Search on Encrypted Data

Hakan Hacıgümüş[1], Bijit Hore[2], Bala Iyer[3], and Sharad Mehrotra[4]

[1] IBM Almaden Research Center hakanh@acm.org
[2] Donald Bren School of Computer Science
University of California, Irvine bhore@ics.uci.edu
[3] IBM Silicon Valley Lab balaiyer@us.ibm.com
[4] Donald Bren School of Computer Science
University of California, Irvine sharad@ics.uci.edu

1 Introduction

The proliferation of a new breed of data management applications that store and process data at remote locations has led to the emergence of search over encrypted data as an important research problem. In a typical setting of the problem, data is stored at the remote location in an encrypted form. A query generated at the client-side is transformed into a representation such that it can be evaluated directly on encrypted data at the remote location. The results might be processed by the client after decryption to determine the final answers.

1.1 Motivation: Database as a Service

The primary interest in search over encrypted data has resulted from the recently proposed *database as a service* (DAS) architecture [10, 26, 12, 7]. DAS architecture is motivated by the *software as a service* initiative of the software industry, also referred to as the application service provider (ASP) model. Today, efficient data processing is a fundamental need not just for academic and business organizations, but also for individuals and end customers. With the advent of new technologies and multimedia devices, the amount of data an average person produces in the form of emails, image and video albums, personal records (e.g., health care, tax documents, financial transactions, etc.) that they need to store and search is rapidly increasing. Effective management of large amounts of diverse types of data requires powerful data management tools. Unfortunately, expecting end-users to develop the ability to install, administer, and manage sophisticated data management systems is both impractical and infeasible. Likewise, in an organizational setting, expecting small or medium size corporations to hire professional staff to manage and run corporation's databases is an expensive and, at times, a cost-prohibitive alternative [12].

The "database as a service" (DAS) model, that offers variety of data management functionalities in the form of a service to clients, is an emerging alternative that overcomes many of the above listed challenges of traditional architectures. A DAS model consists of the following entities:

- **Data Owner**: This is the side that produces data and owns it. It is assumed to have some limited computational resources and storage capabilities but far less than the server.
- **Server**: The remote service provider, or the server stores and manages the data generated by data owners. The service provider supports powerful and intuitive interfaces for data owners/users to create, store, access and manipulate databases. The task of administering the database (e.g., installation, backups, reorganization, migration, software updates, etc.) is entirely taken over by the service provider.

- **Data Clients**: A data client could either be the same as the data owner or if the owner is an organization, it could be its employees and/or its clients (e.g., a bank is the data owner and individuals having accounts with the bank are the data clients). The data clients can only access data according to the access control policies set by the data owner, e.g., a data manager might access all data, a bank-client may only access his personal account data etc.

The DAS architecture offers numerous advantages including lower cost due to the economy of scale, lower operating costs, etc. and enhanced services (e.g., better reliability and availability, access to better hardware and software systems, improved data sharing etc.). Today, DAS model is available in certain vertical market segments – e.g., email service through Yahoo!, MSN, Google, etc., as well as photo albums through companies such as Shutterfly. With the advantages the model offers and the related commercial activities, it is foreseeable that DAS architecture will permeate numerous other consumer as well as business application domains in the near future.

The key technological challenge in DAS is that of *data confidentiality*. In the DAS model, user data resides on the premises of the service provider. Most corporations (and individuals) view their data as a valuable asset. The service provider needs to provide sufficient security measures to guard the data confidentiality. In designing mechanisms to support confidentiality and privacy of user's databases, a key aspect is that of *trust*, i.e., how much trust is placed on the service provider by the data owner.

If servers could be completely trusted, the features are very similar to that of a normal database management system which would have been deployed within the organization of the data owner had the organization chosen to do so. From the security perspective, the service provider has to enable traditional access-control and other network security measures to prevent unauthorized access and in general prevent malicious outsiders from launching any kind of disruptive attacks. Furthermore, service providers may employ additional security mechanisms to ensure safety even if data is stolen from organizations by storing data on disks in an encrypted form [17, 12].

The nature of data processing starts to change when the level of trust in the service-provider itself begins to decrease from complete to partial to (perhaps) none at all! Such a varying trust scenario necessitates the usage of various security enhancing techniques in the context of DAS. The most popular trust model studied is the *passive adversary* or *curious intruder* model. Here, the server-side is considered truthful, in that model, the server implements various data storage and query processing functionalities correctly. The passive adversary is one or more malicious individual(s) on the server-side who has access to the data, e.g., a database administrator. A passive adversary only

tries to learn sensitive information about the data without actively modifying it or disrupting any other kind of services [5].

Almost all the proposed solution approaches in literature employ encryption to protect the customers' data. The data is encrypted in a variety of manner, i.e., at different granularity, using different encryption algorithms etc. Since all customer data is encrypted while stored on the server, the key challenge becomes that of implementing the required data modification/querying functionalities on this encrypted data – the topic of this chapter.

1.2 Overview of Problems Studied in Literature

Techniques to support search over the encrypted data depends upon the nature of data as well as on the nature of search queries. The two data management scenarios that have motivated majority of the research in this area are:

- **Keyword-based search on encrypted text documents**: The most common setting is that of a remote (semi-trusted) email server which stores encrypted emails of users and allows them to search and retrieve their emails using keyword-based queries [7, 21, 6, 22].
- **Query Evaluation on encrypted relational databases**: The setting of this problem is that of a remote (semi-trusted) relational database management system, which stores clients' relational data and allows users to search the database using SQL queries [10, 11, 12, 14, 3, 4].

In this chapter, we discuss advances in both of the problem settings. We note that the solutions depend upon the particular instantiation of the DAS model being studied. The general model allows for multiple data owners who outsource their database management functionalities to the service provider. Each such owner might have multiple clients who access various functionalities from these services. In cases where the client is a different entity from the owner, different models of data access may be enforced, for instance a session might require the client to connect via the data owner's site onto the service provider. Alternatively it might be a direct session with the service provider, which does not involve the owner. Other models can be seen as the simplifications of this general architecture. Different DAS models pose new issues/challenges in ensuring data confidentiality.

For most of this chapter, as is the case with the current literature, we will make a simplifying assumption that the data owner and client are the same entity. We will discuss some of the challenges that arise in generalizing the model towards the end of the chapter. We begin by first discussing approaches to support text search, which is then followed by techniques to support relational/SQL queries.

[5] Almost all the approaches we describe in this chapter address data confidentiality and privacy issues for the passive adversarial model. We will refer to this model interchangeably as the *semi-trusted* model.

2 Keyword search on encrypted text data

In this section we discuss approaches proposed in the literature to support keyword based retrieval of text documents. We begin by first setting up the problem. Let Alice be the data owner who has a collection of text documents $D = \{D_1, \ldots, D_n\}$. A document D_i is modelled a set of keywords $D_i = \{W_1^{D_i}, \ldots, W_{n_i}^{D_i}\}$, each word $w \in \mathcal{W}$, and (W) is the set of all possible keywords. Alice stores her document collection at a service provider. Since the service provider is not trusted, documents are stored encrypted. Each document is encrypted at the word level as follows: Each document is divided up into equal length "Words". Typically each such word corresponds to an English language word where extra padding (with '0' and '1' bits) are added to make all words equal in length. Periodically Alice may pose a query to the server to retrieve a subset of documents. The query itself is a set of keywords and the answer corresponds to the set of documents that contain all the keywords in the query. More formally, the answer to a query q is given by:

$$Ans(q) = \{D_i \in D | \forall k_j \in q, k_j \in D_i\}$$

The goal is to design techniques to retrieve answers while not revealing any information beyond the presence (or absence) of the keywords (of the query) in each document.

A few different variations of the basic keyword-search problem have been studied over the past years [1, 6, 7, 21, 22, 2, 23]. The authors in [7, 21] study the basic problem where a private-key based encryption scheme is used to design a matching technique on encrypted data that can search for any word in the document. Authors in [1] provide a safe public-key based scheme to carry out "non-interactive" search on data encrypted using user's public-key for a select set of words. [6] proposes a document indexing approach using bloom filters that allows the owner to carry out keyword searches efficiently but could result in some false-positive retrievals. The work in [22, 2] propose secure schemes for conjunctive keyword search where the search term might contain the conjunction of two or more keywords. The goal here again is to avoid any leakage of information over and above the fact that the retrieved set of documents contain all the words specified in the query.

In this section, we describe a private-key based approach which is motivated by [7] and was amongst the first published solutions to the problem of searching over encrypted text data. The approach described incurs significant overhead, requiring $O(n)$ cryptographic operations per document where n is the number of words in the document. We briefly discuss how such overhead can be prevented using Bloom filters. The technique we discuss is a simplification of [6] though it captures the essence of the idea.

2.1 Private-Key based Search Scheme on Encrypted Text Data

Consider a data owner Alice who wishes to store a collection of documents with Bob (the service provider). Alice encrypts each document D prior to storing it with Bob. In addition, Alice creates a secure index, $I(D)$, which is stored at the service provider that will help her perform keyword search. The secure index is such that it reveals no information about its content to the adversary. However, it allows the adversary to test for presence or absence of keywords using a *trapdoor* associated with the keyword where a trapdoor is generated with a secret key that resides with the owner. A user wishing to search for documents containing word w, generates a trapdoor for w which can then be used by the adversary to retrieve relevant documents. We next describe an approach to constructing secure index and the corresponding algorithm to search the index for keywords.

The secure index is created over the keywords in D as follows. Let document D consist of the sequence of words $w_1, \ldots, w_l$. The index is created by computing the bitwise XOR (denoted by the symbol $\oplus$) of the clear-text with a sequence of pseudo-random bits that Alice generates using a stream cipher. Alice first generates a sequence of pseudo-random values $s_1, \ldots, s_l$ using a stream cipher, where each s_i is $n-m$ bit long. For each pseudo-random sequence s_i, Alice computes a pseudo-random function $F_{k_c}(s_i)$ seeded on key k_c which generates a random m-bit sequence[6]. Using the result of $F_k(s_i)$, Alice computes a n-bit sequence $t_i :=< s_i, F_k(s_i) >$, where $< a, b >$ denotes concatenation of the string a and b). Now to encrypt the n-bit word w_i, Alice computes the XOR of w_i with t_i, i.e., ciphertext $c_i := w_i \oplus t_i$. Since, only Alice generates the pseudo-random stream $t_1, \ldots, t_l$ so no one else can decrypt c_i.

Given the above representation of text document, the search mechanism works as follows. When Alice needs to search for files that contain a word w, she transmits w and the key k to the server. The server (Bob) searches for w in the index files associated with documents by checking whether $c_i \oplus w$ is of the form $< s, F_k(s) >$. The server returns to Alice documents that contain the keyword w which can then be decrypted by Alice.

The scheme described above provides secrecy if the pseudo-random function F, the stream cipher used to generate s_i, and the encryption of the document D are secure(that is, the value t_i are indistinguishable from truly random bits for any computationally bounded adversary). Essentially, the adversary cannot learn content of the documents simply based on ciphertext representation.

---

[6] **Pseudo-random functions:** A pseudo-random function denoted as $F : K_F \times X \rightarrow Y$, where K_F is the set of keys, X denotes the set $\{0,1\}^n$ and Y denotes the set $\{0,1\}^m$. Intuitively, a pseudo-random function is computationally indistinguishable from a random function - given pairs $(x_i, f(x_1, k)), \ldots, (x_m, f(x_m, k))$, an adversary cannot predict $f(x_{m+1}, k)$ for any x_{m+1}. In other words, F takes a key $k \in K_F$ the set of keys, a n bit sequence $x \in X$ where X is the set $\{0,1\}^n$ and returns a m bit sequence $y \in Y$ where Y is the set $\{0,1\}^m$.

While the approach described above is secure, it has a fundamental limitation that the adversary learns the keyword w_i that the client searches for. The search strategy allows the adversary to learn which documents contain which keywords over time using such query logs. Furthermore, the adversary can launch attacks by searching for words on his own without explicit authorization by the user thereby learning document content.

A simple strategy to prevent server from knowing the exact search word is to pre-encrypt each word w of the clear text separately using a deterministic encryption algorithm E_{k_p}, where the key k_p is a private key which is kept hidden from the adversary. After this pre-encryption phase, the user has a sequence of E-encrypted words $x_1, \ldots, x_l$. Now he post-encrypts that sequence using the stream cipher construction as before to obtain $c_i := x_i \oplus t_i$, where $x_i = E_{k_p}(w_i)$ and $t_i =< s_i, F_{k_c}(x_i) >$. During search, the client, instead of revealing the keyword to be searched, Computes $E_{k_p}(w_i)$ with the server.

The proposed scheme is secure and ensures that the adversary does not learn document content from query logs. The scheme is formalized below.

- $\mathbf{k_p}$: Denotes the private-key of the user. $k_p \in \{0,1\}^s$ which is kept a secret by the user.
- $\mathbf{k_c}$: Denotes a key called the *collection key* of the user. $k_c \in \{0,1\}^s$ and is publicly known
- **Pseudo-Random Function:** $F : \{0,1\}^s \times \{0,1\}^{n-m} \rightarrow \{0,1\}^m$, is a pseudo-random function that takes a $n-m$ bit string, a s-bit key and maps it to a random m-bit string. F is publicly known.
- **Trapdoor function**: Let T denote a *trapdoor* function which takes as input, a private-key k_p and a word w and outputs the trapdoor for the word w, i.e., $T(k_p, w) = E_{k_p}(w)$ where E is a deterministic encryption function. For a given document, we denote the trapdoor for the i^{th} word by t_i.
- **BuildIndex**$(\mathbf{D}, \mathbf{k_p}, \mathbf{k_c})$: This function is used to build the index for document D. It uses a pseudo-random generator G which outputs random string of size s. The pseudo-code of the function is given below.

