

NS2: Networked Searchable Store with Correctness

Radu Sion, Sumeet Bajaj, ({sion,bajaj}@cs.stonybrook.edu)^{*}
Network Security and Applied Cryptography Lab, Computer Science, Stony Brook University

Bogdan Carbunar (carbunar@motorola.com)
Pervasive Platforms and Architectures, Motorola Labs, Schaumburg, IL 60196

Stefan Katzenbeisser (skatzenbeisser@acm.org)
Institut für Informatik, TU München, Germany

ABSTRACT

In an outsourced data framework, we introduce and demonstrate mechanisms for securely storing a set of data items (documents) on an un-trusted server, while allowing for subsequent conjunctive keyword searches for matching documents. The protocols provide full computational privacy, query correctness assurances and no leaks: the server either correctly executes client queries or (if it behaves maliciously) is immediately detected. The client is then provided with strong assurances proving the authenticity and *completeness* of results. This is different from existing secure keyword search research efforts where a cooperating, non-malicious server behavior is assumed. Additionally, not only does the oblivious search protocol conceal the outsourced data (from the un-trusted server) but it also does not leak client access patterns, the queries themselves, the association between different queries or between newly added documents and their corresponding keywords (not even in encrypted form). These assurances come at the expense of additional computation costs which we analyze in the context of today's hardware.

Introduction

In an increasingly networked world, computing and storage services require security assurances against malicious attacks or faulty behavior. Protecting the confidentiality and integrity of *stored* data is paramount to ensure safe computing. In networked storage, data could be geographically distributed, and thus might be stored on potentially vulnerable remote servers or transferred across untrusted networks; this adds security vulnerabilities compared to direct-access storage. Networked storage architectures are becoming increasingly prevalent: e.g., networked file systems and online relational databases in public and commercial infra-structures such as email and storage portals, libraries, health and financial networks.

Today, sensitive data is being stored on remote servers maintained by third party storage vendors. This is because the total cost

^{*}The author is partly supported by the NSF Cybertrust award CNS-0627544 and by the Office of the Vice President for Research of Stony Brook University.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '07, September 23-28, 2007, Vienna, Austria.

Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

of storage management is 5–10 times higher than the initial acquisition costs [10]. Moreover, most third party storage vendors do not provide strong assurances of data confidentiality and integrity. For example, personal emails and confidential files are being stored on third party servers such as Gmail [2], Yahoo Mail [4], Xdrive [3], and Files Anywhere [1]. Privacy guarantees of such services are at best declarative and often subject customers to unreasonable fine-print clauses—e.g., allowing the server operator (and thus malicious attackers gaining access to its systems) to use customer behavior and content for commercial, profiling, or governmental surveillance purposes [9]. To protect data stored in this untrusted server model, security systems should offer users assurances of data confidentiality and access pattern privacy. However, a large class of existing solutions delegate this by assuming the existence of co-operating, non-malicious servers.

Additionally, the ability to search easily and efficiently within remote data is a very important feature. Some efficient content-based keyword search indexing schemes exist today. However, supporting content-based search *with* privacy in a secure remote storage is hard, and often tends to compromise either security or performance significantly. For example, if data is stored encrypted on a remote server, to perform content-based searches, one cannot afford to decrypt it at the server nor transfer the bulk of the encrypted data to the client; the former compromises security as a potentially untrusted server needs to know decryption keys, and the latter compromises performance because of huge data transfers.

In this paper we explore three important security dimensions of networked storage with untrusted servers, in the context of advanced content-based search: (1) confidentiality, (2) access pattern privacy, and (3) retrieval correctness (result completeness and data integrity). We design efficient search algorithms so as to scale to a large number of documents, keywords and queries. Our solution provides content-based searches with the following security assurances:

Confidentiality. The data being stored on the server is not decipherable either during client-server transit, or at the server side, even by a malicious server.

Privacy of Searches. An intruder or a malicious server is not able to perform statistical attacks by exploiting search patterns. For example, it is not able to compromise confidentiality by correlating known public information with frequently searched items.

