Blind Custodians: A Database Service Architecture that Supports Privacy without Encryption

Amihai Motro
ISE Department
George Mason University
ami@gmu.edu

Francesco Parisi-Presicce
ISE Department
George Mason University
fparisip@gmu.edu

Abstract

We describe an architecture for a database service that does not assume that the service provider can be trusted. Unlike other architectures that address this problem, this architecture, which we call *blind custodians*, does not rely on encryption. Instead, it offers confidentiality by means of *information dissociation:* The server only stores "fragments" of information that are considered safe (i.e., each fragment does not violate privacy), while the client stores the associations between the fragments that are necessary to reconstruct the information. We argue that this architecture allows satisfactory confidentiality, while offering two important advantages: (1) It does not restrict the types of queries that can be submitted by clients (as encryption-based methods invariably do), and (2) it requires only light processing at the client, assigning the bulk of the processing to the server (as befits a true service). Moreover, the architecture permits flexible control over the level of confidentiality that should be maintained (at the cost of additional overhead).

1 Introduction

With the improvement of network availability, reliability, and speed, more and more information management tasks that traditionally have been performed on the computer of the data owner, are now being offered as *services*: the information is stored in the computer of the service provider, and can be managed remotely over the network. Primary examples include web hosting, the management of electronic mail, appointment calendars, address books, investment portfolios, and, more recently, general database management services. Such information management services provide clients with performance and features at a level often not available at their own enterprises. These include continuous availability, accessibility from virtually anywhere in the world, reliable data backup and prompt recovery, and protection from unauthorized or malicious access. In many cases, a remote information management service can prove to be more cost-effective than traditional, local processing.

The downside of such information management services is that *privacy* may have to be sacrificed. The operators of the service, the *custodians* of the information, may promise complete confidentiality, but they themselves may not be entirely trustworthy. From the client's point of view, custodians should be *blind*; that is, they should perform the functionalities promised, but without being able to observe the data themselves. We shall refer to such information services as *blind custodians*. In a way, a bank that offers safe deposit boxes to its clients is serving as a blind custodian: It safekeeps the contents without knowing what it is. The complication in being the blind custodian of information is that you are also required to *manipulate* the information without knowing what it is.

There have been several recent works that addressed this or similar issues [9, 8, 5, 6]. Invariably, these works describe architectures that protect the information by means of *encryption*. By encrypting the information, the client is guaranteed that it alone can observe the data. The problem, of course, is how to perform functions such as *selective retrieval* on encrypted information. Simply put, if the records of an employee file are stored encrypted, how does the client request to retrieve only the records of the engineers? This issue has led to solutions that are only partially successful. In a typical solution [8], the file is partitioned into "buckets", and each retrieval request is mapped (at the client's end) to a specification of buckets. Such an architecture has several weaknesses. It requires the predetermination of all columns for which selection is to be enabled (each column requires its own partitioning), it normally retrieves more records than requested, and it limits severely the selection comparisons that are feasible (for example, processing range queries is problematic).

In this paper we take a more general approach that regards information as an *association among values*. For instance, the information comprising the employee record (*Andrei*, *Engineer*, $75,000) is the three constituent values, as well as their mutual association. Consequently, hiding information is done both by hiding the values (e.g., by means of encryption) and by concealing the association. In our approach, the custodian might be able to observe data values (unless they are hidden by encryption), but would be denied knowledge of their association. In duality with encryption keys, the *cipher* for associating the disparate fragments would be available only to the client. Hence the objective of blind custodians is achieved by *information dissociation*.

The proposed architecture is described in Section 4 and preliminary discussion and assessment of this architecture are offered in Section 5. Section 6 concludes the paper with a brief summary and discussion of the considerable work that still remains to be done. We begin with a formal definition of the problem (Section 2), followed by a brief survey of related work (Section 3).

2 Formal Definition of the Problem

The general problem of a database service with privacy is formalized here abstractly. This abstraction allows us to position the works discussed in Section 3 in a unified framework.

Assume a database D and a query Q, and let $A = Q(D)$ denote the answer to the query Q in the database D. In a database service architecture, D is stored on the server, the client sends Q to the server, the server evaluates $A = Q(D)$ and returns A to the client.