Algorithm 1 : *BuildIndex*

```
Input: D, k_p, k_c;
Output: I_D /* The index for the document*/

I_D = φ;
for all w_i ∈ D do
    Generate a pseudo-random string s_i using G;
    Compute trapdoor T(w_i) = E_{k_p}(w_i);
    Compute ciphertext c_i = T(w_i) ⊕ ⟨s_i, F_{k_c}(s_i)⟩;
    I_D = I_D ∪ c_i;
end for
Return I_D;
```

- **SearchIndex($\mathbf{I_D}, \mathbf{T(w)}$)**: Given the document index and the trapdoor for the word w being searched, the *SearchIndex* functionality returns the document D if the word w is present in it. The pseudo-code is given below.

Algorithm 2 : *SearchIndex*

```
1: Input: I_D, T(w);
2: Output: D or φ
3:
4: for all c_i ∈ I_D do
5:    if c_i ⊕ T(w) is of the form ⟨s, F_{k_c}(s)⟩ then
6:       Return D;
7:    end if
8: end for
9: Return φ;
```

2.2 Speeding up search on encrypted data

The approach described above to search over encrypted text has a limitation. Essentially, it requires $O(n)$ comparisons (cryptographic operations) at the server to test if the document contains a given keyword, where n is the number of keywords in the document. While such an overhead might be tolerable for small documents and small document collections, the approach is inherently not scalable. Authors in [6] overcome this limitation by exploiting *bloom filters* for indexing documents. A Bloom filter for a text document is described as follows.

Bloom Filters: A Bloom filter for a document $D = \{w_1, \ldots, w_n\}$ of n words is a m-bit array constructed as follows. All array bits are initially set to 0. The filter uses r independent hash functions $h_1, \ldots, h_r$, where h_i: $\{0,1\}^* \rightarrow [1, m]$ for $i \in [1, r]$. For each word $w \in D$, the array bits at the positions $h_1(w), \ldots, h_r(w)$ are set to 1. A location can be set to 1 multiple times. To determine if a word a belongs is contained in the document D, we check the bits at positions $h_1(a), \ldots, h_r(a)$. If all checked bits are 1's, then a is considered contained in the document D. There is however, some probability of a false positive.

A simple Bloom filter can reveal information about the contents of the document since the hash functions are publicly known. A straightforward strategy to create secure index using Bloom filter is to instead index each word w by its encrypted representation $E_{k_p}(w)$. Thus, the Bloom filter will be constructed using the hash values $h_j(E_{k_p}(w))$, $j = 1, \ldots, r$ instead of applying the hash functions on w directly. This strategy has a vulnerability though, the "footprint" of a word (i.e., the bit-positions in the Bloom filter that are set to '1' corresponding to w) is same for all documents containing the word w. This

makes the scheme vulnerable to frequency-based attacks. One remedy is to use the document-id while encoding the keywords. For instance, one can compute the hash functions for the Bloom filter as follows: $h_j(E_{k_c}(\langle id(D), E_{k_p}(w)\rangle))$, $j = 1 \ldots r$ and set the corresponding bits to 1 in the Bloom filter[7]. This way representation of the same word is different across different documents. As a result, unless a trapdoor is provided, the adversary cannot determine if the same word appears across different documents. The pseudo-code for the $BuildIndex_{BF}$ function is given below.

Algorithm 3 : $BuildIndex_{BF}$

```
Input: D, k_p, k_c, h_1, ..., h_r
Output: BF_D /* The index for the document*/

BF_D = φ;
for all w_i ∈ D do
    Compute trapdoor T(w_i) = E_{k_p}(w_i);
    Compute string x_i = E_{k_c}(⟨id(D), T(w_i)⟩)
    for j = 1 to r do
        compute bit-position b_j = h_j(x_i);
        set BF_D[b_j] = 1;
    end for
end for
Return BF_D;
```

In the current scheme, the search needs to be performed in a slightly different manner. If the user wants to search for a word w, he gives the trapdoor $T(w) = E_{k_p}(w)$ to the server. The server executes the function $SearchIndex_{BF}$ (given below) on each document D in the collection and returns the appropriate ones.

Algorithm 4 : $SearchIndex_{BF}$

```
Input: BF_D, T(w), k_c, h_1, ..., h_r
Output: D or φ

Compute x = E_{k_c}(⟨id(D), T(w)⟩);
for j=1 to r do
    if BF_D[h_j(x)] ≠ 1 then
        Return φ;
    end if
end for
Return D;
```

[7] The extra level of encryption with k_c is not strictly required if the hash functions h_i's are appropriately chosen to be one-way functions with collision resistance

Above, we sketched an approach on how Bloom filters can be used to do secure indices. The technique of using two levels of security (document-id based encryption) to prevent frequency based attacks is similar to what is proposed in [6]. The author in [6] develops a complete strategy for constructing secure indices using Bloom filters and presents a detailed security analysis.

2.3 Secure Keyword Search using Public-Key Encryption

We now consider a variation to the basic encrypted text search problem where the producer (owner) of the data and the data consumer (client) are different. To motivate the problem, consider an (untrusted) email gateway that stores incoming emails from multiple users. If emails are sensitive they will need to be encrypted. So if Bob needs to send a sensitive email to Alice, he will have to encrypt it using Alices public key. Now Alice may wish to have the capability to search for such emails using keywords. Alice (or Alice's mail client) could, of course, download such email, decrypt it, create a secure index using a secret key (as in the previous section) and store the index along with the original encrypted email at the gateway. Such a secure index, if integrated appropriately with the email server could provide Alice with the requisite functionality. A more natural approach would be to instead exploit a public-key encryption technique that directly supports keyword search over encrypted representation. Such a public-key system is developed in [1] in the limited context where Alice pre-specifies the set of keywords she might be interested in searching the mail based on. Using the scheme developed in [1], the mail sender (Bob) can send an email to Alice encrypted using her public key. Alice can give the gateway a limited capability to detect some keywords in her emails (encrypted using her public key) and have these mails routed in a different manner, e.g. an email with keyword "lunch" should be routed to her desktop and one with "urgent" should be routed to her pager etc. The scheme prevents the gateway from learning anything beyond the fact that a certain keyword (for which it has the "capability" to test) is present in the set of keywords associated with the mail.

The approach works by requiring the sender of the mail, Bob, to append to the ciphertext (email encrypted using Alices public key) additional codewords referred to as *Public-key Encryption with Keyword Search* (PEKS), one for each keyword. To send a message M with keywords $W_1, \ldots, W_m$ Bob sends

$$E_{A_{pub}}(M)||PEKS(A_{pub}, W_1)|| \ldots ||PEKS(A_{pub}, W_m)$$

where A_{pub} is Alice's public key. This allows Alice to give the gateway a certain trapdoor T_W that enables the gateway to test whether one of the keywords associated with the message is equal to the word W of Alice's choice. Given $PEKS(A_{pub}, W')$ and T_W the gateway can test whether $W = W'$. If $W \neq W'$ the gateway learns nothing more about W'.

We next describe the main construction of the approach in [1] which is based on using bilinear maps.

Bilinear maps: Let G_1 and G_2 be two groups of order p for some large prime p. A bilinear map $e : G_1 \times G_1 \rightarrow G_2$ satisfies the following properties:

1. *Computable:* given $g, h \in G_1$ there is a polynomial time algorithm to compute $e(g, h) \in G_2$.
2. *Bilinear:* We say that a map $e : G_1 \times G_1 \rightarrow G_2$ is bilinear if for any integers $s, y \in [1, p]$ we have $e(g^x, g^y) = e(g, g)^{xy}$.
3. *Non-degenerate:* The map does not send all pairs of $G_1 \times G_1$ to the identity in G_2. Since G_1, G_2 are groups of prime order this implies that if g is a generator of G_1 then $e(g, g)$ is a generator of G_2.

[1] builds a searchable encryption scheme using bilinear maps as described below.

- $KeyGen$: The input security parameter, s, determines the size, p of the groups G_1 and G_2. The algorithm picks a random $\alpha \in \mathbb{Z}_p^*$ and a generator g of G_1. It outputs the public/private key pair $A_{pub} = [g, h = g^\alpha]$ and $A_{priv} = \alpha$.
- $PEKS(A_{pub}, W)$: for a public key A_{pub} and a word W, produce a searchable encryption of W. First compute $t = e(H_1(W), h^r) \in G_2$ for a random $r \in \mathbb{Z}_p^*$. Output $PEKS(A_{pub}, W) = [g^r, H_2(t)]$, where $H_1 : \{0,1\}^* \rightarrow G_1$ and $H_2 : G_2 \rightarrow \{0,1\}^{log\ p}$.
- $Trapdoor(A_{priv}, W)$: given a private key and a word W, produce a trapdoor T_W as $T_W = H_1(W)^\alpha \in G_1$.
- $Test(A_{pub}, S, T_W)$: given Alice's public key, a searchable encryption $S = PEKS(A_{pub}, W')$, and a trapdoor $T_W = Trapdoor(A_{priv}, W)$, outputs 'yes' if $W = W'$ and 'no' otherwise. The test is performed as follows: let $S = [C, D]$. Check if $H_2(e(T_W, C)) = D$. If so, output 'yes'; if not, output 'no'.

To illustrate how the $PEKS$ based matching takes place, we can see that for a keyword W in an email sent by Bob to Alice, Bob would create a corresponding codeword $PEKS(A_{pub}, W)$ as follows and attach it to his mail.

$$PEKS(A_{pub}, W) = [g^r, H_2(e(H_1(W), h^r))] = S = [C, D]$$

Now, if Alice wanted to search for the same word W in mails, she would produce the trapdoor $T_W = H_1(W)^\alpha \in G_1$ and give it to the mail server to do encrypted matching. Now we have $C = g^r$ and $D = H_2(e(H_1(W), h^r))$. Using the properties of bilinear maps, and the fact that $h = g^\alpha$, we have $D = H_2(e(H_1(W), g)^{r\alpha})$. In the $Test$ function, the server would compute the following:

$$H_2(e(T_W, C)) = H_2(e(H_1(W)^\alpha, g^r)) = H_2(e(H_1(W), g)^{r\alpha}) = D$$

which results in a match if the the trapdoor and $PEKS$ both correspond to the same word W.

The above scheme allows the email gateway to determine if emails encrypted using Alices public key contain one of the keywords of interest to Alice without revealing to the gateway any information about the word W unless T_W is available. The scheme provides security against an active attacker who is able to distinguish an encryption of a keyword W_0 from an encryption of a keyword W_1 for which he did not obtain the trapdoor (referred to as *adaptive chosen keyword attack*).

2.4 Other Research

Another variation to the basic keyword search on encrypted data that has recently been studied, is that of "conjunctive keyword search" [22, 2]. Most keyword searches contain more than one keyword in general. The straight forward way to support such queries is to carry out the search using single keywords and then return the intersection of the retrieved documents as the result set. The authors in [22, 2] claim that such a methodology reveals more information than there is a need for and might make the documents more vulnerable to statistical attacks. They develop special cryptographic protocols that return a document if and only if all the search words are present in it.

There are some shortcomings of all the above cryptographic methods described for keyword search on encrypted data. Once a capability is given to the untrusted server (or once a search for a certain word has been carried out), that capability can be continued to be used forever by the server to check if these words are present in newly arriving (generated) mails (documents) even though the owner might not want to give the server this capability. This can, in turn, make the schemes vulnerable to a variety of statistical attacks.

3 Search over Encrypted Relational Data

In this section, we describe techniques developed in the literature to support queries over encrypted relational data. As in the previous section, we begin by first setting the problem. Consider a user Alice who outsources the database consisting of the following two relations:

```
EMP (eid, ename, salary, addr, did)
DEPARTMENT (did, dname, mgr)
```

The fields in the *EMP* table refer to the employee id, name of the employee, salary, address and the id of the department the employee works for. The fields in the *DEPARTMENT* table correspond to the department id, department name, and name of the manager of the department. In the DAS model, the above tables will be stored at the service provider. Since the service provider is untrusted, the relations must be stored in an encrypted form. Relational data could be encrypted at different granularity – e.g., at the table level, the row level, or the attribute level. As will become clear, the choice of granularity of encryption has significant repercussions on the scheme used to support search and on the system performance. Unless specified otherwise, we will assume that data is encrypted at the row level; that is, each row of each table is encrypted as a single unit. Thus, an encrypted relational representation consists of a set of encrypted records.

The client[8] may wish to execute SQL queries over the database. For instance, Alice may wish to pose following query to evaluate "total salary for employees who work for Bob". Such a query is expressed in SQL as follows:

```
SELECT  SUM(E.salary) FROM EMP as E, DEPARTMENT as D
WHERE E.did = D.did AND D.mgr = "Bob"
```

An approach Alice could use to evaluate such a query might be to request the server for the encrypted form of the *EMP* and *DEPARTMENT* tables. The client could then decrypt the tables and execute the query. Such an approach, however, would defeat the purpose of database outsourcing, reducing it to essentially a remote secure storage. Instead, the goal in DAS is to process the queries directly at the server without the need to decrypt the data. Before we discuss techniques proposed in the literature to process relational queries over encrypted data, we note that processing such queries requires mechanisms to support the following basic operators over encrypted data:

- **Comparison operators** such as $=, \neq, <, \leq, =, \geq, >$ These operators may compare attribute values of a given record with constants (e.g., $DEPARTMENT.sal > 45000$ as in selection queries) or with other attributes (e.g., $EMP.did = DEPARTMENT.did$ as in join conditions).

