Efficient Multi-User Indexing for Secure Keyword Search

Eirini C. Micheli, Giorgos Margaritis, Stergios V. Anastasiadis
Department of Computer Science and Engineering
University of Ioannina, Greece
{emicheli,gmargari,stergios}@cs.uoi.gr

ABSTRACT

Secure keyword search in shared infrastructures prevents stored documents from leaking confidential information to unauthorized users. We assume that a shared index provides confidentiality if it can only be used by users authorized to search all the documents contained in the index. We introduce the Lethe indexing workflow to improve query and update efficiency in secure keyword search. Lethe clusters together documents with similar sets of authorized users, and only creates shared indices for configurable volumes of documents with common users. Based on the published statistics of an existing dataset, we show that Lethe generates an indexing organization that simultaneously achieves both low search and update cost.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; K.6 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Design, Experimentation, Measurement, Security

Keywords

inverted index, clustering, full-text search, shared data storage, confidentiality

1. INTRODUCTION

Keyword (or full-text) search is an indispensable service for the automated retrieval of text documents, whether proprietary within an organization, or public across the web. Over the years, an enormous amount of accumulated text has gradually expanded keyword search to several contemporary storage environments, such as personal content archives, online social networks, and cloud facilities. At the same time, the efficiency benefits of storage consolidation increasingly motivate the maintenance of sensitive data over public infrastructures. Indeed, the access control enforced at the storage level is often presumed sufficient for the necessary confidentiality isolation of co-located users and organizations.

An inverted index is the dominant indexing structure in keyword search. The stored documents are preprocessed into a posting list per keyword (or term), which provides the occurrences (or postings) of the term across all the documents. A single index shared among multiple users offers search and storage efficiency. However, it can also leak confidential information about documents with access permissions limited to a subset of the users [5, 13, 10, 3]. The problem persists even if a query is initially evaluated over the shared index, and later the inaccessible documents are filtered out from the final result list before it is returned to the user [5].

A known secure solution applies a shared index by limiting search to term postings of documents searchable by the user [5]. During query processing it skips dependencies on inaccessible documents through posting filtering at extra list processing overhead. In online social networks, recent research applies advanced list-processing operators and cost models to improve secure search efficiency [3]. First, it organizes the friends of each user into appropriate groups based on characteristics of the search workload. Then, during query handling, it intersects the list of documents that contain a term against the list of documents authored by the querying user and the union of her friend groups.

A different secure solution partitions the document collection by search permissions, and maintains a separate index for each partition [13]. The collection ends up indexed by a limited number of indices, and query handling runs over all the indices that contain documents searchable by the querying user. However, minor variations in search permissions of different documents increases the number of indices. Although smaller indices can be completely eliminated by replicating their contents to private per-user indices, this approach increases document duplication across the indices and the respective update cost.

In this study, we aim to achieve low search latency and index update cost by limiting both the number of indices per user and the document duplication across the indices. We group by search permissions the documents into families, and cluster together the families with similar permissions. We maintain one index for the documents searchable by a maximal common subset of users in a cluster. Cluster docu-

7th International Workshop on Privacy and Anonymity in the Information Society (PAIS'14) March 28, 2014, Athens, Greece

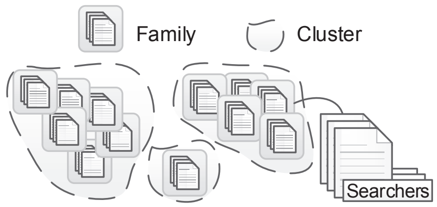


Figure 1: Document families grouped by searcher similarity L_s into clusters.

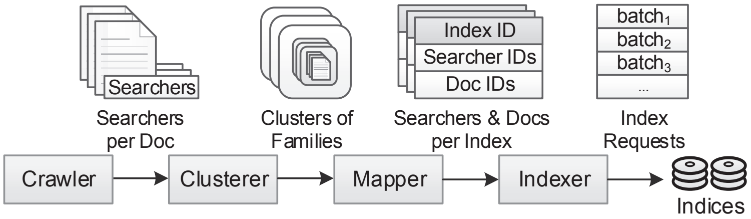


Figure 2: The four stages of the Lethe workflow.

ments whose users lie outside the above subset are inserted into either per-user private indices or additional multi-user indices.