Assume now that certain information in D must be kept confidential (it could be all or just part of D). A transformation T is the required, as follows:

1. T transforms the database to $D^* = T(D)$, and the query to $Q^* = T(Q)$.
2. Let $A^* = Q^*(D^*)$ denote the answer to Q^* in the transformed database D^*.
3. When the inverse transformation T^{-1} is applied to A^* it yields $T^{-1}(A^*) = A$.

In the database service architecture, the transformed database D^* is stored on the server. When the client needs to evaluate a query Q, it sends the server its transformed version Q^*. The server evaluates Q^* on D^* and sends back the answer A^*. The client then transforms A^* to A. The transformation T should hide the confidential information in D, and should be such that knowledge of A^* would not be sufficient to determine the answer A. This discussion is illustrated in the following diagram:

$$\begin{tikzcd}
D \arrow[r, "T"] \arrow[d, "Q"'] & D^* \arrow[d, "Q^*"] \\
A & A^* \arrow[l, "T^{-1}"']
\end{tikzcd}$$

This simple formulation may be overly "tidy" to be sufficiently general, and we consider here several more elaborate variations.

First, we may allow a transformation T that does not satisfy $T^{-1}(A^*) = A$, as long as there exists another query that can extract the answer A from $T^{-1}(A^*)$. That is, there exists a query Q' such that $Q'(T^{-1}(A^*)) = A$. Second, we may divide the transformed database D^* into two parts, one to be stored at the server, the other at the client. The query Q^* is performed on the server's part only, yielding A^*. The client receives A^* and applies T^{-1}

and Q' to both A^* and its part of the transformed database. Third, the computations done at the server and at the client may be *interleaved*, thus producing the final answer after a process of data exchange.

There are two trivial and impractical "solutions" to this problem:

- **Confidentiality without service**. In this solution, the transformation T is an encryption of the entire database, and the inverse transformation T^{-1} is simply decryption. Every query is translated to the trivial query Q^* that, when submitted to D^*, returns the entire database (i.e., $A^* = D^*$). The transformation T^{-1} converts A^* back to D and the additional query Q' is nothing more than the original query Q. In summary, the entire database is encrypted, and this encrypted database is retrieved in response to every query. Following decryption at the client, the query is processed locally. This "solution" obviously preserves the confidentiality of the information in the database, but is otherwise impractical, because no query processing is done on the server, and the client must have complete database management facilities. Indeed, the "server" does not deliver most of the functionalities or benefits of a Web service. Moreover, this solution requires the transmission of the entire database, an expensive, time-consuming operation.
- **Service without confidentiality**. At the opposite end of the spectrum, T is the identity transformation. Thus, $D^* = D$, $Q^* = Q$, $A^* = A$ (and Q' is unnecessary). In this case, all computations are performed at the server and only the exact answer A is transmitted back to the client. Obviously, this approach provides no confidentiality whatsoever.

These two extreme approaches demonstrate the tradeoffs involved: Complete confidentiality, but no service (all work is done at the client), *vs.* complete service (all work is done at the server) but no confidentiality. The works discussed in the following section are positioned in-between these two extremes because they choose smaller granularities for their encrypted transformation T. This allows some queries to be performed on the encrypted version D^* which is on the server.

3 Related Work

Work on information management services is quite extensive, with focus on issues such as availability, anywhere accessibility and reliability. More recently, efforts have concentrated on the need for a particular form of confidentiality [9, 8, 5, 6, 1], intended to protect clients' information from the service provider itself. These works discuss just two basic approaches to achieving confidentiality, both relying exclusively on encryption (either at the tuple level or at the field level).