[8] Alice in this case since we have assumed that the client and the owner is the same entity.

- **Arithmetic operators** such as addition, multiplication, division that perform simple arithmetic operations on attribute values associated with a set of records in one or more relations. Such operators are part of any SQL query that involves aggregation.

The example query given above illustrates usage of both classes of operators. For instance, to execute the query, the *mgr* field of each record in the *DEPARTMENT* table has to be compared with "Bob". Furthermore, records in the *DEPARTMENT* table whose *mgr* is "Bob" have to be matched with records in *EMP* table based on the *did* attribute. Finally, the *salary* fields of the corresponding record that match the query conditions have to be added to result in the final answer.

The first challenge in supporting SQL queries over encrypted relational representation is to develop mechanisms to support comparison and arithmetic operations on encrypted data. The techniques developed in the literature can be classified into the following two categories.

Approaches based on new encryption techniques: that can support either arithmetic and/or comparison operators directly on encrypted representation. Encryption techniques that support limited computation without decryption have been explored in cryptographic literature in the past. Amongst the first such technique is the *privacy homomorphism* (PH) developed in [24, 32] that supports basic arithmetic operations. While PH can be exploited to support aggregation queries at the remote server (see [25] for details), it does not Support comparison and, as such, cannot be used as basis for designing techniques for relational query processing over encrypted data. In [19], the authors developed a data transformation technique that preserves the order in the original data. Such a transformation serves as an *order-preserving* encryption and can hence support comparison operators. Techniques to implement relational operators such as selection, joins, sorting, grouping can be built on top of the order preserving encryption. The encryption mechanism, however, cannot support aggregation at the server. While new cryptographic approaches are interesting, one of the limitation of such approaches has been that they safe only under limited situations where the adversarys knowledge is limited to the ciphertext representation of data. These techniques have either been shown to break under more general attacks (e.g., PH is not secure under chosen plaintext attack [33, 34]), or the security analysis under diverse types of attacks has not been performed.

Information-hiding based Approaches: Unlike encryption-based approaches, such techniques store additional auxiliary information along with encrypted data to facilitate evaluation of comparison and/or arithmetic operations at the server. Such auxiliary information, stored in the form of indices (which we refer to as *secure indices*) may reveal partial information about the data to the server. Secure indices are designed carefully exploiting information hiding mechanisms (developed in the context of statistical disclosure control)

[29, 30, 31] to limit the amount of information disclosure.The basic techniques used for disclosure control are the following [30, 31]:

1. **Perturbation:** For a numeric attribute of a record, add a random value (chosen from some distribution, like normal with mean 0 and standard deviation σ) to the true value.
2. **Generalization:** Replace a numeric or categorical value by a more general value. For numeric values, it could be a range of that covers the original value and for categorical data, this may be a more generic class, e.g., an ancestor node in a taxonomy tree.
3. **Swapping:** Take two different records in the data set and swap the values of a specific attribute (say, the salary value is swapped between the records corresponding to two individuals).

Of all the disclosure-control methods, the one that has been primarily utilized to realize DAS functionalities is that of generalization. Though this is not to say that DAS functionalities cannot be built using other techniques, e.g., hiding data values by noise-addition and developing techniques for querying on the perturbed data. However, we are not aware of any complete proposal based on such a mechanism.

The nature of disclosure in information hiding based schemes is different from that in cryptographic schemes. In the latter, the disclosure risk is inversely proportional to the difficulty of breaking the encryption scheme and if broken, it means there is complete disclosure of the plaintext values. In contrast, the information disclosure in information hiding approaches could be partial or probabilistic in nature. That is, there could be a non-negligible probability of disclosure of a sensitive value given the transformed data, e.g., the bucket identity might give a clue regarding the actual value of the sensitive attribute.

In this section, we will primarily concentrate on the information hiding based approach and show how it has been utilized to support SQL queries. As will be clear, information hiding approaches can be used to support comparison operators on the server and can hence be the basis for implementing SPJ (select-project-join) queries. They can also support sorting and grouping operators. Such techniques, however, cannot support aggregation at the server. A few papers [25, 39] have combined an information hiding approach with PH to support both server-side aggregation as well as SPJ queries. Of course, with PH being used for aggregation, these techniques become vulnerable to diverse types of attacks. In the remainder of the section, we will concentrate on how information hiding techniques are used to support SPJ queries. We will use the query processing architecture proposed in [12, 26] to explain the approach.

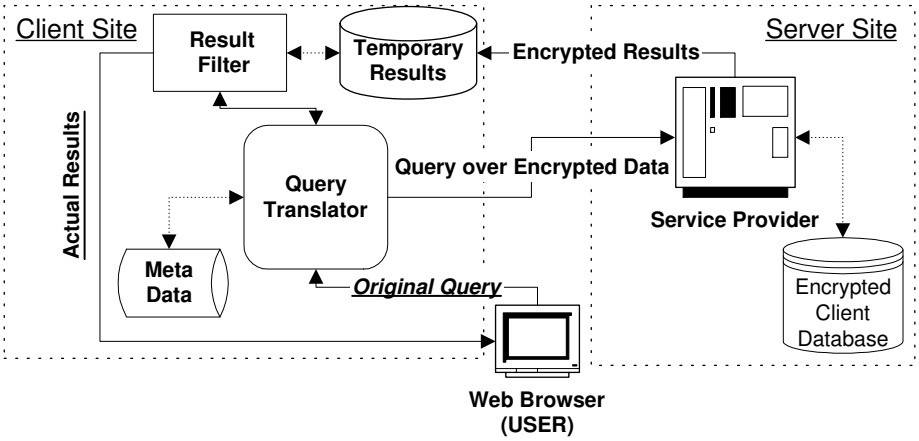

Fig. 1. Query Processing in DAS

Query Processing Architecture for DAS [26]

Figure 1 illustrates the control flow for queries in DAS where information hiding technique is used to represent data at the server. The figure illustrates the three primary entities of the DAS model: *user*, *client* and *server*. As stated earlier, we will not distinguish between the user and the client and will refer to them together as the *client-side*. The client stores the data at the server which is hosted by the service provider and this is known as the *server-side*. The data is stored in an encrypted format at the server-side at all times for security purposes. The encrypted database is augmented with additional information (which we call the secure index) that allows certain amount of query processing to occur at the server without jeopardizing data privacy. The client also maintains *metadata* for translating user queries to the appropriate representation on the server, and performs post-processing on server-query results. Based on the auxiliary information stored, the original query over un-encrypted relations are broken into (1) a server-query over encrypted relations which run on the server, and (2) a client-query which runs on the client and post-processes the results returned after executing the server-query. We achieve this goal by developing an algebraic framework for query rewriting over encrypted representation.

3.1 Relational Encryption and Storage Model

For each relation

$$R(A_1, A_2, \ldots, A_n)$$

one stores on the server an encrypted relation:

$$R^S(etuple, A_1^S, A_2^S, \ldots, A_n^S)$$

where the attribute *etuple* (*etuple* is defined shortly) stores an encrypted string that corresponds to a tuple in relation R[9]. Each attribute A_i^S corresponds to the index for the attribute A_i and is used for query processing at the server. For example, consider a relation *emp* below that stores information about employees.

eid	ename	salary	addr	did
23	Tom	70K	Maple	40
860	Mary	60K	Main	80
320	John	50K	River	50
875	Jerry	55K	Hopewell	110

The *emp* table is mapped to a corresponding table at the server:

$$emp^S(etuple, eid^S, ename^S, salary^S, addr^S, did^S)$$

It is only necessary to create an index for attributes involved in search and join predicates. In the above example, if one knows that there would be no query that involves attribute *addr* in either a selection or a join, then the index on this attribute need not be created. Without loss of generality, one can assume that an index is created over each attribute of the relation.

Partition Functions: To explain what is stored in attribute A_i^S of R^S for each attribute A_i of R the following notations are useful. The domain of values ($\mathcal{D}_i$) of attribute $R.A_i$ are first mapped into partitions $\{p_1, \ldots, p_k\}$, such that (1) these partitions taken together cover the whole domain; and (2) any two partitions do not overlap. The function *partition* is defined as follows:

$$partition(R.A_i) = \{p_1, p_2, \ldots, p_k\}$$

As an example, consider the attribute *eid* of the *emp* table above. Suppose the values of domain of this attribute lie in the range $[0, 1000]$. Assume that the whole range is divided into 5 partitions
: $[0, 200]$, $(200, 400]$, $(400, 600]$, $(600, 800]$, and $(800, 1000]$. That is:

$$partition(emp.eid) = \{[0, 200], (200, 400], (400, 600], (600, 800], (800, 1000]\}$$

Different attributes may be partitioned using different partition functions. The partition of attribute A_i corresponds to a splitting of its domain into a set of buckets. The strategy used to split the domain into a set of buckets has profound implications on both the efficiency of the resulting query processing as well as on the disclosure risk of sensitive information to the server. For now, to explain the query processing strategy, we will make a simplifying

[9] Note that one could alternatively choose to encrypt at the attribute level instead of the row-level. Each alternative has its own pros and cons and for greater detail, the interested reader is referred to [10].

assumption that the bucketization of the domain is based on the equi-width[10] partitioning (though the strategy developed will work for any partitioning of domain). We will revisit the efficiency and disclosure risks in the following subsections.

In the above example, an equi-width histogram was illustrated. Note that when the domain of an attribute corresponds to a field over which ordering is well defined (e.g., the *eid* attribute), we will assume that a partition p_i is a continuous range. We use $p_i.low$ and $p_i.high$ to denote the lower and upper boundary of the partition, respectively.

Identification Functions: An identification function called *ident* assigns an identifier $ident_{R.A_i}(p_j)$ to each partition p_j of attribute A_i. Figure 2 shows the identifiers assigned to the 5 partitions of the attribute *emp.eid*. For instance, $ident_{emp.eid}([0, 200]) = 2$, and $ident_{emp.eid}((800, 1000]) = 4$.

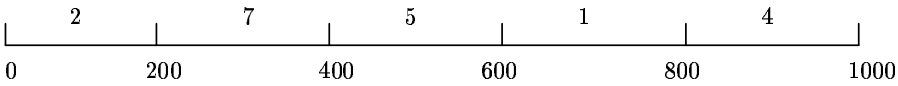

Fig. 2. Partition and identification functions of *emp.eid*

The *ident* function value for a partition is unique, that is, $ident_{R.A_i}(p_j) \neq ident_{R.A_i}(p_l)$, if $j \neq l$. For this purpose, a collision-free hash function that utilizes properties of the partition may be used as an *ident* function. For example, in the case where a partition corresponds to a numeric range, the hash function may use the start and/or end values of a range.

Mapping Functions: Given the above partition and identification functions, a mapping function $Map_{R.A_i}$ maps a value v in the domain of attribute A_i to the identifier of the partition to which v belongs: $Map_{R.A_i}(v) = ident_{R.A_i}(p_j)$, where p_j is the partition that contains v.

For the example given above, the following table shows some values of the mapping function for attribute *emp.eid*. For instance, $Map_{emp.eid}(23) = 2$, $Map_{emp.eid}(860) = 4$, and $Map_{emp.eid}(875) = 4$.

eid value v	23	860	320	875
$Map_{emp.eid}(v)$	2	4	7	4

Three generic mapping functions are illustrated below. Let S be a subset of values in the domain of attribute A_i, and v be a value in the domain. We define the following mapping functions on the partitions associated with A_i:

$$Map_{R.A_i}(S) = \{ident_{R.A_i}(p_j) | p_j \cap S \neq \emptyset\}$$

$$Map^{\geqslant}_{R.A_i}(v) = \{ident_{R.A_i}(p_j) | p_j.high \geqslant v\}$$

[10] where the domain of each bucket has the same width

$$Map^{<}_{R.A_i}(v) = \{ident_{R.A_i}(p_j) | p_j.low \leqslant v\}$$

While the first function defined holds over any attribute, the latter two hold for the attributes whose domain values exhibit total order. Application of the mapping function to a value v, greater than the maximum value in the domain, v_{max}, returns $Map_{R.A_i}(v_{max})$. Similarly, application of the mapping function to a value v, less than the minimum value in the domain, v_{mim}, returns $Map_{R.A_i}(v_{min})$. Essentially, $Map_{R.A_i}(S)$ is the set of identifiers of partitions whose ranges may overlap with the values in S. The result of $Map^{>}_{R.A_i}(v)$ is the set of identifiers corresponding to partitions whose ranges may contain a value not less than v. Likewise, $Map^{<}_{R.A_i}(v)$ is the set of identifiers corresponding to partitions whose ranges may contain a value not greater than v.

Storing Encrypted Data: For each tuple $t = \langle a_1, a_2, \ldots, a_n \rangle$ in R, the relation R^S stores a tuple:

$$\langle encrypt(\{a_1, a_2, \ldots, a_n\}), Map_{R.A_1}(a_1), Map_{R.A_2}(a_2), \ldots, Map_{R.A_n}(a_n) \rangle$$

where *encrypt* is the function used to encrypt a tuple of the relation. For instance, the following is the encrypted relation emp^S stored on the server:

etuple	eid^S	$ename^S$	$salary^S$	$addr^S$	did^S
1100110011110010...	2	19	81	18	2
1000000000011101...	4	31	59	41	4
1111101000010001...	7	7	7	22	2
1010101010111110...	4	71	49	22	4

The first column *etuple* contains the string corresponding to the encrypted tuples in *emp*. For instance, the first tuple is encrypted to "1100110011110010..." that is equal to $encrypt(23, Tom, 70K, Maple, 40)$. The second is encrypted to "1000000000011101..." equal to $encrypt(860, Mary, 60K, Main, 80)$. The encryption function is treated as a black box and any block cipher technique such as AES, Blowfish, DES etc., can be used to encrypt the tuples. The second column corresponds to the index on the employee ids. For example, value for attribute *eid* in the first tuple is 23, and its corresponding partition is $[0, 200]$. Since this partition is identified to 2, we store the value "2" as the identifier of the *eid* for this tuple. Similarly, we store the identifier "4" for the second employee id 860. In the table above, we use different mapping functions for different attributes. The mapping functions for the *ename*, *salary*, *addr*, and *did* attributes are not shown, but they are assumed to generate the identifiers listed in the table.