Our indexing organization for secure keyword search is innovative because we (i) skip query-time list filtering via prebuilt securely-accessible indices, and (ii) effectively reduce the number of searched or maintained indices through configurable partial merging of indices for documents with common authorized users. In Sections 2 and 3 we present the Lethe indexing workflow and our prototype implementation. In Sections 4 and 5 we show some experimental results and examine previous related work, while in Section 6 we summarize our conclusions and plans for future work.

2. INDEXING ORGANIZATION

We next provide the basic assumptions and goals of our work, and describe the stages of the Lethe indexing workflow that we propose.

2.1 Assumptions and Goals

We target collections of text documents in shared storage environments accessible by multiple users. The system applies access control to protect the confidentiality and integrity of the stored documents from actions of unauthorized users. We designate as *owner* the user who creates a document, and *searchers* of the document the users who are authorized to search by keywords for the document and read its contents. The system preprocesses the documents content into the necessary indexing structure to enable interactive search through keyword criteria set by the searchers. In our indexing organization we set the following goals:

- **Security** Ensure that the indexing structure provides confidentiality of the searched documents with respect to the document contents and their statistical characteristics (e.g., number of documents, term properties).
- **Search Efficiency** Minimize the search latency per query as measured through an appropriate metric (e.g., median or high percentile).
- **Indexing Cost** Minimize the document insertion I/O activity and indexing storage space required for the entire collection.

We require that users are authenticated by the system and authorized to only search documents with the necessary access permissions. Accordingly, we build a separate index for each document subset with common access permissions. We presently examine secure search in multi-user environments, but leave outside the study scope the closely related but complementary problem of search over encrypted storage. In fact, search with encrypted keywords over encrypted documents conceals the search activity and stored documents from a storage provider, but it does not necessarily hide the characteristics of stored content from unauthorized searchers [13].

2.2 The Lethe workflow

We introduce the Lethe workflow consisting of four basic stages for crawling, clustering and mapping the documents to the generated indices.

Crawler In order to realize our goals, we build an appropriate indexing organization based on the document search permissions. Let a text dataset $\mathcal{T} = (D_\mathcal{T}, S_\mathcal{T})$, where $D_\mathcal{T}$ is the set of all documents, and $S_\mathcal{T}$ the set of all users with search permissions over one or more documents of $D_\mathcal{T}$. First we crawl the names (e.g., paths) and permissions (e.g., allowed searchers) of documents in $\mathcal{T}$, and assign unique identifiers to the members of $D_\mathcal{T}$ and $S_\mathcal{T}$. Then we group into a separate *family* $f = (D_f, S_f)$, each set of documents $D_f \subseteq D_\mathcal{T}$ with identical set of searchers $S_f \subseteq S_\mathcal{T}$.

Clusterer We aim to maintain a single index for the searchers who are common among similar families. Accordingly, we need to identify those families with substantial overlap in their searcher sets. We address this issue as a universal clustering problem over the searcher sets of the families in the entire dataset (Fig. 1). We parameterize the clustering method as necessary to assign every family to exactly one cluster, without omitting any families as noise.

Let the *searcher similarity* $L_s \in [0, 1]$ be a configurable parameter to adjust the number of common searchers across the families of each created cluster. We generate a set $C_\mathcal{T}$ of clusters, where each cluster $c \in C_\mathcal{T}$ contains a set F_c of families, and each family $f \in F_c$ contains the document set $D_f \subseteq D_\mathcal{T}$. The document set D_c of cluster c is derived from the union of the documents contained across all the families of c, i.e., $D_c = \bigcup_{f \in F_c} D_f$. Thus, the number of documents in cluster c is at least as high as the number of families in c, i.e., $|D_c| \geq |F_c|$.