The overall goal in [8] is to develop techniques for querying an encrypted database. The approach is restricted to integer-valued attributes and encryption is at the *field* level. If the encryption function is monotonic with respect to a predefined partial order on the plaintext database D, then queries Q involving ranges, comparisons, maximal value, minimal value, and so on, can be translated to queries Q^* on D^*, for which the answer A^* contains all (and sometimes, as in the case of maximal value, only) the tuples sought. More generally, encryption can be tailored to be a homomorphism with respect to a restricted set S of allowable queries, so that the result of a query Q in the allowable set S on the plaintext database D is obtained by decrypting the result of the encrypted query Q^* on the encrypted database D^*. An advantage of this approach is that for any query Q in S, only a single encrypted query needs to be evaluated on the encrypted database and its output needs no additional processing except for decryption. The problem, however, is to maintain the requirement that the encryption function be a homomorphism, while still allowing a reasonably broad set of queries. For an encryption to be a homomorphism, restrictions are placed on the possible closed-form (i.e., definable by an expression) or open-form (i.e., definable via an algorithm) encryption functions. Among them are order-preservation and distance-preservation. Although effective and efficient in its restricted domain, the approach is limited by the requested properties on the encryption functions.

A different approach is offered in [9], where encryption is at the *tuple* level, and an *indexing* scheme is used. If A is an attribute of R on which the client may need to issue selection queries, then the values of this attribute are partitioned, and every tuple of R is assigned to a single partition based on its A value. This scheme amounts to an *index* on attribute A. The mapping ϕ_A from the values of A to the partition identifiers is available only to the client, whereas the association between the partition identifiers and the encrypted tuples is known to the server. The latter association can be implemented as an extra attribute in the encrypted relation R, in which the values of ϕ_A will be stored. Each query Q on attribute A is converted to a query Q^* on the new attribute ϕ_A, and evaluated on D^*. The result A^* is returned to the client and decrypted. The initial query Q is then evaluated on the decrypted version of A^*. If Q selects the tuples of R for which the attribute A has value a, then Q^* returns all the encrypted tuples with the index value $\phi_A(a)$. By the very nature of indexing, a statistical query that is derived from the entire set of A values (e.g., the average value or the most frequent value) requires that *all* the tuples of R be retrieved from the server. Another drawback of this approach is that it requires a separate indexing scheme for each attribute on which selection queries are to be enabled; moreover, the selection attributes must be anticipated ahead of querying: A query on any attribute not indexed cannot be properly evaluated and the client would need to retrieve the entire relation. As each index requires an additional attribute on the server, indexing every attribute would double the size of the relation.

This approach is adopted and extended in [5], where different possibilities for representing indexing (partitioning) information are discussed. The authors propose to compute the indexing information so that it relates to the data well enough to provide an effective query execution mechanism, without releasing information about the relationship between indexes and data. Their solution is to base the indexing on direct encryption and hashing. They provide a detailed analysis on the inference exposure of the encrypted/indexed data. The paper also describes an enhancement of the indexing information that supports range queries.

The authors of [6] too investigate the approach of [9], which they term *aggregate-then-encrypt*. They propose a formalization to assess the security of privacy-preserving database outsourcing schemes, and apply it to analyze the scheme in [9]. They identify weaknesses and suggest a number of improvements to strengthen the approach. In addition, their paper contains an interesting result on the *impossibility* of achieving complete security by means of *privacy homomorphisms* (encryption functions that allow limited processing of the encrypted data), which is essentially the technique used in [8]. A review of several solutions to the problem of using the services of a provider without giving unnecessary access to sensitive data is included in [3].

The work in [2] concerns outsourcing scientific computations, where the external agent should not learn the actual data or the result of the computation. The client *disguises* the problem and data with local preprocessing before sending it to the agent, and obtains the true answer after further local post-processing of the result it receives. The major difference between that problem and the one addressed in this paper is that in [2] the client does not keep data permanently at the external agent. As in our approach, and unlike [10], the client's processing power is not limited to encryption and decryption.

The approach in [1] is to enforce privacy by replacing identifying information with values obtained through an anonymization process. It allows a choice of three basic properties of the anonymizing function: reversibility (by using encryption to allow the recovery of the original data if the appropriate key is known), irreversibility (by using one-way hash functions to prevent the recovery of the original information), and inversibility (pseudoanonymization, in Common Criteria terms, allowing the recovery by means of exceptional procedures to be applied by trusted parties only). The results of this anonymizing process are then used in different databases in place of the private information. Regardless of the anonymization objectives listed above, the "link" between random (anonymous) identities and actual identities is protected by encryption.