Retrieval Correctness. Clients are able to verify the integrity and completeness of any results the server returns. For example, when a client sends the server a keyword-based query, the client is able to verify that the server returned *all* matching documents.

To ensure confidentiality, all data and associated meta-data are encrypted at the client side using non-malleable encryption, before being sent to the server. The data remains encrypted throughout its

lifetime at the server. In response to retrieval requests, the server sends encrypted documents which are decrypted by the client. To enable content-based search on encrypted data, any stored documents are indexed securely in the process of storage, at the data owner’s site. This results in the confidential storage of the index structures at the server side, available for future secure client access. We ensure access pattern privacy by both cryptographic obfuscation of the index and the deployment of practical information retrieval schemes with privacy. Retrieval correctness can be ensured partly by deploying a secure, incremental checksumming technique allowing also for data updates.

Model

In the model considered, a server offers data hosting services; for example it may store files, e-mails or relational data. Clients need to perform queries on the stored data items (“documents”) while revealing a minimal amount of information (preferably none) to the server. In this paper we are dealing with applications where documents are retrieved according to keywords. More precisely, we assume that a number of keywords is associated with each outsourced document; later, a client must be able to retrieve *all* documents matching one or more keywords. In addition to document retrieval queries, the clients must be able to add new documents and remove or update arbitrary previously-stored ones.

In a malicious adversarial setting, *correctness* and *completeness* of server replies — besides confidentiality and integrity of the documents — are of crucial importance. In this setting, query correctness assures that only documents are returned that match the query; query completeness assures that *all* such documents are returned by the server. Additionally, it is often essential to conceal client access patterns from the server. Here we introduce query protocols for secure indexed data storage that provide confidentiality, completeness, privacy of search patterns; and the immediate detection of a dishonest server by the client.

Let $\mathbb{D} = (d_i)_{i=1,\dots,n}$ be the outsourced documents. Let $\mathbb{K} = (k_i)_{i=1,\dots,k}$ be the set of keywords associated with the documents and k their total number. Both the stored documents in \mathbb{D} as well as the query keywords $q = \{k_{i_1}, k_{i_2}, \dots, k_{i_q}\}$ will be encrypted with a symmetric key under the client’s control.

For analysis, we represent both the server and the client as interactive polynomial-time Turing Machines; we write CLI for the client and SERV for the server machine. A client interacts with the server and issues a sequence of update requests U_1, \dots, U_l ; here, an update is either a document addition or removal request. We call such a sequence of requests a *trace* \mathcal{T} . In addition to update queries the client can also issue document retrieval queries $Q = (k_1, \dots, k_i)$, indicating that the client wishes to obtain all documents that match (e.g., contain) all keywords k_1, \dots, k_i . After executing a retrieval query the client Turing Machine either outputs \top or \perp , indicating whether the client accepts or rejects the server response (denoted as $D_{\mathcal{T},Q}$) – in the first case the client believes that the server replied honestly. We write $\text{CLI}(\mathcal{T}, Q, D_{\mathcal{T},Q}) \in \{\top, \perp\}$ to denote the output of the client as a result of the server execution of trace \mathcal{T} and query Q and the result $D_{\mathcal{T},Q}$. A server response D is said to be *consistent* with both \mathcal{T} and Q , if an honest server, after starting at the empty database and executing trace \mathcal{T} honestly, would reply with D to the query Q . Two traces \mathcal{T} and \mathcal{T}' are called *similar* with respect to Q , written as $\mathcal{T} \approx_Q \mathcal{T}'$, if the query Q yields the same answer when queried after a trace \mathcal{T} or \mathcal{T}' , i.e., $D_{\mathcal{T},Q} = D_{\mathcal{T}',Q}$.

DEFINITION 1. A query protocol is *complete*, if (except with negligible probability [11]) for all traces \mathcal{T} and \mathcal{T}' with $\mathcal{T}' \not\approx_Q \mathcal{T}$,

document retrieval queries Q and server responses $D_{\mathcal{T}',Q}$, we have $\text{CLI}(\mathcal{T}, Q, D_{\mathcal{T}',Q}) = \perp$.

