Secure Anonymous Database Search

Mariana Raykova
Columbia University
mariana@columbia.edu

Binh Vo
Columbia University
binh@columbia.edu

Steven Bellovin
Columbia University
smb@columbia.edu

Tal Malkin
Columbia University
tal@columbia.edu

1. PROBLEM STATEMENT

Often, different parties possess data of mutual interest. They might wish to share portions of this data for specific ends, but consider the leak of unrelated portions to be a privacy issue. Thus, methods that provide a well-defined and secure sharing of the data between untrusting parties can be useful tools. One such method that we introduce in this paper provides the ability for a client to search the information residing on a server without revealing to the server his identity or the content of his query, while also guaranteeing that query capability is only granted to appropriate clients and that they do not learn anything unrelated to the query. In addition, the very fact that a client is running certain queries is considered sensitive, and thus both his identity and the query content must be protected from the server. Such a tool is useful in deciding and agreeing upon information-sharing between parties who do not initially know if they have data worth sharing with each other, and do not want to share information until they do.

We address the above concerns by defining and implementing Secure Anonymous Database Search (SADS). Although the framework can support more general queries, we focus here on the specific functionality of keyword search, which allows an authorized client to anonymously and securely query a server for documents containing a desired keyword. We design an efficient SADS scheme, and provide for it proofs of security and performance evaluation.

2. SECURITY ARCHITECTURE

2.1 Problem Setting and Requirements

The general scenario we consider involves multiple parties who possess private sensitive data, which they are willing to share in a limited fashion. Each transaction in the scheme will involve a party who owns a set of documents he wishes to make available for secure anonymous keyword search by authorized parties. Any party may be authorized by the data owner to take the role of the querier, whose input is some keyword that he wishes to search for in the database. We will interchangeably refer to the first party as data owner or server, and the second party as querier or client. Our protocol will meet the following requirements: *correctness, client security, server security, server access control, client anonymity. Practical efficiency* is a central requirement for our system, and we design our model and protocol accordingly. In particular, high communication complexity, or per-query computation complexity that scales linearly with the number of words in a document will not be acceptable. This rules out the use of existing generic cryptographic techniques from secure multiparty computation [5,10,11] and PIR [3,4].

2.2 The SADS Model and Protocol Structure

Security, anonymity, and efficiency can be conflicting goals, and cannot all be achieved simultaneously without adjusting the model. For example, sublinear computation and constant-time communication conflict with client privacy, as they cannot both be achieved without allowing the server to gain information about the query results, and thus on the query itself. Client anonymity conflicts with server access control, and obviously anonymity cannot be achieved if the server and client are the only two parties participating in the interaction. Trying to solve the latter problem by involving all parties in the system for each search is not practical for efficiency. Furthermore, it would require a fixed and known set of parties.

To address this, we expand our model by adding two new parties that will participate in each search, the *Index Server (IS)* and the *Query Router (QR)*. These may be viewed as neutral parties trusted with the responsibility of regulating the data sharing process, but not trusted to see the participants' private inputs. The security and anonymity requirements with respect to these new parties will be reasonable, but weaker than those between the client and server; in return, they allow us to achieve practical efficiency. We will now overview the general architecture of our SADS protocol, demonstrating the roles of IS and QR and their trust implications.

Figure 1 illustrates the search protocol. A database owner (P1) generates a search structure computed from (an encryption of) his data and gives it to the index server. This structure enables IS to answer (encrypted) queries but does not reveal information about the provided database. Outsourcing the search to IS prevents the data owner from finding out the results to encrypted queries. The IS sees the

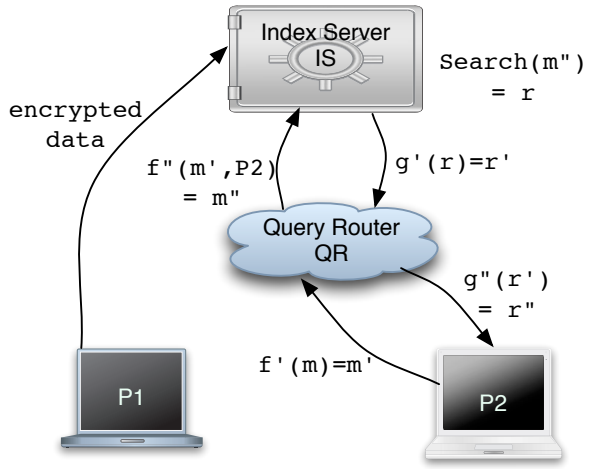

Figure 1: General Setup: P1 makes its data available for search providing IS with search structures, P2 submits keyword queries anonymously to IS via QR, IS sends back the search result though QR