Two research areas that are somewhat related are privacy-preserving data mining [12] and secure multi-party computation [7, Chapter 7]. The blind custodian approach differs from these two areas in two important aspects: The secret data is not distributed among several parties (it could be physically distributed, but it is still controlled by a single client) and the client is expected to perform only minimal database work.

4 The Architecture

4.1 Information Dissociation

We have already observed that information is an *association among values*, and hence confidentiality may be achieved by hiding the association. The technique for hiding the *association*, which we term *information dissociation*, is discussed next.

Consider a database relation $R = (A_1, \ldots, A_m)$. Assume that it has been determined that while the tuples in this relation are confidential and should not be disclosed, subtuples with attributes $A_1, \ldots, A_i$ and subtuples with attributes $A_{i+1}, \ldots, A_m$ may be disclosed. For example, consider a table of employees with fields that concern employment (e.g., employee id, name, position and department), and fields that contain a home address (e.g., street, number, city, state and zip code; but not employee id). It may be considered acceptable to disclose each of these groups of fields separately, but not complete records (as they associate employees with their addresses).

Two views of R are defined (e.g., by appropriate SQL queries), and each of the resulting relations is augmented by an identifying field (an *index*) I_1 or I_2, thus obtaining new relations F_1 and F_2. The blind custodian is then entrusted with F_1 and F_2, while the client maintains the correspondence between I_1 and I_2 which is necessary to recover the complete tuples of the original relation R. The latter information is referred to as the *cipher* of the dissociation. In general, the number of fragment views may be arbitrary (i.e., not limited to two).

The database transformation $D^* = T(D)$ described in Section 2 is the decomposition of each of the database relations to a set of fragment relations and a cipher. This decomposition is somewhat reminiscent of other decompositions known from database theory, such as lossless-join decompositions or dependency-preserving decompositions in normalization theory [11, pages 392–412], or file fragmentation and replication in distributed database design [4, pages 67–92]. To satisfy the requirement that T hide confidential information this decomposition must be done judiciously. Presently, we observe two broad approaches towards this problem, one qualitative and one quantitative.

The qualitative approach uses external (subjective) judgment to determine that information, say, on an individual's employment, should not be associated with this individual's address. A simple way to annotate these constraints is to identify maximal sets of attributes that *may* be kept together (in the same fragment). Typically, several such safe fragments would be identified. From these, a set of fragments should be chosen that is both *consistent* and *complete*. Consistency guarantees that the set would not include two fragments that overlap on an attribute that is a key for at least one of the fragments, as this would allow the construction of a larger (unsafe) fragment. Completeness guarantees that the fragments in the set are sufficient to reconstruct the original relation (using the cipher).

A second possible way to dissociate a relation is to use objective (quantitative) criteria. The intuition here is that decomposing a relation into F_1 and F_2 is not very useful if R contains most of the tuples of the Cartesian product of the two fragments. If the original relation R has n tuples and is split into fragments F_1 and F_2 with n_1 and n_2 tuples, respectively, with no attributes in common, then the probability that a random tuple from F_1 and a random tuple from F_2 are related (form a tuple in the original relation R) is $p = n/(n_1 * n_2)$. When p is small (its lower bound is 0), then it is difficult to guess the associations among the subtuples of F_1 and F_2 that are valid. When p is high (its upper bound is 1), then a random tuple of F_1 is more likely to be associated with a random tuple of F_2. In the latter case, the value of decomposing R into F_1 and F_2 is rather low. Hence, decompositions with low p values should be preferred.

A combination of the two approaches would call for an initial dissociation based on subjective criteria, followed by a quantitative approach to choose among the resulting alternatives.

4.2 Query Evaluation

We now turn to the issue of evaluating client queries in this architecture. For simplicity, we assume that the database D consists of a single relation R, and that queries are selection-projection expressions; the generalization to multi-relation databases and other types of queries will be discussed later.