Mapper We strive to map each family f to the minimum number of indices required to securely handle keyword queries over the documents in D_f, but also minimize the total number of indices in the system. First, we dedicate to every searcher $u \in S_\mathcal{T}$ the pair $P_u = (D^e_u, \{u\})$, where $D^e_u \subseteq D_\mathcal{T}$ is the set of documents exclusively searchable by u. Then, we assign to P_u a *private* index I_u containing the documents of D^e_u.

Let the *cluster intersection* P_c of cluster $c \in C_\mathcal{T}$ be a pair (D^i_c, S^i_c), with $D^i_c = D_c$, and $S^i_c = \bigcap_{f \in F_c} S_f$ the intersection of searchers in the families of F_c. By family definition, the documents in D^i_c are searchable by all the searchers in S^i_c. If $\left|S^i_c\right| \neq \emptyset$, we dedicate a separate index I_c to the intersection P_c. For every family $f \in F_c$, we also define a *family* *difference* P_f as the pair (D^d_f, S^d_f), where $D^d_f = D_f$ and $S^d_f = S_f - S^i_c$, i.e., S^d_f corresponds to the searchers of family f not contained in S^i_c of P_c. If $S^d_f \neq \emptyset$, we have to allow the users $u \in S^d_f$ to securely search for documents $d \in D^d_f$.

An extreme approach to address the above P_f search problem is to insert every document $d \in D^d_f$ to every private index $I_u, u \in S^d_f$. However, a difference P_f may contain a relatively large number $\left|D^d_f\right|$ of documents searchable by a considerable number $\left|S^d_f\right|$ of users. Hence, the above approach would end up to a large number of documents duplicated across the private indices of many users. At the other extreme, we could dedicate a separate index I_f to every difference P_f with $\left|S^d_f\right| \neq \emptyset$. However, this approach runs the risk of generating in the system a large number of indices, each serving a small number of documents and searchers.

We introduce the *duplication product* $R^d_f = \left|D^d_f\right| \cdot \left|S^d_f\right|$ to approximate[1] the potential document duplication resulting from indexing a family difference P_f. Subsequently, the decision of whether we should create a dedicated index I_f depends on how R^d_f compares to the configurable *duplication threshold* T_d. We assume that $R^d_f < T_d$ implies an affordable cost of inserting the documents $d, \forall d \in D^d_f$, to private indices $I_u, \forall u \in S^d_f$. Instead, $R^d_f \geq T_d$ suggests that devoting a separate index I_f to the difference P_f is preferable.

An optimization that we do not examine further due to its complexity is to pursue additional duplication reduction by intersecting the searchers of the differences $P_f, \forall f \in F'_c$, for appropriate $F'_c \subset F_c$ corresponding to cluster c.

Indexer We insert each document $d \in D_{\mathcal{T}}$ to the appropriate I_c, I_f, and I_u indices specified by the above mapping phase. In order to keep low the necessary I/O activity, we separately generate each index through a specification of the contained documents. We experimentally validated that the alternative approach of specifying to the system the indices of each document leads to higher I/O activity due to lower storage locality during the index updates. As new documents are added to the collection, we look for existing indices to securely serve all the searchers of each document. Periodically, we repeat the previous clustering and mapping phases to optimize the search over the accumulated document collection. Deletions or modifications of inserted documents are handled with the necessary changes of the index contents and potential reorganization of their mapping to documents. We summarize the four stages of the Lethe workflow along with their outputs in Fig. 2.

3. PROTOTYPE IMPLEMENTATION

Based on the above design, our prototype implementation consists of four components: (i) *crawler*, (ii) *clusterer*, (iii) *mapper*, and (iv) *indexer*. The crawler specifies a unique identifier for each document and gathers information about the permitted document searchers. The clusterer organizes the documents into families according to their searchers, and then clusters the families based on the searcher similarity L_s. We use the searchers of each document as key to create the families over a hash table. Thus, all documents with identical searchers end up at the same entry of the table.