In general the notation "E" ("Encrypt") is used to map a relation R to its encrypted representation. That is, given relation $R(A_1, A_2, \ldots, A_n)$, relation $E(R)$ is $R^S(etuple, A^S_1, A^S_2, \ldots, A^S_n)$. In the above example, $E(emp)$ is the table emp^S.

Decryption Functions: Given the operator E that maps a relation to its encrypted representation, its inverse operator D maps the encrypted representation to its corresponding decrypted representation. That is, $D(R^S) = R$. In the example above, $D(emp^S) = emp$. The D operator may also be applied on query expressions. A query expression consists of multiple tables related by arbitrary relational operators (e.g., joins, selections, etc).

As it will be clear later, the general schema of an encrypted relation or the result of relational operators amongst encrypted relations, R_i^S is:

$$\langle R_1^S.etuple, R_2^S.etuple, \ldots, R_1^S.A_1^S, R_1^S.A_2^S, \ldots, R_2^S.A_1^S, R_2^S.A_2^S, \ldots \rangle$$

When the decryption operator D is applied to R_i^S, it strips off the index values ($R_1^S.A_1^S$, $R_1^S.A_2^S, \ldots, R_2^S.A_1^S, R_2^S.A_2^S, \ldots$) and decrypts ($R_1^S.etuple$, $R_2^S.etuple, \ldots$) to their un-encrypted attribute values.

As an example, assume that another table defined as mgr (mid, did) was also stored in the database. The corresponding encrypted representation $E(mgr)$ will be a table mgr^S ($etuple$, mid^S, did^S). Suppose we were to compute a join between tables emp^S and mgr^S on their did^S attributes. The resulting relation $temp^S$ will contain the attributes $\langle emp^S.etuple$, eid^S, $ename^S$, $salary^S$, $addr^S$, $emp^S.$ did^S, $mgr^S.etuple$, mid^S, $mgr^S.did^S \rangle$. If we were to decrypt the $temp^S$ relation using the D operator to compute $D(temp^S)$, the corresponding table will contain the attributes

$$(eid, ename, salary, addr, emp.did, mid, mgr.did)$$

That is, $D(temp^S)$ will decrypt *all* of the encrypted columns in $temp^S$ and drop the auxiliary columns corresponding to the indices.

Mapping Conditions

To translate specific query conditions in operations (such as selections and joins) to corresponding conditions over the server-side representation, a translation function called Map_{cond} is used. These conditions help translate relational operators for server-side implementation, and how query trees are translated.

For each relation, the server-side stores the encrypted tuples, along with the attribute indices determined by their mapping functions. The client stores the meta data about the specific indices, such as the information about the partitioning of attributes, the mapping functions, etc. The client utilizes this information to translate a given query Q to its server-side representation Q^S, which is then executed by the server. The query conditions are characterized by the following grammar rules:

- Condition ← Attribute *op* Value;
- Condition ← Attribute *op* Attribute;
- Condition ← (Condition ∨ Condition) | (Condition ∧ Condition);.

Allowed operations for *op* include $\{=, <, >, \leqslant, \geqslant\}$. Now consider the following tables to illustrate the translation.

```
emp(eid, ename, salary, addr, did, pid)
mgr(mid, did, mname)
proj(pid, pname, did, budget)
```

Attribute = Value: Such a condition arises in selection operations. The mapping is defined as follows:

$$Map_{cond}(A_i = v) \equiv A_i^S = Map_{A_i}(v)$$

As defined above, function Map_{A_i} maps v to the identifier of A_i's partition that contains the value v. For instance, consider the *emp* table above, we have:

$$Map_{cond}(eid = 860) \equiv eid^S = 4$$

since $eid = 860$ is mapped to 4 by the mapping function of this attribute.

Attribute < Value: Such a condition arises in selection operations. The attribute must have a well-defined ordering over which the "<" operator is defined. The translation is a little complex. One needs to check if the attribute value representation A_i^S lies in any of the partitions that may contain a value v' where $v' < v$. Formally, the translation is:

$$Map_{cond}(A_i < v) \equiv A_i^S \in Map_{A_i}^{<}(v)$$

For instance, the following condition is translated:

$$Map_{cond}(eid < 280) \equiv eid^S \in \{2, 7\}$$

since all employee ids less than 280 have two partitions $[0, 200]$ and $(200, 400]$, whose identifiers are $\{2, 7\}$.

Attribute > Value: This condition is symmetric with the previous one. The translation is as follows:

$$Map_{cond}(A_i > v) \equiv A_i^S \in Map_{A_i}^{>}(v)$$

.

For instance, the following condition is translated:

$$Map_{cond}(eid > 650) \equiv eid^S \in \{1, 4\}$$

since all employee ids greater than 650 are mapped to identifiers: $\{1, 4\}$.

Attribute1 = Attribute2: Such a condition might arise in a join. The two attributes can be from two different tables, or from two instances of the same table. The condition can also arise in a selection, and the two attributes can be from the same table. The following is the translation:

$$Map_{cond}(A_i = A_j) \equiv \bigvee_{\varphi} \left(A_i^S = ident_{A_i}(p_k) \wedge A_j^S = ident_{A_j}(p_l)\right)$$

where φ is $p_k \in partition(A_i), p_l \in partition(A_j), p_k \cap p_l \neq \emptyset$. That is, one needs to consider all possible pairs of partitions of A_i and A_j that overlap. For each pair (p_k, p_l), one needs a condition on the identifiers of the two partitions: $A_i^S = ident_{A_i}(p_k) \wedge A_j^S = ident_{A_j}(p_l)$. Finally the disjunction of these conditions need to be taken. The intuition is that each pair of partitions may provide some values of A_i and A_j that can satisfy the condition $A_i = A_j$.

For instance, the table below shows the partition and identification functions of two attributes *emp.did* and *mgr.did*.

Partitions	$Ident_{emp.did}$	Partitions	$Ident_{mgr.did}$
[0,100]	2	[0,200]	9
(100,200]	4	(200,400]	8
(200,300]	3		
(300,400]	1		

Then condition $emp.did = mgr.did$ is translated to the following condition C_1:

$$\begin{aligned} C_1: \quad & (emp^S.did^S = 2 \wedge mgr^S.did^S = 9) \\ \vee\ & (emp^S.did^S = 4 \wedge mgr^S.did^S = 9) \\ \vee\ & (emp^S.did^S = 3 \wedge mgr^S.did^S = 8) \\ \vee\ & (emp^S.did^S = 1 \wedge mgr^S.did^S = 8). \end{aligned}$$

Attribute1 < Attribute2: Again such a condition might arise in either a join or in a selection. Let the condition be $A_i < A_j$, then the translation is the following:

$$Map_{cond}(A_i < A_j) \equiv \bigvee_{\varphi} \left(A_i^S = ident_{A_i}(p_k) \wedge A_j^S = ident_{A_j}(p_l)\right)$$

where φ is $p_k \in partition(A_i), p_l \in partition(A_j), p_l.high \geqslant p_k.low$. One needs to consider all pairs of partitions of A_i and A_j that could satisfy the condition. For each pair, there is a condition corresponding to the pair of their identifiers and one needs to take the disjunction of all these conditions.

For example, condition $C_2 : emp.did < mgr.did$ is translated to:

$$\begin{aligned} C_2: \quad & (emp^S.did^S = 2 \wedge mgr^S.did^S = 9) \\ \vee\ & (emp^S.did^S = 2 \wedge mgr^S.did^S = 8) \\ \vee\ & (emp^S.did^S = 4 \wedge mgr^S.did^S = 9) \\ \vee\ & (emp^S.did^S = 4 \wedge mgr^S.did^S = 8) \\ \vee\ & (emp^S.did^S = 3 \wedge mgr^S.did^S = 8) \\ \vee\ & (emp^S.did^S = 1 \wedge mgr^S.did^S = 8). \end{aligned}$$

Condition $emp^S.did^S = 4 \wedge mgr^S.did^S = 9$ is included, since partition $(100, 200]$ for attribute *emp.did* and partition $(200, 400]$ for attribute *mgr.did* can provide pairs of values that satisfy $emp.did < mgr.did$.

For condition $Attribute1 > Attribute2$, the Map_{cond} mapping is same as the mapping of $Attribute2 < Attribute1$, as described above with the roles of the attributes reversed.

Condition1 $\vee$ Condition2, Condition1 $\wedge$ Condition2: The translation of the two composite conditions is given as follows:

$Map_{cond}(Condition1 \vee Condition2) \equiv$
$Map_{cond}(Condition1) \vee Map_{cond}(Condition2)$
$Map_{cond}(Condition1 \wedge Condition2) \equiv$
$Map_{cond}(Condition1) \wedge Map_{cond}(Condition2)$

Operator $\leqslant$ follows the same mapping as $<$ and operator $\geqslant$ follows the same mapping as $>$.

Translating Relational Operators

In this section we describe how relational operators are implemented in [26]. We illustrate the implementation of the selection and join operators in the proposed architecture. The strategy is to partition the computation of the operators across the client and the server such that a superset of answers is generated by the operator using the attribute indices stored at the server. This set is then filtered at the client after decryption to generate the true results. The goal is to minimize the work done at the client as much as possible. We use R and T to denote two relations, and use the operator notations in [8].

The Selection Operator (σ): Consider a selection operation $\sigma_C(R)$ on a relation R, where C is a condition specified on one or more of the attributes $A_1, A_2, \ldots, A_n$ of R. A straightforward implementation of such an operator is to transmit the relation R^S from the server to the client. Then the client decrypts the result using the D operator, and implements the selection. This strategy, however, pushes the entire work of implementing the selection to the client. In addition, the entire encrypted relation needs to be transmitted from the server to the client. An alternative mechanism is to partially compute the selection operator at the server using the indices associated with the attributes in C, and push the results to the client. The client decrypts the results and filters out tuples that do not satisfy C. Specifically, the operator can be rewritten as follows:

$$\sigma_C(R) = \sigma_C\Big(D(\sigma^S_{Map_{cond}(C)}(R^S))\Big)$$

Note that the σ operator that executes at the server with a superscript "S" to highlight the fact that the select operator executes at the server. All

non-adorned operators execute at the client. The decryption operator D will only keep the attribute *etuple* of R^S, and drop all the other A_i^S attributes. We explain the above implementation using an example $\sigma_{eid<395 \wedge did=140}(emp)$. Based on the definition of $Map_{cond}(C)$ discussed in the previous section, the above selection operation will be translated into

$$\sigma_C\Big(D(\sigma^S_{C'}(emp^S))\Big)$$

where the condition C' on the server is:

$$C' = Map_{cond}(C) = \big(eid^S \in [2,7] \wedge did^S = 4\big)$$

The Join Operator ($\bowtie$): Consider a join operation $R \overset{\bowtie}{_C} S$. The join condition C could be either an equality condition (in which case the join corresponds to an equijoin), or could be a more general condition (resulting in theta-joins). The above join operation can be implemented as follows:

$$R \overset{\bowtie}{_C} T = \sigma_C\Big(D\big(R^S \overset{\bowtie^S}{_{Map_{cond}(C)}} T^S\big)\Big)$$

As before, the S adornment on the join operator denotes the fact that the join is to be executed at the server. For instance, join operation

$$emp \overset{\bowtie}{_{emp.did=mgr.did}} mgr$$

is translated to:

$$\sigma_C\Big(D\big(emp^S \overset{\bowtie^S}{_{C'}} mgr^S\big)\Big)$$

where the condition C' on the server is condition C_1 defined in Section 3.1.

Now we show how the above operators are used to rewrite SQL queries for the purpose of splitting the query computation across the client and the server.

Query Execution

Given a query Q, the goal is to split the computation of Q across the server and the client. The server will use the implementation of the relational operators discussed in the previous subsection to compute "as much of the query as possible", relegating the remainder of the computation to the client. Query processing and optimization have been extensively studied in database research [9, 5, 20]. The objective is to come up with the "best" query plan for Q that minimizes the amount of work to be done at the client site. In our setting, the cost of a query consists of many components – the I/O and CPU cost of evaluating the query at the server, the network transmission cost, and the I/O and CPU cost at the client. As an example, consider the following query over the *emp* table than retrieves employees whose salary is greater that the average salary of employees in the department identified by $did = 1$.

