannot be evaluated for the encrypted data and search criteria specified in search

query. Besides this, standard search queries do not ensure privacy of the outsourced data as malicious or curious cloud service provider can use them to learn confidential information about the outsourced data.

Numerous efforts have been made in the form of cryptographic primitives and enterprise search products to achieve searching capabilities over the encrypted data. These systems mainly utilize cryptographic trapdoors or index data structures to execute search queries. However, these approaches lose their efficacy in the area of cloud storage due to their intrinsic properties of trapdoor distribution, and in-house index management. Moreover, these systems either do not enforce access control policies or rely on trusted third party to achieve privacy-aware data search in an untrusted domain of CSP.

In order to leverage subscribers of cloud storage with searching capabilities, we have proposed Oblivious Term Matching (OTM). It is privacy-aware data search that can be regarded as a value-added service for existing cloud storages. OTM ensures that search queries are obliviously evaluated by a CSP, without learning any information about the outsourced data. Since OTM is an indexed-based data search, unlike trapdoor-based approaches, authorized subscribers are not confined to a limited number of trapdoors defined by a data owner. To restrain CSP from compromising privacy of the outsource data, indexes are encrypted before they can be outsourced to a CSP. OTM is independent of data encryption that ensures confidentiality of the outsource data, thus it can be integrated with any cloud storage system to realize a privacy-aware data search.

OTM utilizes homomorphic and proxy re-encryption to ensure that a cloud server obliviously evaluates search queries and only authorized subscribers can search cloud storage. Since we utilize private matching, cloud server cannot learn any information about the search criteria (i.e. keywords) as coefficients of a search query are homomorphically encrypted by using Pascal Paillier cryptosystem. Private matching ensures that cloud server cannot even learn the selection criteria that match with the values in encrypted inverted index. Thus, for query evaluation, OTM provides two levels of secrecy. First, cloud server cannot learn the search criteria. Second, it cannot learn the keywords that are common between the search criteria and encrypted inverted index.

To realize a privacy-aware data search service, we utilize proxy re-encryption. It encrypts the inverted index generated by the data owner and search criteria defined by an authorized subscriber. Proxy re-encryption ensures that cloud server only has to persist single copy of encrypted inverted index and yet it is able to evaluate oblivious queries generated by authorized subscribers. For each authorized subscriber, cloud server has a valid transformation key. Whenever a subscriber initiates a data search request, cloud server transforms the encrypted inverted index by using an appropriate transformation key. Since only the authorized users have their respective transformation key with the cloud server, search queries of an unauthorized user cannot be evaluated successfully. Even if cloud server behaves maliciously and teams up with an unauthorized subscriber, privacy of the outsourced data cannot be compromised because cloud server does not have valid transformation key and unauthorized user does not have its proxy re-encryption secret key.

OTM utilizes trusted third party to model oblivious queries, process the response of cloud server, and sort the result of oblivious query evaluation according to the

frequencies of matched keywords. Utilizing trusted third party to sort result reveals frequency of individual encrypted keywords. Since these keywords are concealed with proxy re-encryption, no oblivious information is leaked to the trusted third party. However, if sorting of results is not required, trusted third party can be seamlessly avoided without losing efficacy of OTM.

The practicality of proposed privacy-aware cloud search is demonstrated by realizing it for Google's cloud ecosystem. We deploy search service on Google App Engine, and encrypted indexes are stored on Google's Datastore. To generate inverted index, we opt for Apache Lucene. However, OTM is not confined to any specific indexing framework. Data owner can even manually associate keywords and their respective frequencies with the outsourced data. OTM is mainly based on two asymmetric cryptosystems, i.e., Pascal Paillier and proxy re-encryption. For our implementation we opt for 1248 bit long keys (by default). As recommended by ECRYPT II 2011, 1248-bit long key based on Discrete Logarithm Group provides long-term protection against small organizations, and very short-term protection against agencies [32, 33]. However, key lengths of both Pascal Paillier and proxy re-encryption are configurable by the data owner according to the level resilience required against a determined attacker.