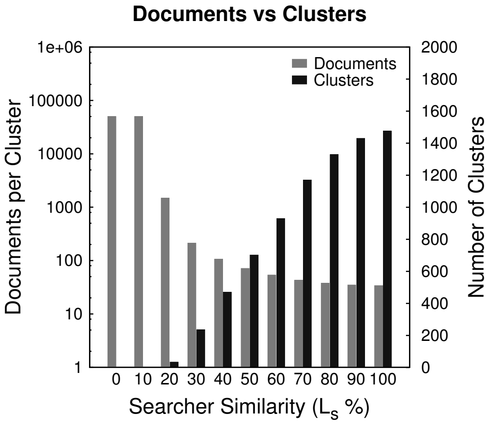


Figure 3: For the synthetic dataset based on DocuShare[14], we examine the number of created clusters and the number of documents per cluster across different L_s values.

Subsequently, we group the families with similar searchers into the same cluster represented as a vector of family identifiers. The searchers of a family f are concisely represented through a *searcher bitmap* M_f of length equal to the number of users $|S_{\mathcal{T}}|$ in the stored dataset. In bitmap M_f we set equal to 1 the values at bit positions specified by identifiers of permitted family searchers $u \in S_{\mathcal{T}}$.

Since we do not know in advance the number of clusters, we use a clustering algorithm that produces this number as output (e.g., DBSCAN) rather than requiring it as input (e.g., K-means) [16]. Within each cluster, the mapper identifies the cluster intersections and family differences. Each intersection or difference is specified through the contained documents and authorized searchers. We assign a dedicated index to each cluster intersection, and we use a dedicated index or the private indices of the respective searchers for each family difference according to the duplication threshold T_d. The indexer receives the index specifications from the mapper, and splits each index into document batches. Then it communicates with the search engine to insert the documents of each batch to the respective index, after initializing it if necessary. Finally, the search engine serves queries by using the indices permitted to each authorized searcher.

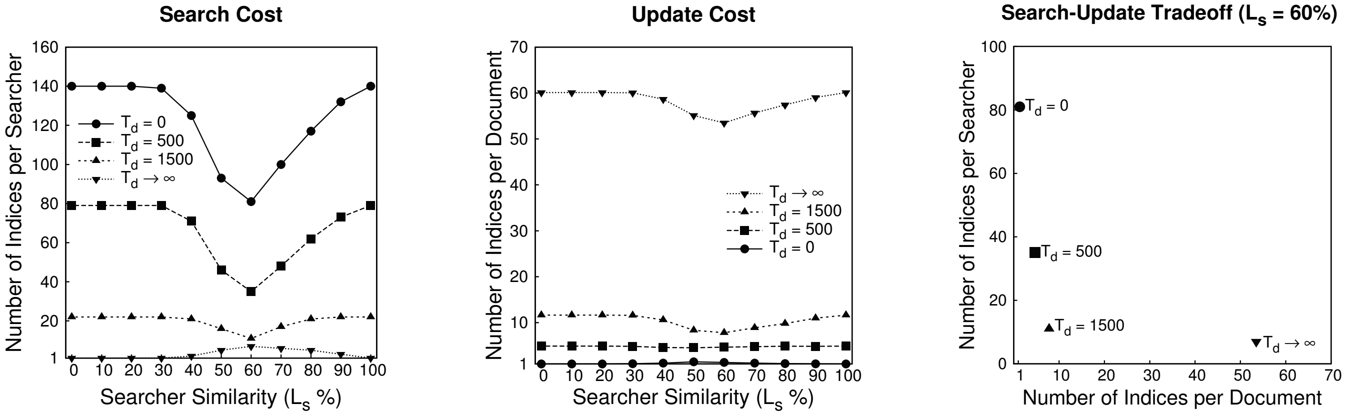
4. EXPERIMENTAL EVALUATION

We use the published statistics of a real dataset to generate a synthetic workload, and apply a prototype implementation of the Lethe workflow that we developed. Then we measure the number of indices per user and document for different parameters, and analyze the security and efficiency characteristics of our approach.