Informally, a query mechanism is *complete*, if the server is bound to the sequence of update requests performed by the client: either the server responds correctly to a query or its malicious behavior is immediately detected by the client.

Information Leaks. Even though all documents are stored in encrypted form on the server, the queries performed by the client (the *client access patterns*) leak (potentially essential) information about the data. A curious server can, for example, perform statistics on the search queries and relate the queries to corresponding documents. In the following we provide an informal classification of such information leaks that are *introduced either by the query protocol itself or by the storage data structures on the server*.

In order to perform a document retrieval request, the client will need to submit a query, naturally composed of a set of information items (*keyword tokens*) that relate uniquely to the queried keywords. In a very simple query protocol, these tokens could be the (encrypted) actual query keywords. Depending on the information that is obtained by the server from the keyword tokens, we can distinguish several leak types:

A *type 1 leak* occurred if, after receiving and executing a query, the server can systematically construct any association between the already seen keyword tokens (including the ones for the current query) and the query results (encrypted documents) returned so far.

A *type 2 leak*, which is arguably more undesirable, allows the server to construct a mapping between *each and every* considered keyword tokens and all stored documents.

A *type 3 leak* leak occurs in the process of adding new keywords if as a result of a query the server knows that documents that were previously stored do not contain the added new keywords.

Solution Outline

Our solution is composed of a set of layered mechanisms that operate together to provide security assurances. For confidentiality, non-malleable, semantically secure, symmetric encryption is deployed. Correctness assurances are achieved by maintaining a minimal set of client-side checksums that can operate for dynamic access patterns (e.g., document removals). Client access privacy is provided by a combination of custom private information retrieval protocols, together with search index obfuscation mechanisms. In the following we briefly outline these mechanisms. We will start by discussing search query correctness assurances.

We will represent the server-side search index as a collection of *posting lists*, or PL for short. For each search keyword k_i we maintain PL_{k_i} containing all document identifiers of documents associated with it. To avoid possible manipulations of the PLs by the server, we keep a cryptographic checksum *hash* of each PL on the client; for this purpose, we use a special purpose hash function $H(\text{PL}_{k_i})$ that is able to hash sets. The server maintains the PL elements encrypted: each PL_{k_i} contains encrypted document identifiers of documents that match keyword k_i . To prevent the server from cross-correlating different PLs, we use a different encryption key for each set K_{k_i} . We compute this key from a fixed master secret key and the keyword name using a one-way random cryptographic hash function or a HMAC [14].

For correctness, upon issuing a search query, clients will be able to check server replies by using these set hashes. In other words, upon receipt of these sets, the client first checks whether the server honestly returned the PLs, i.e., the client computes all hash values $H(\text{PL}_{k_1}), H(\text{PL}_{k_2}), \dots, H(\text{PL}_{k_n})$ and compares them with the locally stored values. If at least one hash differs, the client outputs

\perp and assumes that the server is malicious. Otherwise, the client computes the encryption keys for the PLs from its master key and decrypts all elements in the received PLs. Finally, the client computes the intersection of the received PLs, requests the corresponding documents from the server and outputs \top .

Due to special properties of the set hashes, adding documents to or removing documents from the server can be done in a straightforward manner, with minimal communication overheads. The client hash values can be updated directly upon removal / addition of documents, without retrieving additional data from the server. Without discussing further implementation details (the *incremental hashing* paradigm of Bellare and Micciancio [8] is deployed to construct this function), we point out that it can be shown that the above query protocol provides query completeness if the hash function H is *collision-resistant*, i.e., if it is computationally infeasible to find two sets A and B with $A \neq B$ and $H(A) = H(B)$:

THEOREM 1. *The above query protocol provides completeness if H is collision-resistant.*

Even though the proposed protocol is provably complete, the solution does not achieve the desired privacy properties. In a static setting (where no documents are added or removed) it exhibits a type 1 leak, since the server can map the queried encrypted PLs to the retrieved encrypted documents. In a dynamic setting, it even exhibits a type 2 leak: each time a document is added, for consistency, the client needs to update the PLs corresponding to keywords contained in the document. The server can then map these PL updates to the newly added documents. In the following we show how more sophisticated access protocols and data structures for representing the PLs can reduce information leaks.