With OTM, we have realized a data search service which ensures that only authorized subscribers can search the outsourced data obliviously. Search queries are obliviously executed by a cloud server that does not learn any information about the outsourced data, not even about the result of query execution. OTM ensures that cloud server cannot relate search queries of two different users even if the queries are modelled with similar search criteria. Through our implementation we have highlighted the fact the OTM is independent of underlying cloud storage system. It can seamlessly be integrated with other cloud storage services to leverage subscribers with privacy-aware searching capabilities.

9 Conclusion and future directions

In this paper, we addressed the problem of privacy-aware data search over encrypted data residing within the untrusted domain of a cloud storage provider (CSP). To leverage subscribers of cloud storage, we proposed Oblivious Term Matching (OTM) that ensures authorized search over encrypted data without revealing any information about the search queries. Unlike existing methodologies of searching encrypted data, OTM is not confined to a limited number of trapdoors and allows authorized subscribers to define their own search queries. OTM utilizes amalgam of homomorphic encryption and proxy re-encryption to ensure that CSP can obliviously evaluate search queries without learning any information about encrypted outsourced data. OTM can be regarded a value-added service that provides searching capabilities to existing cloud-based data sharing and collaborative systems. It enables data owner to define encrypted index once, allowing multiple subscribers to search it according to their access privileges. Evaluation of OTM has highlighted the fact that it exerts reasonable execution load on each of the involved entities. Besides this, its estimated execution cost is fairly reasonable, being only 0.03 to 1.09 $ for 1000 search requests having search criteria comprised of 2 to 14 distinct keywords.

The current implementation of proposed privacy-aware data search service does not support range queries. For our future research directions, we have intention of incorporating range queries. However, the inclusion of range queries does not affect the fundamental concept of OTM that is based on private matching and proxy re-encryption. Besides this, we will incorporate order preserving encryption [34] to restrain trusted third party from learning frequency information during response extraction.

Acknowledgement This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Centre) support program supervised by the NIPA (National IT Industry Promotion Agency)" (NIPA-2012-(H0301-12-2001)).

Appendix: Performance evaluation: data tables

Performance evaluation presented in Sect. 7 is based on the following data tables. Figure 2 presented the visual representation of Table 2. Similarly, Figs. 3, 4, and 5 are associated with Tables 3, 4, and 5, respectively.

Table 2 Inverted index generation and indexed term encryption time

File size (mb)	Lucene indexing time (ms)	No. of indexed terms	Encryption time (ms)
1	631	10	1260
5	960	15	1915
10	1291	20	2572
15	1601	25	3186
20	1912	30	3803
25	2246	35	4467
30	2582	40	5134

Table 3 Search criteria encryption and decryption time

Search criteria size	Encryption time (ms)	Decryption time (ms)
2	252	484
4	500	970
6	751	1468
8	1003	1973
10	1253	2430
12	1524	2915
14	1772	3401

Table 4 Query modelling, oblivious query generation encryption and response extraction time

Search criteria size	Query modelling time (ms)	Query encryption time (ms)	Response extraction time (ms)
2	0.001	483	902
4	0.003	821	1493
6	0.005	1181	2116
8	0.010	1571	2680
10	0.016	2008	3361
12	0.024	2421	4011
14	0.037	2901	4627

Table 5 Oblivious query evaluation time, cloud server response time and estimated execution cost for 1000 requests

Query size	CPU time (ms)	Response time (ms)	Estimated CPU cost ($) for 1000 requests
2	639	1255	0.035
4	1966	3934	0.110
6	4106	8245	0.229
8	6850	13827	0.384
10	10435	21027	0.584
12	14415	29069	0.808
14	19554	39504	1.098