4.1 Document Dataset

We generate a synthetic document collection with searcher lists based on published measurements of an existing dataset (DocuShare [14]). We set the number of users to 200, user groups to 131, documents to 50000, and max group size to 50. We specify the sizes of individual groups from the DocuShare statistics, and uniformly pick users as group members. Based on the DocuShare statistics, we specify the number of users and groups allowed to search each docu-

[1] For increased accuracy of R^d_f over diverse document sizes, we could replace $\left|D^d_f\right|$ with the total number of postings contained in all documents $d \in D^d_f$.



(a) (b) (c)

Figure 4: For the synthetic dataset based on DocuShare[14], we illustrate the number of indices (a) per searcher and (b) per document across different L_s and T_d values. In addition, we show (c) the search-update tradeoff for different T_d values at fixed L_s=60%.

ment, and then uniformly assign to each document specific users and groups. We implemented the crawler, clusterer and mapper in C/C++ with STL, and the indexer in Perl (v5.10.1). For clustering we applied the DBSCAN algorithm with MinObjs=1 and Eps=L_s [16]. We run the computations over Linux v2.6.32 on quad-core x86 2.33GHz processor, 4GB RAM, and 7.2KRPM SATA disks.

4.2 Measurement Results

We applied the Lethe workflow to organize the examined dataset into clusters of document families. For different L_s values, in Fig. 3 we show the average number of documents per cluster and the total number of clusters. The duplication threshold T_d is not included because it only applies to the subsequent mapping stage. The ideal similarity should result into family clusters with common searchers per cluster to be efficiently served by a single index. For instance, setting L_s=60% generates 929 clusters with 53.82 documents per cluster. At the extreme case of L_s=0%, there is 1 cluster containing all 1475 families and 50000 documents. At the other extreme of L_s=100%, there are 1475 clusters, each containing 1 family with 33.90 documents on average.

We regard the number of indices per searcher as a proxy of the search cost, because it specifies the number of document lists that have to be merged into the final search result. Accordingly, in Fig. 4a we examine the sensitivity of the search cost to the values of the L_s and T_d parameters. We experimented with T_d values in the range [0,+∞). The indices per searcher vary between 35 and 79 at T_d=500, and between 11 and 22 at T_d=1500. Setting L_s=0% or 100% usually maximizes the number of indices per searcher. This follows from the fact that index sharing is limited in a single cluster of diverse families, or numerous clusters of one family each. On the contrary, setting L_s=60% leads to non-empty cluster intersections, and roughly minimizes the number of indices per searcher. One exception to the above pattern occurs with $T_d \to \infty$, which prohibits index sharing within family differences, and minimizes the indices per searcher at L_s=0% or 100% instead of L_s=60%.

The update cost of the indexing organization can be proxied through the average number of indices that contain each document, and have to be updated during document insertion. In Fig. 4b we examine the sensitivity of the update cost to L_s and T_d. At L_s=60%, we notice that setting T_d=1500 or $T_d \to \infty$ minimizes the number of indices per document to 7.80 and 53.48, respectively. Instead, the curves remain almost flat across different L_s values when T_d=0 or 500. If we combine this observation with the outcome of the previous paragraph, we conclude that L_s=60% leads to both low update and search cost.

A striking difference between Figures 4a and 4b is the opposite effect of T_d to the search and update cost. This tradeoff is further illustrated in Fig. 4c for different T_d values and fixed L_s=60%. We found T_d=1500 to provide a reasonable choice, because it simultaneously achieves a low number of 11 indices per searcher and 7.8 per document. Overall, at L_s=60% and T_d=1500, the mapper specifies a total of 298 indices: 84 and 182 shared indices for intersections and differences, respectively, and 32 private indices. In early measurements (not shown) that we did over a search engine, the above results directly translated to low search and update latency unlike alternative settings.

4.3 Analysis of Results

Our preliminary experiments provide strong evidence for an improved method to achieve efficient and secure keyword indexing. The method is secure because a query can only use indices of documents that the searcher is permitted to access [5]. The method is also efficient for several reasons.