Assume that R has been decomposed to the fragments $F_1, \ldots, F_k$. Let I_i denote the index field that was added to the fragment F_i $(i = 1, \ldots, k)$, and let $C = (I_1, \ldots, I_k)$ denote the new cipher relation. $F_1, \ldots, F_k$ are stored at the server, and C is stored at the client. In terms of the formal problem defined in Section 2, this decomposition is the transformation T, and $F_1, \ldots, F_k$ and C make up the new database $D^* = T(D)$.

Consider a query Q on R submitted at a client. First, Q is transformed to a query Q^* on the server's database $F_1, \ldots, F_k$. The evaluation of this query on the server's database is returned to the client (this result is denoted A^*). The client then transforms this result to a new relation using its cipher C (this result is denoted $T^{-1}(A^*)$). To this, the client applies final processing (Q') to obtain the answer A. We observe at least two possible implementations of this process, each based on a well-known query optimization technique.

To illustrate the two techniques, we describe a simple example, in which information on employees is dissociated, so as to separate employment-related information from personal information:

$$F_1 = (I_1, \mathit{Eid}, \mathit{Ename}, \mathit{Salary}, \mathit{YearHired})$$
$$F_2 = (I_2, \mathit{Gender}, \mathit{Nationality}, \mathit{YearBorn})$$

Assume that the cardinality of R is 1,000, the cardinality of F_1 is also 1,000, but the cardinality of F_2 is only 750. Since the number of possible matchings among tuples of F_1 and F_2 is $1,000 \cdot 750 = 75,000$, and since only 1,000 of these are valid tuples of R, it follows that the probability that a random tuple from F_1 and a random of F_2 form a tuple of R is $1/750 = 0.00133$. We may assume that this probability is low enough to provide confidentiality (i.e., to thwart guessing).

Frugal Join. The relation R is substituted in Q by an expression that joins the cipher and the relevant fragments, and the query's selection and projection operations are "pushed" to the individual fragments, as practicable. The evaluation of the transformed query proceeds as follows. First, the server performs selections and projections on its fragments and sends the results to the client. The client then joins the results using its cipher and applies the final selection and projection (which could not be pushed to the fragments).

Consider now a query about the id's of female employees who earn over \$80,000 and have been in employment more than half their lives:

$Q:$
select *Eid*
from *Employee*
where $Salary > 80,000$ **and** $Gender = \text{'female'}$
  and $(2005 - YearHired) > 0.5 * (2005 - YearBorn)$

The server performs this two-part query Q^*:

$Q_1^*:$	$Q_2^*:$
select I_1, *Eid*, *YearHired*	**select** I_2, *YearBorn*
from F_1	**from** F_2
where $Salary > 80,000$	**where** $Gender = \text{'female'}$

It sends the results, denoted A_1^* and A_2^* respectively, to the client, who then concludes the processing with a query that joins the answers through its cipher and then extracts the final tuples that constitute the answer A:

$Q':$
select *Eid*
from A_1^*, A_2^*, C
where $A_1^*.I_1 = C.I_1$ **and** $A_2^*.I_2 = C.I_2$
  and $(2005 - YearHired) > 0.5 * (2005 - YearBorn)$

With respect to confidentiality, this process does not disclose to the server anything than it does not already know. From the request to deliver the two sets A_1^* and A_2^*, the server may be able to guess the client's query Q, but forming the two sets (something it could do all along, anyhow) does not increase its ability to match F_1 tuples with F_2 tuples. In other words, the probability of generating random R tuples through random guessing remains unchanged at 0.00133.

With respect to transmission costs, assume further that of the 1,000 employees 500 are females, 400 earn over \$80,000, and of the latter only 100 are females. The server sends the client 400 tuples of three fields each and 500 tuples of two fields each, for a total of 2,200 fields.

Semi-Join. Here, the joins among the fragments and the cipher are done in stages. Assume that m fragments are involved in the query, and let α_i denote the selection-projection query on the i'th fragment. The server begins by sending the client the result of performing α_1 on the first fragment. The client matches the id's of the tuples it received through its cipher, and sends the server the corresponding tuple id's for the second fragment. The server then performs α_2 on the second fragment after it has been pruned with the id's it received, and sends the result to the client. The client matches the id's of the tuples it received through its cipher, and sends the server the corresponding tuple id's for the third fragment. The process continues until the results from m'th fragment are sent to the client. The client then constructs the required answer from the data it has received. This version is often superior to the previous one because it reduces data transmission substantially (transmission can be further optimized by scheduling the order of fragments effectively).