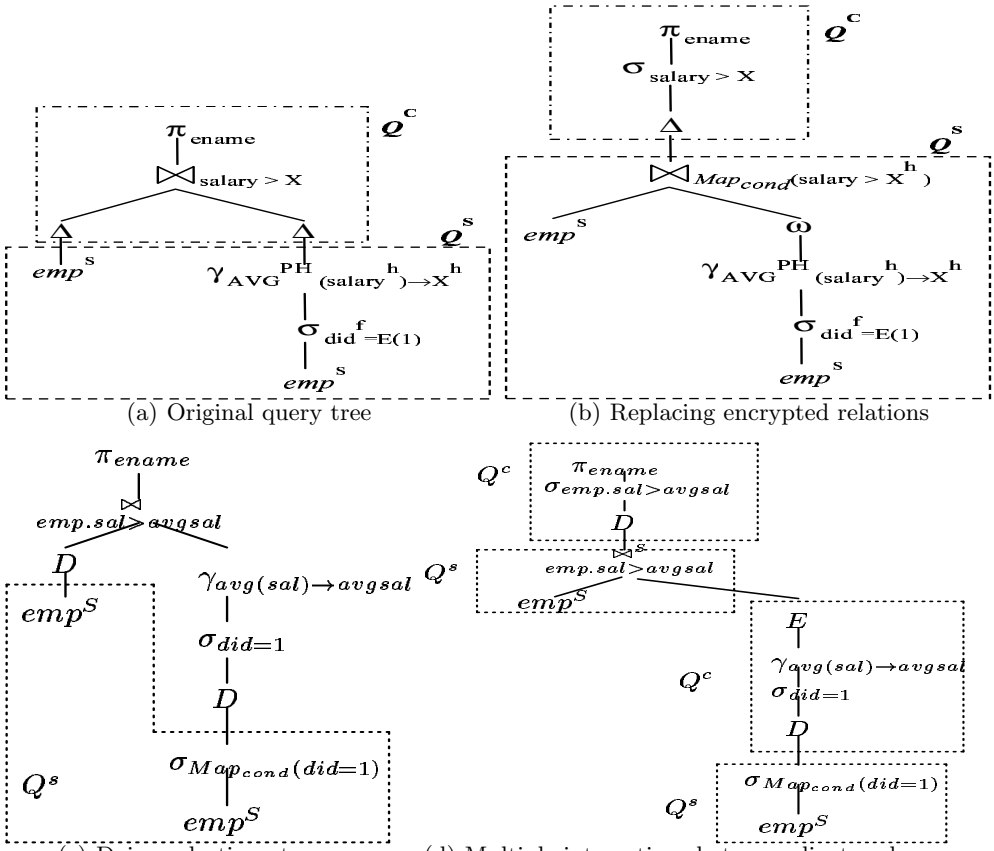

Fig. 3. Query plans for employees who make more than average salary of employees who are in did=1

```
SELECT emp.name FROM emp
WHERE  emp.salary > (SELECT AVG(salary)
FROM emp WHERE did = 1);
```

The corresponding query tree and some of the evaluation strategies are illustrated in Figures 3(a) to (d). The first strategy (Figure 3(b)) is to simply transmit the *emp* table to the client, which evaluates the query. An alternative strategy (Figure 3(c)) is to compute part of the inner query at the server, which selects (as many as possible) tuples corresponding to $Map_{cond}(did = 1)$. The server sends to the client the encrypted version of the *emp* table, i.e., emp^S, along with the encrypted representation of the set of tuples that satisfy the inner query. The client decrypts the tuples to evaluate the remainder of the query. Yet another possibility (Figure 3(d)) is to evaluate the inner query at the server. That is, select the tuples corresponding to the employees that work in department $did = 1$. The results are shipped to the client,

which decrypts the tuples and computes average salary. The average salary is encrypted by the client and shipped back to the server, which then computes the join at the server. Finally, the results are decrypted at the client.

Supporting Aggregation Operators in Queries: The query translation techniques discussed above are designed explicitly for relational operators that perform comparisons. While information hiding techniques work for relational operators, they do not work for arithmetic operators such as aggregation. Notice that in the previous query there is an aggregation but that aggregation is done at the client side after decryption. If aggregation is to be performed at the server side, the information hiding approach has to be augmented with an encryption approach that supports arithmetic operations on encrypted representation. [25] illustrates how *privacy homomorphisms* (PH) [24, 32] can be combined with the basic approach described above for this purpose. Additional complexities arise since the information hiding technique does not exactly identify the target group to be aggregated (i.e., the server side results typically contain false positives). The paper develops algebraic manipulation techniques to separate an aggregation group into two subsets a set that *certainly* qualifies the conditions specified in the query, and a set that *may or may not* satisfy the selection predicates of the query (i.e., could contain false positives). The first set can be directly aggregated at the server using PH while the tuples belonging to the second category will need to be transmitted to the client side to determine if they indeed satisfy the query conditions.

Query Optimization in DAS: As in traditional relational query evaluation, in DAS multiple equivalent realizations for a given query are possible. This naturally raises the challenge of query optimization. In [39], query optimization in DAS is formulated as a cost-based optimization problem by introducing new query processing functions and defining new query transformation rules. The intuition is to define transfer of tuples from server to the client and decryption at the client as operators in the query tree. Given different hardware constraints and software capabilities at the client and the server different cost measures are applied to the client-side and server-side computations. A novel query plan enumeration algorithm is developed that identifies the least cost plan.

3.2 Privacy Aware Bucketization

In the previous section we discussed how DAS functionality can be realized when data is represented in the form of buckets. Such a bucketized representation can result in disclosure of sensitive attributes. For instance, given a sensitive numeric attribute (e.g., salary) which has been bucketized, assume that the adversary somehow comes to know the maximum and minimum values occurring in the bucket B. Now he can be sure that all data elements in this bucket have a value that falls in the range $[min_B, max_B]$, thereby leading to partial disclosure of sensitive values for data elements in B. If, the adversary has knowledge of distribution of values in the bucket, he may also be able to make further inference about the specific records. A natural question is how much information does the generalized representation of data reveal that is, given the bucket label, how well can the adversary predict/guess the value of the sensitive attribute of a given entity? Intuitively, this depends upon the granularity at which data is generalized. For instance, assigning all values in the domain to a single bucket will make the bucket-label completely non-informative. However, such a strategy will require the client to retrieve every record from the server. On the other extreme, if each possible data value lies has a corresponding bucket, the client will get no confidentiality, though the records returned by the server will contain no false positives. There is a natural trade-off between the performance overhead and the degree of disclosure. Such a tradeoff has been studied in [16] where authors develop a strategy to minimize the disclosure with constraint on the performance degradation[11].

In the rest of this section, we introduce the measures of disclosure-risk arising from bucketization and then tackle the issue of optimal bucketization to support range queries on a numeric data set. Finally, we present the algorithm that allows one to tune the performance-privacy trade-off in this scheme. The discussion is restricted to the case where bucket based generalization is Performed over a single dimensional ordered data set, e.g., a numeric attribute And the query class is that of 1-dimensional range queries.

Measures of Disclosure Risk

The authors in [14] propose *entropy* and *variance* of the value distributions in the bucket as appropriate measure of (the inverse) disclosure risk. Entropy captures the notion of uncertainty associated with a random element chosen with a probability that follows a certain distribution. The higher the value of entropy of a distribution (i.e., larger the number of distinct values and more uniform the frequencies, larger is the value of the entropy), greater is the uncertainty regarding the true value of the element. For example, given a

[11] Notice the dual of the problem maximize performance with a constraint on information disclosure would also be addressed once we agree on the metric for information disclosure. However, such an articulation of the problem has not been studied in the literature.

domain having 5 distinct values and the data set having 20 data points, the entropy is maximized if all 5 values appear equal number of time, i.e. each value has a frequency of 4.

Now, the adversary sees only the bucket label B of a data element t. Therefore if the adversary (somehow) knows the complete distribution of values within B, he can guess the true value (say v^*) of t with a probability equal to the fractional proportion of elements with value v^* within the bucket. The notion of uncertainty regarding the true value can be captured in an aggregate manner by the entropy of the value distribution within B. Entropy of a discrete random variable X taking values $x_i = 1, \ldots, n$ with corresponding probabilities $p_i, i = 1, \ldots, n$ is given by:

$$Entropy(X) = H(X) = -\sum_{i=1}^{n} p_i log_2(p_i)$$

If the domain of the attribute has an order defined on it as in the case of a numeric attribute, the above definition of entropy does not capture the notion of distance between two values. In the worst case model, since the value distribution is assumed to be known to the adversary, greater the spread of each bucket distribution, better is the protection against disclosure. Therefore, the authors propose *variance* of the bucket distribution as the second (inverse) measure of disclosure risk associated with each bucket. That is, higher the variance of the value distribution, lower is the disclosure risk.

$$Variance(X) = \sum_{i=1}^{n} p_i(x_i - E(X))^2, \text{ where } E(X) = \frac{1}{n}\sum_{i=1}^{n} p_i x_i$$

For more discussion on the choice of these privacy measures refer to [14]. Next we present the criteria for optimal bucketization.

Optimal Buckets for Range Queries

¿From the point of view of security, the best case is to put all data into one bucket, but this is obviously very bad from the performance perspective. Assuming the query workload consists of only range queries, intuitively one can see that more the number of buckets, better is the performance on an average. That is, on an average the number of false positives retrieved will be lesser per range query for a partition scheme that uses more number of buckets. In section 3, we assumed that the bucketization was equiwidth, but as we pointed out earlier, that might not be the best case from the point of view of efficiency. Here we present the analysis (from [14]) that tells us how to compute the optimal buckets given a numeric data set and the number of required buckets.

Consider a relational table with single numeric attribute from a discrete domain like $\mathbf{Z}$ (set of non-negative integers). So given such a data set, what

is the optimal criteria for partitioning this set. The problem can be formally posed as follows. (Refer to the table 1 for notations.)

Problem 1 *Given an input relation $R = (V, F)$ (where V is the set of distinct numeric values appearing at least once in the column and F is the set of corresponding frequencies), a query distribution P (defined on the set of all range queries, Q) and the maximum number of buckets M, partition R into at most M buckets such that the total number of* false positives *over all possible range queries (weighted by their respective probabilities) is minimized.*

For an ordered domain with N distinct values, there are $N(N+1)/2$ possible range queries in the query set Q. The problem of histogram construction for summarizing large data, has similarities to the present problem. Optimal histogram algorithms either optimize their buckets i) independent of the workload, by just looking at the data distribution or ii) with respect to a given workload. The authors first address the following two cases:
1) Uniform: All queries are equi-probable. Therefore probability of any query is $= \frac{2}{N(N+1)}$.
2) Workload-induced: There is a probability distribution P induced over the set of possible queries Q, where the probability of a query q is given by the fraction of times it occurs in the workload W (W is a bag of queries from Q).

The analysis of the uniform query-distribution is given fist and then a discussion on how the general distribution (workload induced) case can be tackled is presented.

Uniform query distribution: The total number of false positives (TFP), where all queries are equiprobable can be expressed as:

$$\text{TFP} \;=\; \sum_{\forall q \in Q} (\; |R^S_{T(q)}| - |R_q| \;)$$

The average query precision (AQP) can be expressed as (see notation in table 1):

$$\text{AQP} \;=\; \frac{\sum_{q \in Q} |R_q|}{\sum_{q \in Q} |R^S_{T(q)}|} \;=\; 1 - \frac{\text{TFP}}{\sum_{q' \in Q'} |R^S_{q'}|}$$

where $q' = T(q)$.

Therefore minimizing the total number of false positives is equivalent to maximizing *average precision* of all queries.

For a bucket B, there are $N_B = (H_B - L_B + 1)$ distinct values (for the discrete numeric domain) where L_B and H_B denote the low and high bucket boundary, respectively. Let V_B denote the set of all values falling in range B and let $F_B = \{f^B_1, \ldots, f^B_{N_B}\}$ denote the set of corresponding value frequencies. Recall that Q is the set of all range queries over the given attribute. One needs to consider all queries that involve at least one value in B and compute the total overhead (false positives) as follows:

V_{min}	minimum possible value for a given attribute
V_{max}	maximum possible value for a given attribute
N	number of possible distinct attribute values; $N = V_{max} - V_{min} + 1$
R	relation (in cleartext), $R = (V, F)$
$\|R\|$	number of tuples in R (i.e. size of table)
V	ordered set (increasing order) of all values from the interval $[V_{min}, V_{max}]$ that occur at least once in R; $V = \{v_i \mid 1 \leq i \leq n\}$
F	set of corresponding frequencies (non-zero); $F = \{0 < f_n \leq \|R\| \mid 1 \leq i \leq n\}$ therefore we have $\|R\| = \sum_{i=1}^{n} f_i$
n	$n = \|V\| = \|F\|$ (Note: $n \leq N$)
R^S	encrypted and bucketized relation, on server
M	maximum number of buckets
Q	set of all "legal" range queries over R
q	a random range query drawn from Q; $q = [l, h]$ where $l \leq h$ and $h, l \in [V_{min}, V_{max}]$
Q'	set of all bucket-level queries
q'	random bucket-level query drawn from Q'; basically q' is a sequence of at least one and at most M bucket identifiers.
$T(q)$	translation function (on the client side) which, on input of $q \in Q$, returns $q' \in Q'$
R_q	set of tuples in R satisfying query q
$R^S_{q'}$	set of tuples in R^S satisfying query q'
W	query workload, induces probability dist on Q

Table 1. Notations for Buckets

Let the set of all queries of size k be denoted by Q_k and $q_k = [l, h]$ denote a random query from Q_k where $h - l + 1 = k$. Then, the total number of queries from Q_k that overlap with one or more points in bucket B can be expressed as: $N_B + k - 1$. Of these, the number of queries that overlap with a single point v_i within the bucket is equal to k. The case for $k = 2$ is illustrated in figure 4. Therefore, for the remaining $N_B - 1$ queries, v_i contributes f_i false positives to the returned set (since the complete bucket needs to be returned). Therefore, for all $N_B + k - 1$ queries of size k that overlap with B, the total number of false positives returned can be written as:

$$\sum_{v_i \in B} (N_B - 1) * f_i = (N_B - 1) * \sum_{v_i \in B} f_i$$

$$= (N_B - 1) * F_B \approx N_B * F_B$$

where F_B is the total number of elements that fall in the bucket (i.e., the sum of the frequencies of the values that fall in B). We make the following important observation here:-

Observation 1 *For the uniform query distribution, the total number of false positives contributed by a bucket B, for set of all queries of size k, is independent of k. In effect the total number of false positives contributed by a bucket (over all query sizes) depends only on the width of the bucket (i.e. minimum and maximum values) and sum of their frequencies.*

From the above observation, it is clear that minimizing the expression $N_B * F_B$ for all buckets would minimize the total number of false-positives for all values of k (over all the $\frac{N(N+1)}{2}$ range queries).