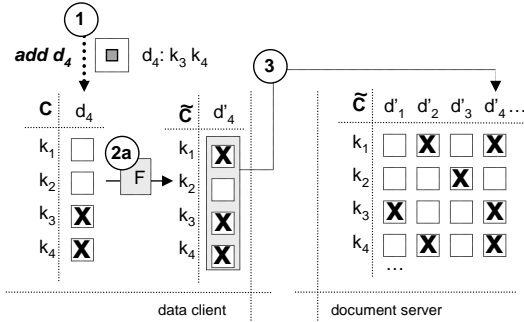


Figure 1: Adding a new document requires (1) creating a vector indicating the contained keywords, (2a) encrypting the vector with F , (2b) permuting the column according to σ (not shown) and (3) sending the result to the server.

The leak source of the above methods is the client signaling the server the keywords contained in newly added document. To avoid this leak, we will modify document addition such that the server learns nothing of the new document through the addition. We now represent the PLs in the form of a $k \times n$ binary matrix C . The bit at row i and column j , denoted $C_{i,j}$, is set to one if and only if keyword i is contained in document j . On the server we store an “obfuscated” version of the matrix, \tilde{C} , computed by applying F , a bit-wise pseudo-random function [11] and σ , a random permutation of $1, \dots, k$. The matrix element $\tilde{C}_{i,j}$ is given by $\tilde{C}_{i,j} = \text{lsb}(F(k_{\sigma(i)} \| d_j)) \oplus C_{\sigma(i),j}$, where \oplus denotes the XOR operation, and lsb denotes the least significant bit of a string. Now keyword k_i corresponds to row $\sigma(i)$ of \tilde{C} . While the pseudo-random function assures proper encryption of the matrix C , the

permutation assures that the server cannot infer information on the keywords k_i by looking at the order in which they are represented in \tilde{C} .

To add a new document, the client will now construct a column having ones (1s) only in positions corresponding to keywords contained in the document and encoding this column using the permutation σ and the function F . The resulting column, along with an encrypted version of the document are then sent to the server, who appends the column to his matrix \tilde{C} and stores the document using its unique identifier.

In a conjunctive keyword search $\{k_{i_1}, k_{i_2}, \dots, k_{i_m}\}$, the client requests rows $\sigma(i_1), \sigma(i_2), \dots, \sigma(i_m)$ of the matrix \tilde{C} from the server, which correspond to the PLs of the searched keywords; we denote these rows by $C_{\sigma(i_1)}, \dots, C_{\sigma(i_m)}$.

The client reconstructs the rows $\tilde{C}_{i_j}, 1 \leq j \leq m$, by computing the XOR of the received row $\tilde{C}_{\sigma(i_j)}$ with a vector formed of the values $\text{lsb}(F(k_{i_j} \| d_l))$, for $1 \leq l \leq n$. The reconstructed rows $\tilde{C}_{i_1}, \dots, \tilde{C}_{i_m}$ uniquely correspond to the PLs $\text{PL}^{k_{i_1}}, \dots, \text{PL}^{k_{i_m}}$: $\text{PL}^{k_{i_j}}$ contains all document names d_l with $\tilde{C}_{i_j,l} = 1$. Now, the protocol continues in a similar manner as before: the client computes the $H(\text{PL}^{k_{i_j}})$ checksum for each requested keyword and checks whether it matches the locally stored value. If any one hash value differs, the client outputs \perp . Similar to Theorem 1 the completeness of the query protocol can be established. This solution exhibits a type 1 leak however, since the client reveals row indexes corresponding to the searched keywords and also the encrypted documents containing them.