First, we guarantee that the result returned by an index does not require any filtering to remove documents inaccessible to the searcher. We only require to merge the results from multiple indices for ranking purposes, as is typically already done by parallel or distributed search engines. Thus, we avoid the extra query-time overhead for list processing required by previous secure methods [3].

Second, the clustering of document families allows the service of common searchers in the cluster intersection with a single index. Thus, we reduce the average number of indices per searcher, which translates into smaller number of result lists to be generated and merged during query handling. To the best of our knowledge, this is the first time that clustering is applied for the efficiency of secure keyword search.

Third, the control of indexing duplication through the threshold T_d prevents the insertion of the same document to an excessive number of multiple private indices, which was previously required [13]. Instead, we create extra shared indices whenever the number of documents and their common searchers justify their cost.

5. RELATED WORK

We compare our work with related research results previously developed for secure text indexing, remote storage of encrypted documents, and online social networks.

Security-aware Indexing Büttcher and Clarke examine the problem of filesystem search with relevance ranking based on the vector space model [5]. A secure search engine must only deliver query results dependent on files searchable by the querying user. Thus, a system-wide index to find and rank all matching files is insecure, because it can leak the total number of files matching a term, or term statistics normally unavailable to a user. As a solution, the authors propose to restrict query processing to the parts of posting lists that the querying user is permitted to access. The resulting performance slowdown can be reduced through appropriate reordering of query operators.

Singh et al. logically organize the filesystem into sets of files, called access-control barrels, with identical access privileges of users and groups [13]. The system constructs a separate index per barrel, and restricts query handling to permitted barrels. The authors define the access credentials of users, groups and barrels, and use them as nodes of the access credentials graph. The graph includes edges that minimally connect users to their groups and searchable barrels. The authors safely reduce the number of maintained indices by eliminating from the graph each barrel with number of files less than a configured threshold. Then, they replicate the respective index across the minimal set of nodes that can search the files of the eliminated barrel.

A different study aims to improve metadata search efficiency by hierarchically partitioning the filesystem by access permissions [10]. This approach creates many small partitions, but the authors leave for future study the full merging of partitions with identical permissions. However, the above problem is essentially family clustering with L_s=100% in the context of the present paper.

Encrypted Storage Song et al. describe techniques to securely search remote documents maintained in encrypted form [15]. The client queries the server through a key and a plaintext or encrypted keyword. The server identifies keyword locations through linear scan of the encrypted documents. For large datasets, the server may use inverted index of encrypted keywords, and encrypted or plaintext posting lists. In contrast, the Mafdet system inserts keyed hashes of document keywords into a Bloom filter at the server [1]. Thus, a client only submits keyword hashes to search for documents at the server.

Chang and Mitzenmacher use an encrypted bitmap to encode the presence of particular keywords in a document [6]. The user submits a permuted keyword identifier along with a key to search for the encrypted documents that contain the keyword. The only information leaked to the server is the keyword sharing among the documents. Instead, CryptDB supports keyword search over individually encrypted words of a text column in a relational database [12]. PRISM transforms the problem of keyword search over encrypted files into privacy-preserving map and reduce tasks [4].

Pervez et al. assume that both files and inverted indices are stored in encrypted form at the cloud [11]. Authorized users submit encrypted search criteria to a third party, which homomorphically encrypts them before their transmission to the cloud server. The cloud server uses a user-specific key to re-encrypt the index for query evaluation.

Online Social Networks Keyword search in social networks is possible through a set of inverted indices with each index containing keyword occurrences (posting lists) of documents from particular users. Access control is enforced through intersection of the search result with the identifiers (author list) of documents authored by a particular set of users [2]. The authors examine alternative cost models to optimally include specific friends in the author list of each user, and introduce the HeapUnion operator to efficiently process multiple lists of document identifiers [3].

Hummingbird is a microblogging system that cryptographically hides from a user the topics on which other users follow her, and from third parties the fact that a user follows another user on a specific topic [8]. More generally, Cheng et al. enable fine-grain specification of access-control policies in user-to-user, user-to-resource and resource-to-resource relationships over social networks [7]. Hails provides data-flow confinement at the client and server side so that mutually-untrusted web applications can interact safely [9]. These are more general issues of access control in social networks, and lie beyond the scope of our present study.