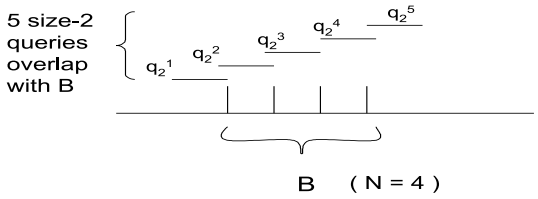

Fig. 4. Queries overlapping with bucket

The Query-Optimal-Bucketization Algorithm (uniform distribution case)

The goal is to minimize the objective function : $\sum_{B_i} N_{B_i} * F_{B_i}$. Let $QOB(1, n, M)$ (*Query Optimal Bucketization*) refer to the problem of optimally bucketizing the set of values $V = \{v_1, \ldots, v_n\}$, using at most M buckets (Note that $v_1 < \ldots < v_n$, each occurring at least once in the table). We make the following two key observations:

1) Optimal substructure property: The problem has the optimal substructure property, therefore allowing one to express the optimum solution of the original problem as the combination of optimum solutions of two smaller sub-problems such that one contains the leftmost $M-1$ buckets covering the $(n-i)$ smallest points from V and the other contains the extreme right single bucket covering the remaining largest i points from V:

$$\begin{aligned} QOB(1,n,M) &= Min_i[QOB(1,n-i,M-1) \\ &\quad +BC(n-i+1,n)] \\ \text{where } BC(i,j) &= (v_j - v_i + 1) * \sum_{i \le t \le j} f_t \end{aligned}$$

($BC(i,j)$ is cost of a single bucket covering $[v_i, v_j]$)

2) Bucket boundary property: It can be intuitively seen that for an optimal solution, the bucket boundaries will always coincide with some value from the set V (i.e. values with non-zero frequency). Therefore in the solution space, one needs to consider only buckets whose end points coincide with values in V, irrespective of the total size of the domain.

The algorithm solves the problem bottom-up by solving and storing solutions to the smaller sub-problems first and using their optimal solutions to solve the larger problems. All intermediate solutions are stored in the 2-dimensional matrix H. The rows of H are indexed from $1, \ldots, n$ denoting the number of leftmost values from V that are covered by the buckets for the given sub-problem and the columns are indexed by the number of maximum allowed buckets (from $1, \ldots, M$). Also note that the cost of any single bucket covering a consecutive set of values from V can be computed in constant time by storing the cumulative sum of frequencies from the right end of the domain, call them $EndSum$ (i.e. $EndSum_n = f_n, EndSum_{n-1} = f_{n-1} + f_n \ldots$. Storing this information uses $O(n)$ space. The algorithm also stores along with the optimum cost of a bucketization, the lower end point of its last bucket in the $n \times M$ matrix OPP (Optimal Partition Point) for each sub-problem solved. It is easy to see that the matrix OPP can be used to reconstruct the exact bucket boundaries of the optimal partition computed by the algorithm in $O(M)$ time. The dynamic programming algorithm is shown in figure 5[12] and an illustrative example is given below.

Example: Assume the input to QOB algorithm is the following set of (data-value, frequency) pairs:
$D = \{(1,4),(2,4),(3,4),(4,10),(5,10),(6,4),(7,6),(8,2),(9,4),(10,2)\}$ and say the maximum number of buckets allowed is 4, then (figure 6) displays the optimal histogram that minimizes the cost function. The resulting partition is $\{1,2,3\},\{4,5\},\{6,7\},\{8,9,10\}$. Note that this histogram is not equi-depth (i.e all bucket need not have the same number of elements). The minimum value of the cost function comes out to be = 120. In comparison the approximately equi-depth partition $\{1,2,3\},\{4\},\{5,6\},\{7,8,9,10\}$ has a cost = 130. ♢

Generic Query Workload: The same dynamic programming algorithm of figure 5 can be used for an arbitrary distribution induced by a given query workload W. The workload is represented by $W = \{(q, w_q) | q \in W \cap w_q > 0\}$, where w_q is the probability that a randomly selected query from W is same as q. As in the case of uniform workload, the optimal substructure property holds in this case as well. The only difference is in computation of the bucket costs BC which translates into the computation of the array $Endsum[1, \ldots, n]$ in the preprocessing step. The algorithm to compute the entries of the $EndSum$

[12] in the workload-induced case, only the $EndSum$ computation is done differently, the rest of the algorithms remains the same

Algorithm: QOB(D, M)
Input: Data set $D = (V, F)$ and max # buckets M
(where $|V| = |F| = n$)
Output: Cost of optimal bucketization & matrix H
Initialize
(i) matrix $H[n][M]$ to 0
(ii) matrix $OPP[n][M]$ to 0
(iii) compute $EndSum(j) = EndSum(j+1) + f_j$
for $j = 1 \ldots n$
For $k = 1 \ldots n$ // For sub-problems with max 2 buckets
$H[k][2] = Min_{2 \le i \le k-1}(BC(1, i) + BC(i+1, K))$
Store *optimal-partition-point* i_{best} in $OPP[k][2]$
For $l = 3 \ldots M$ // For the max of 3 up to M buckets
For $k = l \ldots n$
$H[k][l] = Min_{l-1 \le i \le k-1}(H[i][l-1] + BC(i+1, k))$
Store *optimal-partition-point* i_{best} in $OPP[k][l]$
Output "Min Cost of Bucketization = $H[n][M]$"
end

Fig. 5. Algorithm to compute query optimal buckets

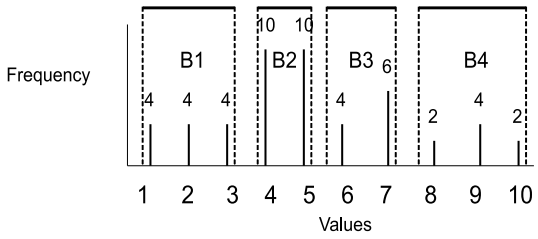

Fig. 6. Optimum buckets for uniform query distribution

array for a generic workload W is given in figure 7. Any bucket B covering range $[i, j]$ will have the cost given by $BC(i, j) = EndSum[i] - EndSum[j]$.

Next we address the issue how the security and performance tradeoff.

The Security-Performance Trade-off

The optimal buckets offer some base level of security due to the indistinguishability of elements within a bucket, but in many cases that might not be good enough, i.e., a bucket's value distributions might not have a large enough variance and/or entropy. To address this problem, the authors in [14] propose a re-bucketization of the data, starting with the optimal buckets and allowing a bounded amount of performance degradation, in order to maximize the two

Algorithm: Compute-EndSum$(D, W, EndSum[1 \ldots n])$
Input: $D = (V, F)$, workload W & $EndSum[1 \ldots n]$
(note: $EndSum[1 \ldots n]$ is initialized to $0's$)
Output: Array $EndSum$ with entries filled in
For $i = (n-1) \ldots 1$
$EndSum[i] = EndSum[i+1]$
For all $q = [l_q, h_q] \in W$ such that $h_q \in [v_i, v_{i+1})$
$EndSum[i] = EndSum[i] + (\sum_{j=i+1}^{n} f_j) * w_q$
For all $q = [l_q, h_q] \in W$ such that $l_q \in (v_i, v_n]$
$EndSum[i] = EndSum[i] + f_i * w_q$
Output $EndSum[1 \ldots n]$
end

Fig. 7. EndSum for generic query workload W

privacy measures (**entropy** and **variance**) simultaneously. The problem is formally presented below:

Problem 2 Trade-off Problem*: Given a dataset $D = (V, F)$ and an optimal set of M buckets on the data $\{B_1, B_2, \ldots, B_M\}$, re-bucketize the data into M new buckets, $\{CB_1, CB_2, \ldots, CB_M\}$ such that no more than a factor K of performance degradation is introduced and the* **minimum variance** *and* **minimum entropy** *amongst the M random variables $X_1, \ldots, X_M$ are simultaneously* **maximized***, where the random variable X_i follows the distribution of values within the i^{th} bucket.*

The above mentioned problem can be viewed as a multi-objective constrained optimization problem, where the entities *minimum entropy* and *minimum variance* amongst the set of buckets are the two objective functions and the constraint is the *maximum allowed performance degradation factor* K (called the *Quality of Service*(QoS) constraint). Such problems are combinatorial in nature and the most popular solution techniques seem to revolve around the *Genetic Algorithm* (GA) framework [27], [28]. GA's are iterative algorithms and cannot guarantee termination in polynomial time. Further their efficiency degrades rapidly with the increasing size of the data set. Therefore instead of trying to attain optimality at the cost of efficiency, the authors propose a new algorithm called the *controlled diffusion algorithm* (CDf-algorithm). The CDf-algorithm increases the privacy of buckets substantially while ensuring that the performance constraint is not violated.
Controlled Diffusion: The optimal bucketization for a given data set is computed using the QOB-algorithm presented in figure 5. Let the resulting optimal buckets be denoted by $B_i's$ for $i = 1, \ldots, M$. The controlled diffusion process creates a new set of M approximately equi-depth buckets which are called the *composite buckets* (denoted by $CB_j, j = 1, \ldots, M$) by *diffusing* (i.e. re-distributing) elements from the B_i's into the CB_j's. The diffusion process

is carried out in a controlled manner by restricting the number of distinct CB's that the elements from a particular B_i get diffused into. This resulting set of composite buckets, the $\{CB_1, \ldots, CB_M\}$ form the final bucketized representation of the client data. Note that, to retrieve the data elements in response to a range query q, the client needs to first compute the query overlap with the optimal buckets B_i's, say $q(B)$ and then retrieve all the contents of composite buckets CB_j's that overlap with one or more of the buckets in $q(B)$. The retrieved data elements comprise the solution to query q.

The M composite buckets need to be approximately equal in size in order to ensure the QoS constraint, as will become clear below. The equi-depth constraint sets the target size of each CB to be a constant $= f_{CB} = |D|/M$ where $|D|$ is size of the data set (i.e. rows in the table). The QoS constraint is enforced as follows: If the maximum allowed performance degradation $= K$, then for an optimal bucket B_i of size $|B_i|$ its elements are diffused into no more than $d_i = \frac{K*|B_i|}{f_{CB}}$ composite buckets (as mentioned above $f_{CB} = |D|/M$). The diffusion factor d_i is rounded-off to the closest integer. Assume that in response to a range query q, the server using the set of optimal buckets $\{B_1, \ldots, B_M\}$, retrieves a total of t buckets containing T elements in all. Then in response to the same query q this scheme guarantees that the server would extract no more than $K * T$ elements at most, using the set $\{CB_1, \ldots, CB_M\}$ instead of $\{B_1, \ldots, B_M\}$. For example, if the optimal buckets retrieved in response to a query q were B_1 and B_2 (here $t = 2$ and $T = |B_1| + |B_2|$), then to evaluate q using the CB_j's, the server won't retrieve any more than $K * |B_1| + K * |B_2|$ elements, hence ensuring that precision of the retrieved set does not reduce by a factor greater than K.

An added advantage of the diffusion method lies in the fact that it guarantees the QoS lower bound is met not just for the average precision of queries but for each and every individual query. The important point to note is that the domains of the composite buckets overlap where as in the case of the optimal buckets, they do not. Elements with the same value can end up going to multiple CB's as a result of this diffusion procedure. This is the key characteristic that allows one to vary the privacy measure while being able to control the performance degradation. Therefore, this scheme allows one to explore the "privacy-performance trade-off curve". The controlled diffusion algorithm is given in figure 8. The diffusion process is illustrated by an example below.