References

1. Armbrust M et al (2010) A view of cloud computing. Commun ACM 53:50–58
2. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25:599–616
3. Gartner—top trends for 2010, by Brian Prentice
4. Grossman RL (2009) The case for cloud computing. In: IT professional, March, vol 11, pp 23–27
5. Amazon s3 pricing
6. Google App Engine—pricing and features
7. Hacigümüş H, Iyer B, Li C, Mehrotra S (2002) Executing sql over encrypted data in the database-service-provider model. In: Proceedings of the 2002 ACM SIGMOD international conference on management of data, SIGMOD '02, New York, NY, USA. ACM, New York, pp 216–227
8. Kaufman LM (2009) In: Data security in the world of cloud computing, Piscataway, NJ, USA, July, vol 7, pp 61–64. IEEE Educational Activities Department
9. Curino C, Jones E, Popa RA, Malviya N, Wu E, Madden S, Balakrishnan H, Zeldovich N (2011) Relational cloud: a database service for the cloud. In: 5th biennial conference on innovative data systems research, Asilomar, CA, January
10. Chow R, Golle P, Jakobsson M, Shi E, Staddon J, Masuoka R, Molina J (2009) Controlling data in the cloud: outsourcing computation without outsourcing control. In: Proceedings of the 2009 ACM workshop on cloud computing security, CCSW '09, New York, NY, USA. ACM, New York, pp 85–90
11. Song DX, Wagner D, Perrig A (2000) Practical techniques for searches on encrypted data. In: Proceedings of 2000 IEEE symposium on security and privacy, S P 2000, 2000, pp 44–55
12. Boneh D, Crescenzo GD, Ostrovsky R, Persiano G (2004) Public key encryption with keyword search. In: EUROCRYPT, pp 506–522

13. Li M, Yu S, Cao N, Lou W (2011) Authorized private keyword search over encrypted data in cloud computing. In: 2011 31st international conference on distributed computing systems (ICDCS), June, pp 383–392
14. Wang C, Cao N, Li J, Ren K, Lou W (2010) Secure ranked keyword search over encrypted cloud data. In: 2010 IEEE 30th international conference on distributed computing systems (ICDCS), June, pp 253–262
15. Google Search Appliance
16. Enterprise search server solutions
17. Paillier P (1999) Public key cryptosystems based on composite degree residuosity classes. In: Proceedings of the 17th international conference on theory and application of cryptographic techniques, EUROCRYPT'99, Berlin, Heidelberg. Springer, Berlin, pp 223–238
18. Ateniese G, Fu K, Green M, Hohenberger S (2006) Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans Inf Syst Secur 9:1–30
19. Google App Engine—run your web applications on Google's infrastructure
20. Google Docs—create and share your work online
21. Google App Engine—using the datastore
22. cheng Chang Y, Mitzenmacher M (2005) Privacy preserving keyword searches on remote encrypted data. In: Proc of 3rd applied cryptography and network security conference (ACNS), pp 442–455
23. Curtmola R, Garay J, Kamara S, Ostrovsky R (2006) Searchable symmetric encryption: improved definitions and efficient constructions
24. Yang Z, Zhong S, Wright RN (2006) Privacy-preserving queries on encrypted data. In: Proc of 11th European symposium on research in computer security (Esorics), pp 479–495
25. Kamara S, Papamanthou C, Roeder T (2011) Cs2: a searchable cryptographic cloud storage system. Tech report MSR-TR-2011-58, Microsoft research
26. Singh A, Srivatsa M, Liu L (2009) Search-as-a-service: outsourced search over outsourced storage. ACM Trans Web 3:13:1–13:33
27. Paillier P (2000) Trapdooring discrete logarithms on elliptic curves over rings. In: Proceedings of the 6th international conference on the theory and application of cryptology and information security: advances in cryptology, ASIACRYPT '00, London, UK. Springer, Berlin, pp 573–584
28. Freedman M, Nissim K, Pinkas B (2004) Efficient private matching and set intersection. Springer, Berlin, pp 1–19
29. Yu S, Wang C, Ren K, Lou W (2010) Achieving secure, scalable, and Fine-grained data access control in cloud computing. In: Proceedings of the 29th conference on information communications, INFOCOM'10, Piscataway, NJ, USA. IEEE Press, New York, pp 534–542
30. Apache Lucene core
31. Google App Engine—adjusting application performance
32. Ecrypt ii yearly report on algorithms and keysizes
33. New European schemes for signatures, integrity, and encryption
34. Agrawal R, Kiernan J, Srikant R, Xu Y (2004) Order preserving encryption for numeric data. In: Proceedings of the 2004 ACM SIGMOD international conference on management of data, SIGMOD '04, New York, NY, USA. ACM, New York, pp 563–574