In the following we propose a method that prevents even such leaks (by deploying a variation of computational PIR) at the expense of additional computation costs. The only information leaked to the server consists of the number of keywords contained in conjunctive queries, shared by sets of documents. To achieve this goal, we deploy a modified version of the Computational PIR mechanisms of Kushilevitz and Ostrovsky [12]. Initially, the client randomly chooses two prime numbers p and q of equal bit length, computes their product, $N = pq$ and sends it to the server.

To perform a conjunctive keyword search $\{k_{i_1}, k_{i_2}, \dots, k_{i_m}\}$, for each keyword k_{i_j} a client deploys PIR to obtain the row \tilde{C}_{i_j} without leaking to the server the row index i_j , as follows. It creates k numbers (one for each stored keyword) s_1, s_2, \dots, s_k , such that the i_j -th number s_{i_j} , corresponding to the row i_j of C , is a quadratic non-residue (QNR) and the rest are quadratic residues (QR) in \mathbb{Z}_N^* . The client sends s_1, s_2, \dots, s_k to the server. For each column c in the bit matrix \tilde{C} , the server computes exactly one value v_c as $v_c = \prod_{i=1}^k v_{ic}$, where v_{ic} corresponds to the i -th row of column c and is computed in the following fashion. If $\tilde{C}_{ic} = 0$ then $v_{ic} = 1$, otherwise, $v_{ic} = s_i$. The server sends the computed values (for all columns) v_1, v_2, \dots, v_n to the client, who checks their quadratic residuosity in \mathbb{Z}_N^* . Then, if v_c is a quadratic residue, $\tilde{C}_{p_i,c}$ is known to be 0, otherwise it is 1. Since s_{p_i} is a quadratic non-residue, $v_{p_i,c}$ is a quadratic residue if and only if $\tilde{C}_{p_i,c}$ is 0 (see [12]). The remaining steps of this solution follow exactly the protocol presented above. Again, similar to Theorem 1 the completeness of the query protocol can be established. Moreover:

THEOREM 2. *If the quadratic residuosity assumption holds then the above protocol offers full computational access pattern privacy. It only leaks the number of keywords in a conjunctive query.*

Scaling Up. Multiple Clients. Dynamic Data.

The server side complexity of the oblivious keyword search protocol presented above grows linearly with the number of indexed key-

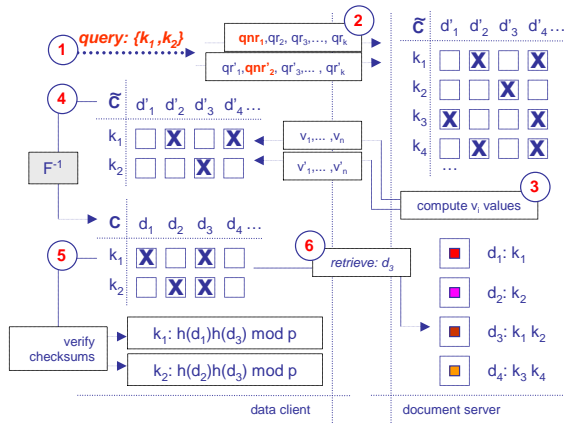


Figure 2: Oblivious Keyword Search.

words and documents. To reduce this computation overhead, two techniques can be used, either separately or in conjunction: (1) To keep the number of indexed keywords small, an additional search structure can be deployed, which allows searching for each keyword exactly once – not yet indexed keywords can be first searched for in the additional data structure and then obviously added to the index. (2) The index can be accessed more efficiently in a partitioned fashion, at the expense of reduced privacy (Figure 3).