Example: Consider the optimal buckets of the example in figure 6 and say a performance degradation of up to 2 times the optimal ($K = 2$) is allowed. Figure 9 illustrates the procedure. In the figure, the vertical arrows show which of the composite buckets, the elements of an optimal bucket gets assigned to (i.e. diffused to). The final resulting buckets are shown in the bottom right hand-side of the figure and we can see that all the 4 CB's roughly have the same number size (between 11 and 14). The average entropy of a bucket increases from 1.264 to 2.052 and standard deviation increases from 0.628 to 1.875 as one goes from the B's to CB's. In this example the entropy increases

Algorithm : Controlled-Diffusion(D, M, K)
Input : Data set $D = (V, F)$,
M = # of $CB's$ (usually same as # opt buckets)
K = maximum performance-degradation factor
Output : An M-Partition of the dataset (i.e. M buckets)

Compute optimal buckets $\{B_i, \ldots, B_M\}$ using QOB algo
Initialize M empty composite buckets $CB_1 \ldots, CB_M$
For each B_i
Select $d_i = \frac{K*|B_i|}{f_{CB}}$ distinct CB's randomly, $f_{CB} = \frac{|D|}{M}$
Assign elements of B_i **equiprobably** to the d_i $CB's$
(roughly $|B_i|/d_i$ elements of B_i go into each CB)
end For
Return the set buckets $\{CB_j | j = 1, \ldots, M\}$.
end

Fig. 8. Controlled diffusion algorithm

since the number of distinct elements in the $CB's$ are more than those in the B's. The variance of the CB's is also higher on an average than that of the B's since the domain (or spread) of each bucket has increased. Note that average precision of the queries (using the composite buckets) remains within a factor of 2 of the optimal. For instance, take the range query $q = [2, 4]$, it would have retrieved the buckets B_1 and B_2 had we used the optimal buckets resulting in a precision of $18/32 = 0.5625$. Now evaluating the same query using the composite buckets, we would end up retrieving all the buckets CB_1 through CB_4 with the reduced precision as $18/50 \approx 0.36 > \frac{1}{2} * 0.5625$. (Note: Due to the small error margin allowed in the size of the composite buckets (i.e. they need not be exactly equal in size), the precision of few of the queries might reduce by a factor slightly greater than K). $\diamondsuit$

Discussion

In this section, we considered only single dimensional data. Most real data sets have multiple attributes with various kinds of dependencies and correlations between the attributes. There may be some kinds of functional dependencies (exact or partial) and correlations as in multidimensional relational data or even structural dependencies as in XML data. Therefore, knowledge about one attribute might disclose the value of another via the knowledge of such associations. The security-cost analysis for such data becomes significantly different as shown in [16, 35]. Also, in this section, the analysis that was presented, was carried out for the worst-case scenario where it was assumed that the complete value distribution of the bucket is known to an adversary. In reality it is unrealistic to assume that an adversary has exact knowledge of

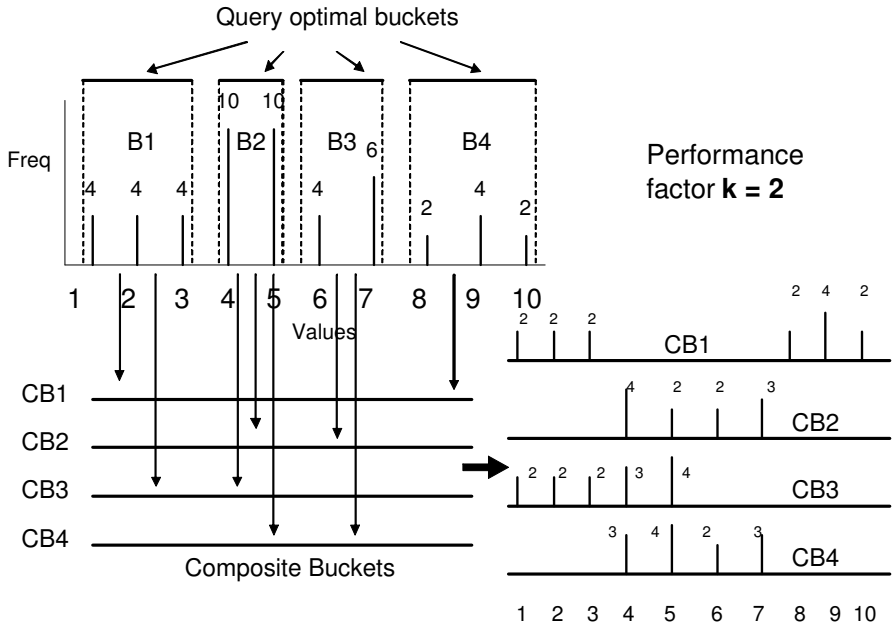

Fig. 9. controlled diffusion (adhoc version)

the complete distribution of a data set. Moreover, to learn the bucket-level joint-distribution of data, the required size of the training set (in order to approximate the distribution to a given level of accuracy) grows exponentially with the number of attributes/dimensions. This makes the assumption of "complete bucket-level" knowledge of distribution even more unrealistic for multidimensional data. [16] proposes a new approach to analyze the disclosure risk for multidimensional data and extends the work in [14] to the this case.

3.3 Two Server Model for DAS

The authors in [4] propose an alternate model for DAS where they propose a distributed approach to ensure data confidentiality. This scheme requires the presence of two or more distinct servers. The individual servers can be untrusted as before, the only constraint being, they should be mutually non-colluding and non-communicating. Also deviating from the previous approach, here the data owner is required to specify the privacy requirement in the form of constraints as we describe below.

Privacy Requirements

Consider a relation R, the privacy requirements are specified as a set of *constraints* $\mathcal{P}$ expressed on the schema of relation R. Each constraint is denoted by a subset, say P, of the attributes of R. The privacy constraints informally mean the following: If R is decomposed into into R_1 and R_2, and let an adversary have access to the entire contents of either R_1 or R_2. The privacy requirement is that for every tuple in R, the value of at least one of the attributes in P should be completely opaque to the adversary, i.e., the adversary should be unable to infer anything about the value of that attribute.

For example, let relation R consist of the attributes *Name, Date of Birth (DoB), Gender, Zipcode, Position, Salary, Email, Telephone*. The company specifies that *Telephone* and *Email* are sensitive even on their own. *Salary, Position* and *DoB* are considered private details of individuals and so cannot be stored together. Similarly {*DoB, Gender, Zipcode*} might also be deemed sensitive since they together can identify an individual. Other things that might be protected are like sensitive rules, e.g., relation between position and salary or between age and salary: {*Position, Salary*}, {*Salary, DoB*}.

The goal is therefore to ensure that each of the the privacy constraints are met. For constraints containing single sensitive attributes, e.g., the "Telephone Number" that needs to be hidden, one can XOR the number with a random number r and store the resulting number in one server and the number r on another. To recover the original telephone number one simply needs to XOR these two pieces of information and each of these by themselves reveal nothing. For the other kind of constraints that contain more than one attribute, say {*Salary, Position*}, the client can vertically partition the attribute sets of R so that the salary information is kept in one column and the position information is kept in another. Since the two servers are assumed to be non-colluding and non-communicating, distributing the attributes across the servers provides an approach to implement such multi-attribute privacy policies. The decomposition criteria is described more formally below.

Relational Decomposition

The two requirements from the decomposition of a relational data set is that it should be *lossless* and *privacy preserving*. Traditional relation decomposition in distributed databases is of two types: *Horizontal Fragmentation* where each tuple of a relation R is either stored in server S_1 or server S_2 and *Vertical Fragmentation* where the attribute set is partitioned across S_1 and S_2 [40]. The vertical fragmentation of data is one that is investigated as a candidate partitioning scheme and is investigated further in this work. Vertical partitioning requires the rows in the two servers to have some unique tuple ID associated with them. They also propose to use *attribute encoding* schemes where can be built by combining the parts stored on the two servers. One-time pad, deterministic encryption and random noise addition are explored as alternative candidates for semantic partitioning of an attribute. The authors propose using partitioning of the attribute set along with attribute encoding to meet the privacy constraints. Remember that $\mathcal{P} \subseteq 2^R$. Consider a decomposition of R as $\mathcal{D}(R) =< R_1, R_2, E >$, where R_1 and R_2 are the sets of attributes in the two fragments, and E refers to the set of attributes that are encoded. $E \subseteq R_1$ and $E \subseteq R_2$ and $R_1 \cup R_2 = R$. Then the privacy constraint need to satisfy the following requirements.

Condition 1 *The decomposition $\mathcal{D}(R)$ is said to obey the privacy constraints $\mathcal{P}$ if, for every $P \in \mathcal{P}$, $P \nsubseteq (R_1 - E)$ and $P \subsetneq (R_2 - E)$*

Each constraint $P \in \mathcal{P}$ can be obeyed in two ways:

1. Ensure that P is not contained in either R_1 or R_2, using vertical fragmentation. For example, the privacy constraint {*Name,Salary*} may be obeyed by placing *Name* in R_1 and *Salary* in R_2
2. Encode at least one of the attributes in P. For example, a different way to obey the privacy constraint {*Name, Salary*} would be to encode *Salary* across R_1 and R_2. For a more detailed discussion, we point the interested reader to [4]

Query Reformulation, Optimization & Execution

The suggested query reformulation is straightforward and identical to that in distributed databases. When a query refers to R, it is replaced by $R_1 \bowtie R_2$, where all the encoded tuples (i.e., those that occur across both R_1 and R_2) are assumed to be suitably decoded in the process. Consider the query with selection condition c. If c is of the form $< Attr >< op >< value >$, and $< Attr >$ has not undergone fragmentation, condition c, may be pushed down to R_1 or R_2, whichever contains $< Attr >$. Similarly if c is of the form $< Attr1 >< op >< Attr2 >$ and both attributes are un-fragmented and present in either R_1 or R_2, the condition may be pushed down to the appropriate relation. Similar "push down" operations can also be applied to projection,

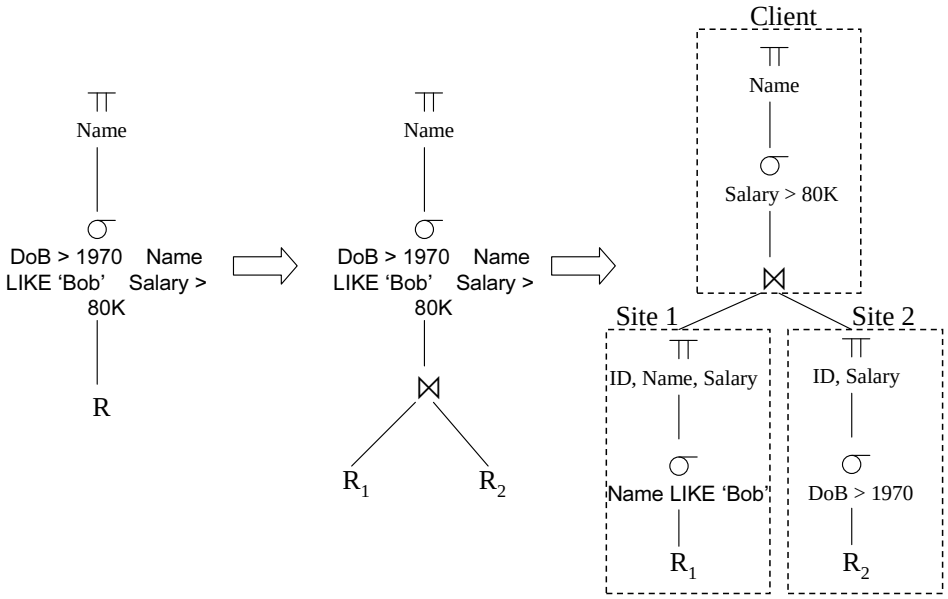

Fig. 10. Example of Query Reformulation and Optimization)

Group-by and aggregation clauses. When the computation cannot be done without decoding (or merging from the two servers) then the computation shifts to the client. Notice that unlike in distributed database query processing where semijoin approach (and shipping fields to other servers can be utilized) such techniques cannot be used since that may result in violation of the privacy constraint.

The small example in figure 10 illustrates the process of logical query plan generation and execution in the proposed architecture. For some more examples and discussions, the interested reader can refer to section 4 in the paper [4].

Optimal Decomposition for Minimizing Query Cost

For a given workload W of queries, there is a cost associated with each distinct partitioning of the set of attributes between the two databases. Since there are exponentially many distinct partitioning schemes, a brute force evaluation is not efficient. The authors use an approach from distributed computing based on *affinity matrix* M, where the entry M_{ij} represents the "cost" of placing the unencoded attributes i and j in different segments. The entry M_{ij} represents the "cost" of encoding attribute i across both fragments. The cost of decomposition is assumed to be simply represented by a linear combination of entries in the affinity matrix. Let $R = \{A_1, A_2, \ldots, A_n\}$

represents the original set of n attributes, and consider a decomposition of $\mathcal{D}(R) =< R_1, R_2, E >$. Then we assume the cost of this decomposition $C(\mathcal{D}) = \sum_{i \in (R_1 - E), j \in (R_2 - E)} M_{ij} + \sum_{i \in E} M_{ii}$. Now, the optimization problem is to minimize the above cost while partitioning the data in a way that satisfies all the privacy constraints. The two issues to be addressed are: (i) How can the affinity matrix M be generated from a knowledge of the query workload? (ii) How can the optimization problem be solved?

The simplest scheme to populate the affinity matrix is to set M_{ij} to be the number of queries that refer to both attributes i and j. Similarly M_{ii} is set to the number of queries in the workload that refer to attribute i. Some other heuristics to populate the entries of the affinity matrix are given in the appendix of the paper [4] and the interested reader can refer to it. Now, we summarize the solution outlined in the paper for the second problem.

The Optimization Problem

The optimization problem is modelled as a complete graph $G(R)$, with both vertex and edge weights defined by the affinity matrix M. (Diagonal entries stand for vertex weights). Along with it, the set of privacy constraints are also given, $\mathcal{P} \subseteq 2^R$, representing a hypergraph $H(R, \mathcal{P})$ on the same vertices. The requirement is then, to 2-color the set of vertices in R such that (a) no hypergraph edge in H is monochromatic, and (b) the weight of bichromatic graph edges in G is minimized. The difference is that an additional freedom to delete any vertex in R (and all the hyperedges that contain it) by paying a price equal to the vertex weight. The coloring of a vertex is equivalent to placing the corresponding attribute in one of the two segments and deleting it is equivalent to encoding the attribute; so all privacy constraints associated with the attribute is satisfied by the vertex deletion. Some vertex deletions might always be necessary since it might not be possible to always two color a hypergraph.

The above problem is NP-hard and the authors go on to give three heuristic solutions that use approximation algorithms for "Min-Cut" and "Weighted Set Cover" problems in graphs. The former component is used to determine two-colorings of the graph $G(R)$ (say all cuts of the graph $G(R)$ which are within a small constant factor of the minimum cut) and is used to decide which attributes are assigned to which segment. The weighted set cover algorithm is used for detecting the least costly set of vertices to delete, these correspond to attributes that need to be encoded across both the segments. Good approximation solutions are there for both the algorithms and therefore makes them practical to use. We summarize one of the heuristic approaches mentioned in the paper, and refer the interested reader to the original paper for the remaining ones.