6. CONCLUSIONS AND FUTURE WORK

We use clustering to identify documents with similar sets of authorized searchers. Accordingly, we generate shared indices for documents with common authorized searchers of sufficient volume. We experimentally show that with tunable parameters we achieve an indexing organization that combines low number of indices per user with low number of indices per document. In our future work we plan to integrate the Lethe workflow into a distributed search engine and experiment with a broad collection of datasets from collaborative environments, cloud storage and social networks.

7. ACKNOWLEDGEMENTS

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

8. REFERENCES

[1] S. Artzi, A. Kieżun, C. Newport, and D. Schultz. Encrypted keyword search in a distributed storage system. Technical Report MIT-CSAIL-TR-2006-10, CSAIL, MIT, Feb. 2006.

[2] T. A. Bjørklund, M. Götz, and J. Gehrke. Search in social networks with access control. In *Intl. Work. Keyword Search on Structured Data (KEYS)*, pages 4:1–4:6, Indianapolis, IN, June 2010.

[3] T. A. Bjørklund, M. Götz, J. Gehrke, and N. Grimsmo. Workload-aware indexing for keyword search in social networks. In *ACM Intl. Conf.*

Information and Knowledge Management (CIKM), pages 535–544, Glasgow, UK, Oct. 2011.

[4] E.-O. Blass, R. D. Pietro, R. Molva, and M. Önen. PRISM - privacy-preserving search in MapReduce. In *Privacy Enhancing Technologies Symposium*, pages 180–200, Vigo, Spain, July 2012.

[5] S. Büttcher and C. L. A. Clarke. A security model for full-text file system search in multi-user environments. In *USENIX Conf. on File and Storage Technologies (FAST)*, pages 169–182, San Francisco, CA, Dec. 2005.

[6] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Intl. Conf. Applied Cryptography and Network Security*, pages 442–455, New York, NY, June 2005.

[7] Y. Cheng, J. Park, and R. Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *Intl. Conf. Social Computing/Intl. Conf. Privacy, Security, Risk and Trust (SocialCom/PASSAT)*, pages 646–655, Amsterdam, Netherlands, Sept. 2012.

[8] E. D. Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the time of Twitter. In *IEEE Symp. Security and Privacy*, pages 285–299, San Francisco, CA, May 2012.

[9] D. G. Giffin, A. Levy, D. Stefan, D. Terei, D. Maziéres, J. C. Mitchell, and A. Russo. Hails: Protecting data privacy in untrusted web applications. In *USENIX Symp. Operating Systems Design and Implementation (OSDI)*, pages 47–60, Hollywood, CA, Oct. 2012.

[10] A. Parker-Wood, C. Strong, E. L. Miller, and D. D. Long. Security aware partitioning for efficient file system search. In *IEEE Symp. Massive Storage Systems and Technologies*, pages 1–14, Incline Village, NV, May 2010.

[11] Z. Pervez, A. A. Awan, A. M. Khattak, S. Lee, and E.-N. Huh. Privacy-aware searching with oblivious term matching for cloud storage. *Journal of Supercomputing*, 63(2):538–560, Feb. 2013.

[12] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *ACM Symp. Operating Systems Principles (SOSP)*, pages 85–100, Cascais, Portugal, Oct. 2011.

[13] A. Singh, M. Srivatsa, and L. Liu. Search-as-a-service: Outsourced search over outsourced storage. *ACM Transactions on the Web*, 3(4):13:1–13:33, Sept. 2009.

[14] D. K. Smetters and N. Good. How users use access control. In *Symp. On Usable Privacy and Security (SOUPS)*, Mountain View, CA, July 2009.

[15] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symp. Security and Privacy*, pages 44–55, Berkeley, CA, May 2000.

[16] P.-N. Tan, M. Steinbach, and V. Kumar. *Data Mining*, chapter 8. Addison-Wesley, May 2005.