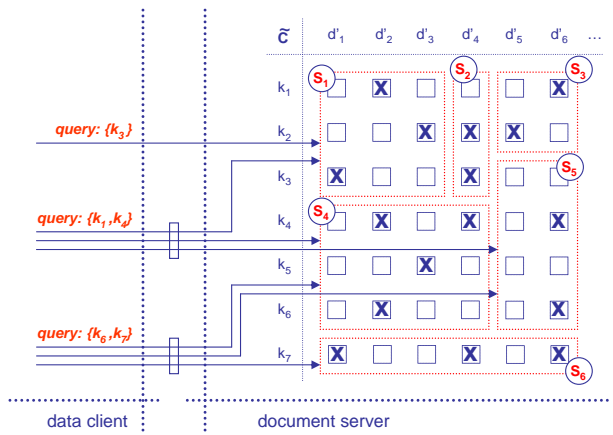


Figure 3: Index segmentation.

To enable multiple client instances to access shared data, the (encrypted) set hashes can be placed on the server to be accessed by the different instances. For dynamic data, commitment protocols will need to be put in place in conjunction with exclusive write locks. Moreover, now the clients need to defend against a server aiming to present different versions of the universe to each client, e.g., by ignoring the other clients’ updates. This can be achieved in a fashion similar to SUNDR [13], by requiring clients to share some mutual awareness state about their transactions, e.g., in the simplest case, client instances will be required to share last-time-of-update state as well as authentication and encryption secrets. If such inter-instance interaction is not possible however, multi-client access becomes a harder problem. It can likely be solved by deploying *trusted hardware* [5, 7] at the server side.

Future Work

Several ongoing results impact the future of this work. Specifically, in recent research [15, 6] we explored the limits of single-server computational private information retrieval (PIR) for the purpose of preserving client access patterns leakage. We realized that deployment of non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude less time-efficient than trivially transferring the entire database. These results are beyond existing knowledge of mere “impracticality” under unfavorable assumptions. They rather reflect an inherent limitation with respect to modern hardware, likely the result of a communication-cost centric protocol design. We argued that this is likely to hold on non-specialized traditional hardware in the foreseeable future. We also validated our reasoning in an experimental setup on modern off-the-shelf hardware.

This result has a direct implication in the NS2 work. Specifically, in the privacy setup, deploying PIR is likely to be less efficient than trivially transferring the relevant index information (we left the PIR description above in place for completeness and to allow the understanding of this intuition). However, in the immediate future it is important thus to develop novel PIR-compatible private information access mechanisms that can be applied in this and other related contexts.

1. REFERENCES

- [1] FilesAnywhere. Online at <http://www.filesanywhere.com/>.
- [2] Gmail. Online at <http://gmail.google.com/>.
- [3] Xdrive. Online at <http://www.xdrive.com/>.
- [4] Yahoo Mail. Online at <http://mail.yahoo.com/>.
- [5] Trusted Computing Group. Online at <https://www.trustedcomputinggroup.org/>, 2005.
- [6] Achieving Practical Private Information Retrieval (Panel). Online at <https://www.cs.stonybrook.edu/~sion/research/PIR.Panel.Securecomm.2006/>, 2006.
- [7] IBM Cryptographic Hardware. Online at <http://www-03.ibm.com/security/products/>, 2006.
- [8] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Proceedings of EuroCrypt*, 1997.
- [9] CNN. Feds seek Google records in porn probe. Online at <http://www.cnn.com>, Jan. 2006.
- [10] Gartner, Inc. Server Storage and RAID Worldwide. Technical report, Gartner Group/Dataquest, 1999. www.gartner.com.
- [11] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.
- [12] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings of FOCS*. IEEE Computer Society, 1997.
- [13] J. Li, M. Krohn, D. Mazières, and D. Shasha. Secure Untrusted Data Repository (SUNDR). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI 2004)*, pages 121–136, San Francisco, CA, December 2004. ACM SIGOPS.
- [14] B. Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. Wiley & Sons, 1996.
- [15] R. Sion and B. Carbunar. On the Computational Practicality of Private Information Retrieval. In *Proceedings of the Network and Distributed Systems Security Symposium*, 2007. Stony Brook Network Security and Applied Cryptography Lab Tech Report 2006-06.