1. Ignore fragmentation, and delete vertices to cover all the constraints using **Approximate Weighted Set Cover**. Call the set of deleted vertices E.

2. Consider the remaining vertices, and use **Approximate Min-Cuts** to find different 2-colorings of the vertices, all of which approximately minimize the weight of the bichromatic edges in G
3. For each of the 2-colorings obtained in step (2): Find all deleted vertices that are present only in bichromatic hyperedges, and consider "rolling back" their deletion, and coloring them instead, to obtain a better solution.
4. Choose the best of (a) the solution from step (3) for each of the 2-colorings, and (b) the decomposition $< R - E, E, E >$.

In the first step, all the privacy constraints are covered by ensuring at least one attribute in each constraint is encoded. This implies a simple solution $\mathcal{D}_1$, where all the unencoded attributes are assigned to one segment. In steps (2) and (3), one tries to improve upon this by avoiding encrypting all the attributes, hoping to use fragmentation to cover some of the constraints. This is done iteratively by computing various other small cuts and trying to roll back some of the encodings. Finally the better solution (computed in step (3) or $\mathcal{D}_1$) is returned.

Discussion

In this paper, the authors have proposed an alternative way of supporting database functionalities in a secure manner in the outsourcing model. The approach deviates from the usual DAS model in that it requires two or more non-communicating service providers to be available to enable privacy. The approach though novel and definitely interesting, does raise many questions which would be interesting to investigate, e.g., what would be the performance of such a system in a real deployment, how would more complicated queries be split between the two servers, what would be the overhead of multiple-rounds in which a query might have to be answered. Also, currently there is no direct communication between the two servers, could some of the distributed query processing overhead be reduced if some kind of secure, "two-party communications" were to be enabled between these two servers. Moreover, the nature of privacy violation due to associations between multiple attributes (that are kept on different servers or are encoded) could lead to privacy violations, how to tackle these is not known. The paper does offer an interesting new approach which could give rise to some new research.

3.4 Summary

In this section, we presented example approaches that have been developed in the literature to support relational queries over encrypted relational data in the DAS setting. Such approaches broadly fall under two categories: (1) cryptographic approaches that attempt to develop encryption algorithms that allow queries to be evaluated directly over encrypted representation, and (2)

approaches based on exploiting information hiding techniques developed in the disclosure control literature. While cryptographic approaches prevent leakage of any information (but are applicable only under limited scenarios), approaches based on information hiding may allow limited information disclosure but are more widely applicable. Such approaches explore a natural tradeoff between potential information loss and efficiency that can be achieved. We discussed how data generalization into buckets can be performed such that the information disclosure is minimized while constraining the performance degradation.

We note that information disclosure in bucket-based approach has formally only been studied for single dimensional data, under the worst case assumption that the adversary knows the definition of the buckets and the complete distribution of data within buckets. As we noted earlier, this assumption is too strong in reality and cannot be expected to hold in general for multidimensional data. Some recent work has been done in this area [16] to develop a framework for security analysis for multidimensional bucketization.

There also exist some work in literature [19] that propose data transformation based schemes which allow a richer set of SQL functions to be computed directly on the transformed numeric data like aggregation and sorting. Though, the main shortcoming of this scheme is that it is only secure under the "ciphertext-only attack" model and breaks down when the adversary possesses background knowledge and/or a few plaintext-ciphertext pairs.

Finally, while our focus in this chapter has been on techniques developed in the literature to support relational queries in the DAS model, we note that limited systems that follow the DAS paradigm have also been built [3, 18]. Whereas [3] uses a smart card based safe storage to implement limited SQL functionalities, [18] proposes a system which outsources user profile data (e.g., bookmarks, passwords, query logs, web data, cookies, etc) and supports simple searching capabilities. The rationale is to use DAS approach for supporting mobility across different software and machines. some limited work on schemes such as [19]

4 Conclusions

In this chapter, we summarized research on supporting search over encrypted data representation that has been studied in the context of database as a service model. Much of the existing work has studied the search problem in one of the two contexts: keyword search over encrypted document representations, and SQL search over encrypted relational data. Since the initial work [7, 26] in these areas, many extensions to the problem have been considered. We briefly mention these advances that we have not covered so far to provide interested readers with references.

The problem of query evaluation over encrypted relational databases has been generalized to XML data sources in [35, 36]. XML data, unlike relational databases, is semi-structured which introduces certain additional complications in encrypting as well as translating queries. For instance, authors in [35] propose XML-encryption schemes taking its structure into consideration, develop techniques to evaluate SPJ queries and optimize the search process during query processing.

Besides extending the data model, some researchers have considered relaxing assumptions made by the basic DAS model itself. The basic DAS model, as discussed in this chapter, assumes that the service provider though untrusted, is honest. Such an assumption might not necessarily hold in certain situations. In particular, the service provider may return erroneous data. An error in the result to a query may manifest itself in two ways – the returned answers may be tampered by the service provider, or alternatively, the results returned by the service provider may not be the complete set of matching records. The problem of integrity of the returned results was first studied in [26] which used message authentication codes (MACs) to authenticate the result set. Any such authentication mechanism adds additional processing cost at the client. Authentication mechanisms using Merkle Hash trees and group signatures that attempt to reduce such an overhead have been studied in [37]. The authors have developed techniques for both the situation where the client (i.e., the user who poses the query) is the same as well as different from the data owner.

Another avenue of DAS research has been to exploit secure coprocessor to maintain confidentiality of outsourced database [38]. Unlike the basic DAS model in which the client is trusted and the service provider is entirely untrusted, in the model enhanced with a secure coprocessor, it is assumed that the service provider has a tamper proof hardware – a secure coprocessor – which is attached to the untrusted server and has (limited) amount of storage and processing capabilities. Data while outside the secure processor must be in the encrypted form, it could be in plaintext within the coprocessor without jeopardizing data confidentiality. Exploiting a secure coprocessor significantly simplifies the DAS model since now intermediate query results do not need to be transmitted to the clients if further computation requires data to be in plaintext. Instead, secure coprocessor can perform such a function, there-

fore significantly reducing network overheads and optimizing performance. Another additional advantage is that such a model can naturally support situations where the owner of the database is different from the user who poses the query.

While much progress in research has been made over the past few years on DAS, we believe that many further challenges remain before the vision outlined in [26] of a secure data management service that simultaneously meets the data confidentiality and efficiency requirements. A few of the many practical challenges that still remain open are the following: (1) techniques to support dynamic updates – some initial approaches to this problem have been studied in [10], (2) mechanisms to support stored procedures and function execution as part of SQL processing, and (3) support for a more complete SQL - e.g., pattern matching queries. Furthermore, given multiple competing models for DAS (e.g., the basic model, the model with secure coprocessor, model with two servers) as well as multiple competing approaches, there is a need for a detailed comparative study that evaluates these approaches from different perspectives: feasibility, applicability under diverse conditions, efficiency, and achievable confidentiality. Finally, a detailed security analysis including the nature of attacks as well as privacy guarantees supported by different schemes needs to be studied.

5 Acknowledgements

This work has been possible due to the following NSF grants: 0331707 and IIS-0220069.

References

1. D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano Public Key Encryption with Keyword Search. In *Advances in Cryptology - Eurocrypt 2004* (2004). volume 3027 of *Lecture Notes in Computer Science*, pp. 506-522. Springer-Verlag, 2004.
2. L. Ballard, S. Kamara, F. Monrose Achieving Efficient conjunctive keyword searches over encrypted data. In *Seventh International Conference on Information and Communication Security (ICICS 2005)*, volume 3983 of *Lecture Notes in Computer Science*, pp. 414-426. Springer, 2005.
3. L. Bouganim, and P. Pucheral. Chip-Secured Data Access: Confidential Data on Untrusted Servers In *Proc. of VLDB* 2002.
4. G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, U. Srivastava, D. Thomas, Y. Xu. Two Can Keep a Secret: A Distributed Architecture for Secure Database Services In *Proc. of CIDR* 2005.
5. S. Chaudhuri. An overview of query optimization in relational systems. In *Proc. of ACM Symposium on Principles of Database Systems (PODS)*, 1998.
6. E-J., Goh Secure Indexes. Technical report 2003/216, In IACR ePrint Cryptography Archive, (2003). See http://eprint.iacr.org/2003/216.
7. D. Song and D. Wagner and A. Perrig. Practical Techniques for Search on Encrypted Data. In *Proc. of IEEE SRSP*, 2000.
8. H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2002.
9. G. Graefe. Query eveluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.
10. H. Hacıgümüş. Privacy in Database-as-a-Service Model. *Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine*, 2003.
11. H. Hacıgümüş, B. Iyer, and S. Mehrotra. Encrypted Database Integrity in Database Service Provider Model. In *Proc. of Certification and Security in E-Services (CSES'02), IFIP 17th World Computer Congress*, 2002.
12. H. Hacıgümüş, B. Iyer, and S. Mehrotra. Providing Database as a Service. In *Proc. of ICDE*, 2002.
13. H. Hacıgümüş, B. Iyer, and S. Mehrotra. Ensuring the Integrity of Encrypted Databases in Database as a Service Model. In *Proc. of 17th IFIP WG 11.3 Conference on Data and Applications Security*, 2003.
14. B. Hore, S. Mehrotra, and G. Tsudik. A Privacy-Preserving Index for Range Queries. In *Proc. of VLDB* 2004.
15. B. Hore, S. Mehrotra, and G. Tsudik. A Privacy-Preserving Index for Range Queries. UCI-ICS tech report 2004, http://www.ics.uci.edu/ bhore/papers.
16. B. Hore, and S. Mehrotra. Supporting Multidimensional Range and Join Queries over Remote Encrypted Databases. UCI-ICS tech report 2006.
17. B. Iyer, S. Mehrotra, E. Mykletun, G. Tsudik, and Y. Wu A Framework for Efficient Storage Security in RDBMS In *Proc. of EDBT* 2004.
18. R. C. Jammalamadaka, S. Mehrotra, and N. Venkatasubramanian PVault: A Client-Server System Providing Mobile Access to Personal Data In *International Workshop on Storage Security and Survivability*, 2005.
19. R. Agrawal, J. Kiernan, R. Srikant, Y. Xu Order Preserving Encryption for Numeric Data In *Proc. of SIGMOD* 2004.

20. P. Sellinger, M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access Path Selection in Relational Database Management Systems. In *Proc. of ACM SIGMOD*, 1979.
21. Y. Chang and M. Mitzenmacher Privacy preserving keyword searches on remote encrypted data. In *Third International Conference on Applied Cryptography and Network Security (ACNS 2005)*, volume 3531 of *Lecture Notes in Computer Science*, pp. 442-455. Springer-Verlag, 2005.
22. P. Golle, J. Staddon, B. Waters Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security (ACNS 2004)*, volume 3089 of *Lecture Notes in Computer Science*, pp. 31-45. Springer, 2004.
23. B. Waters, D. Balfanz, G. Durfee, and D. Smetters Building and encrypted and searchable audit log. In *Network and Distributed System Security Symposium (NDSS 2004)* (2004), The Internet Society, 2004.
24. R. Rivest, R.L. Adleman and M. Dertouzos. On Data Banks and Privacy Homomorphisms. In *Foundations of Secure Computations*, 1978.
25. H. Hacıgümüş, B. Iyer, and S. Mehrotra Efficient Execution of Aggregation Queries over Encrypted Relational Databases. In *Proceedings of DASFAA*, 2004.
26. H. Hacıgümüş, B. Iyer, C. Li and S. Mehrotra Executing SQL over encrypted data in the database-service-provider model. In Proc. SIGMOD, 2002.
27. D.E. Goldberg Genetic Algorithms in Search, Optimization and Machine Learning. Published by Addison-Wesley, Reading, Massachusetts.
28. D.R. Jones and M.A. Beltramo Solving Partitioning Problems with Genetic Algorithms. In *Proc. of the 4th International Conference of Genetic Algorithms*, 1991.
29. L. Willenborg, T. De Waal Statistical Disclosure Control in Practice *Springer-Verlag*, 1996.
30. L. Willenborg, T. De Waal Elements of Statistical Disclosure Control *Springer*, 2001.
31. N.R. Adam, J.C. Worthmann Security-control methods for statistical databases: a comparative study In *ACM Computing Surveys, Vol 21, No. 4*, 1989.
32. J. Domingo-Ferrer A New Privacy Homomorphism and Applications In *Information Processing Letters*, 6(5):277-282, 1996.
33. N. Ahituv, Y. Lapid, S. Neumann Processing Encrypted Data In *Communications of the ACM*, 1987 Vol. 30, 9, pp.777-780
34. E. Brickell, Y. Yacobi On Privacy Homomorphisms In *Proc. Adavances in Cryptology-Eurocrypt'87*
35. R. Jammalamadaka, S. Mehrotra Querying Encrypted XML Documents *MS thesis*, 2004.
36. H. Wang, L. Lakshmanan Efficient Secure Query Evaluation over Encrypted XML Databases In *VLDB*, 2006.
37. M. Narasimhan, G. Tsudik DSAC: Integrity of Outsourced Databases with Signature Aggregation and Chaining In *CIKM*, 2005.
38. R. Agrawal, D. Asonov, M. Kantarcioglu, Y. Li Sovereign Joins In *ICDE* 2006.
39. H. Hacıgümüş, B. Iyer, S. Mehrotra Query Optimization in Encrypted Database Systems, In *Proc. of International Conference on Database Systems for Advanced Applications (DASFAA)*, 2005.
40. M. T. Ozsu, and P. Valduirez Principles of Distributed Database Systems, *Prentice Hall*, 2^{nd} Edition, 1999.