results, but does not know what documents they correspond to. At most, the IS will be able to tell when two submitted queries have overlapping results. This is further mitigated by preserving the anonymity of the queriers with respect to the index server. However, providing such anonymity introduces a new problem: how to guarantee that only authorized users are submitting queries. This is addressed by the query router, who serves as an intermediary in the communication path between querier and IS. QR is trusted to know and protect the identities of the participants, while enforcing correct authorization before allowing queries to reach the IS. However, he is not trusted to see the content of the queries or results. Thus a querier (P2 in Fig. 1) will submit his encrypted query to the QR, who checks the authorization of the user, transforms the query and forwards it to the IS. The IS will send back search results to the QR, which will be able to forward them to the respective user. The results are encrypted so that the QR does not learn their content.

With this architecture in mind, we make the following requirements with respect to IS and QR: *data security against IS and QR, client anonymity against IS, clients result-security-up-to-equality against IS, client query-security-up-to-equality against QR*

3. SADS PROTOCOL

3.1 Building Protocols

Re-routable Encryption Re-routable encryption is a new primitive we will use in our system to protect identities, when routing (encrypted) queries from an authorized client to IS, and also when routing the (encrypted) results back to the client. Informally, re-routable encryption is a protocol to send an encrypted message, or some function of the message, from a sender to receiver through a query router QR, such that two security requirements are satisfied. First is the security of the sender's message with respect to QR, and second is the anonymity of sender and reciver with respect to each other.

DET-CCA Deterministic Private Key Encryption While the standard definitions of security (e.g., [6]) require an encryption scheme to be probabilistic, a deterministic scheme will allow us considerable efficiency gains, while still providing a level of security which is acceptable in our setting (security-up-to-equality). This tradeoff follows the idea introduced by [7], who define deterministic encryption in the public-key setting, and show how to convert a standard (probabilistic) PKE to a deterministic one. We follow the same approach, adapting it to the secret key setting and defining DET-CCA security. We instantiate the above deterministic private key encryption scheme following the construction of RSA-DOAEP in [7] but with different primitives that give more security and the group property that we need. We use the Pohlig-Hellman (PH) function [8] and the SAEP+ (short for Simple-OAEP) padding construction introduced in [2].

Bloom Filters

The deterministic encryption scheme that we presented provides ciphertexts that are suitable to be used in efficient search protocols according to [7]. Bellare et al. in [7] suggest that the search functionality over encrypted data produced with a deterministic encryption should be realized by attaching "tags" that will be easily searchable and easily computed by both the querying party and the server. We realize this with a Bloom filter [1]. This allows efficient search, guarantees there will be no false negatives, and allows a tunable rate of false positives.

3.2 Secure Anonymous Database Search

We now present the SADS scheme that allows a data owner to make its database available for search. To do so we compute BF structures for encrypted search on it and send them to an index server, which executes queries submitted to it anonymously by authorized queriers via a query router. We use two instantiations of re-routable encryption: one for query submission instantiated with (DET-CCA secure) PH-DSAEP+, where the QR computes the first BF indices of the encrypted query before passing them on to IS. And another for returning query results to the querier, instantiated with (IND-CCA secure) PH-SAEP+ directly. SADS protocol involves the following parties: a server(S), a client (C), a query router (QR) and an index server (IS) and consists of the following five stages:

Preprocessing: S generates for each of its documents a Bloom filter from the encryptions of its stemmed keywords under PH-DSAEP+ with its private key.

Key Generation: To authorize C for search S, QR and C generate keys for query submission and return the result in encrypted form.

Query Submission: To submit an encrypted query for keyword W, C encrypts it under its private submission key and sends it to QR, which converts the received ciphertext to the key of the server, extracts the BF indexes that it defines, and sends them to IS.

Search: IS runs BF search on the received indexes to get the result R.

Query Return: IS encrypts the obtained result with its private return key and sends it to QR. QR then transforms the ciphertext to the private key of the corresponding client and sends it to C, which decrypts it to obtain the result R.

4. EFFICIENT STORAGE AND EFFICIENT BLOOM FILTER SEARCH

To minimize the number of bits that need to be read to satisfy queries across a large number of Bloom filters, we store them in transposed order. First, they are divided into blocks of filters; within each block, all bits from a single index across the filters are stored contiguously. Thus, each document is represented by a bit across the same position within multiple slices, one slice for each index of its Bloom filter representation. To run a query, we need only fetch those slices which correspond to the indices of the query term, which is a large savings since normally we would have to read the full contents of every Bloom filter for every document for any query. This technique is referred to as bitslicing and has been studied as a method for storing signature files in database indexes [12]. We apply slicing optimizations that help minimize the number of blocks read from memory and run BF search in parallel on many documents. While supporting AND queries is trivial, we also achieve improvements in the run times of OR queries by running the queries over all terms in parallel, thus avoiding multiple reads of slices that coincide.

5. PERFORMANCE

We implemented our system in C++ to demonstrate practicality of use. We ran experiments on a Ubuntu 8.04 Linux PC with a Pentium 4, 3.4 GHz cpu and 2GB of RAM. A variety of corpus sizes from 1K to 50K were extracted from the Enron Email Dataset, available at `http://www.cs.cmu.edu/~enron/`. Each document was stemmed using the techniques provided by the Clair library [9], and the stems were inserted into the Bloom filter index per document. Bloom filter sizes were computed to give a false positive rate of 0.1% based on the number of stems we wished to be able to index.

Before running queries, we extracted a subset of query terms, and grouped them by document frequency within the database. In each experiment, we ran a total of 100 queries and took the average time to completion for each query. If we had less than 100 terms to query on, we cycled through the existing ones, spacing out identical queries to minimize artificial cache gains. Figure 2 shows the average time per query plotted against the size of the corpus the index was computed from. This relationship is shown for each of four different types of queries based on frequency of the query terms being searched on. The four frequency groups used were: **0-Freq** - terms which do not appear anywhere in the corpus; **Low Freq** - terms which appear in 1 or 2 documents; **Med Freq** - terms which appear in 45-55% of the corpus; **High Freq** - terms which appear in all but 1 or 2 of the documents.

Figure 3 shows the average time per query plotted against OR queries as a ratio against the amount of time it would take to run these queries individually and union the results

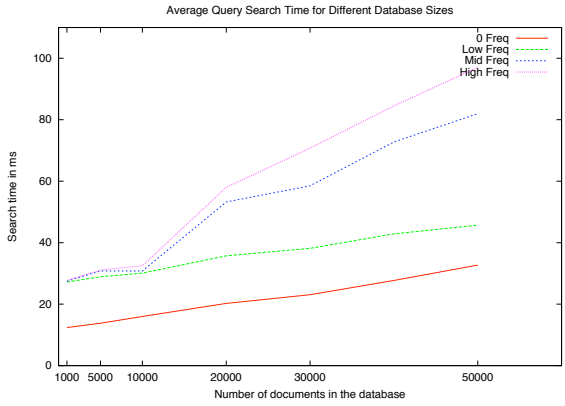

Figure 2: Search Times

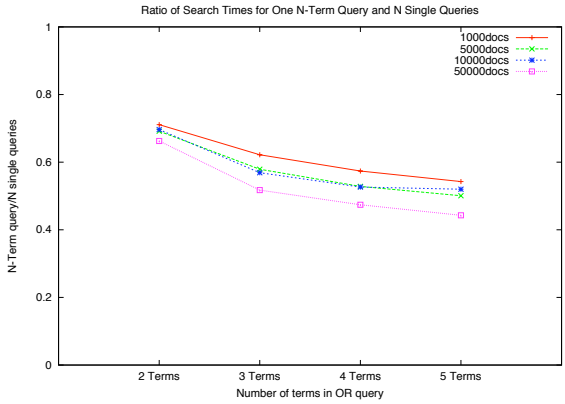

Figure 3: Ratio of OR Query time vs Individual Queries

afterwards. As we can see the savings are significant, and grow more so as the number of terms increases. When running these terms in parallel as an OR query, a slice fetched remains in memory and can be checked against each query quickly. When running them separately, they must be fetched multiple times. As we can also see, this effect grows less pronounced with larger corpus sizes, since with smaller corpus sizes there is an increased likelihood that slices will remain cached from previous runs even while running the queries individually.

6. REFERENCES

[1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[2] Dan Boneh. Simplified OAEP for the RSA and Rabin functions. *Lecture Notes in Computer Science*, 2139:275–291, 2001.

[3] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*,

45(6):965–981, 1998.

[4] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Sciences*, 60(3):592–629, 2000.

[5] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[6] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[7] A. Boldyareva M. Bellare and A. O'Neill. Deterministic and efficiently searchable encryption. In *Proceedings of CRYPTO'07*, 2007.

[8] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms overgf(p)and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[9] Dragomir R. Radev, Mark Hodges, Anthony Fader, Mark Joseph, Joshua Gerrish, Mark Schaller, Jonathan dePeri, and Bryan Gibson. Clairlib documentation v1.03. technical report cse-tr-536-07. *University of Michigan. Department of Electrical Engineering and Computer Science*, 2007.

[10] Andrew Chi-Chih Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[11] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[12] Justin Zobel and Alistair Moffat. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23:453–490, 1998.