In our two-fragment example, consider the same query. The server begins with a query on the first fragment

Q_1^* :
select I_1, *Eid*, *YearHired*
from F_1
where *Salary* $> 80,000$

and sends the result, denoted A_1^*, to the client. The client matches this information through its cipher

Q_1' :
select I_2
from A_1^*, C
where $A_1^*.I_1 = C.I_1$

and sends the resulting tuple id's, denoted A_1', back to the server. The server then performs

Q_2^* :
select I_2, *YearBorn*
from F_2, A_1'
where $F_2.I_2 = A_1'.I_2$
  and *Gender* $=$ 'female'

and sends the result, denoted A_2^*, to the client. The client concludes the processing with the query

Q_2' :
select *Eid*
from A_1^*, A_2^*, C
where $A_1^*.I_1 = C.I_1$ **and** $A_2^*.I_2 = C.I_2$
  and $(2005 - \mathit{YearHired}) > 0.5 * (2005 - \mathit{YearBorn})$

Assuming the cardinalities given above, transmission costs are reduced. The server sends the client 400 tuples of 3 fields each, the client then sends the server about 300 tuples of one field each,[1] and the server sends the client only 100 tuples of two fields each, for a total of only 1,700 fields.

However, with respect to confidentiality, this strategy discloses information to the server. The server delivered a set of I_1 values and received in return a matching set of I_2 values. In our example, the cardinality of these sets are 400 and 300, respectively. Hence, the probability of reconstructing an employee tuple in this subset of high-salaried employees is

[1]Since 1,000 employees share 750 personal records, we may assume that 400 employees share 300 personal records.

$400/(400 \cdot 300) = 0.00333$. Confidentiality has thus been reduced by a factor of 2.5. This reduction in confidentiality can be seen as a result of providing the server with information that was cycled through the cipher. Care must be taken when using the semi-join strategy, to assure that a desirable level of confidentiality is maintained.

5 Discussion

5.1 Measuring and Maintaining Protection Levels

How much protection does the blind custodian architecture provide? Essentially, the challenge for the server is to recover protected information by finding associations among the fragments. Assume a relation R is dissociated into two fragments F_1 and F_2. Let n, n_1 and n_2 denote the cardinalities of R, F_1 and F_2, respectively. As already suggested in the previous section, confidentiality is provided by having large enough cardinalities n_1 and n_2 and a relatively smaller cardinality n. Specifically, the number of possible associations between tuples of F_1 and tuples of F_2 is $n_1 \cdot n_2$, of which only n are valid. Consequently, the probability that a random matching of a tuple of F_1 with a tuple of F_2 will coincide with an actual tuple of R is $n/(n_1 \cdot n_2)$. We adopt this probability of disclosure as a measure of the protection afforded to the fragments F_1 and F_2 (or to R itself). It quantifies the ability to associate information from the two fragments, and thus gain knowledge of protected information. Note that lower probabilities indicate better protection. So that higher values indicate better protection, we define the *protection level* of F_1 and F_2 as $1 - n/(n_1 \cdot n_2)$.

Each pair of fragments has its own level of protection. Protection levels can also be defined for sets of fragments larger than two. Assume R is dissociated into fragments $F_1, \ldots, F_k$. The protection level of $F_1, \ldots, F_k$ is $1 - n/(n_1 \cdots n_k)$. This number reflects the ability to create complete tuples of R.[2] Protection levels can also be defined for fragment subsets that do not "cover" all of R. In such cases the numerator cardinality is the number of tuples in the projection of R that corresponds to the attributes in the fragment subset. A definition of protection level in the general case follows.

Assume a relation R is dissociated into fragments $F_1, \ldots, F_k$. Let $F_{i_1}, \ldots, F_{i_p}$ be a subset of the fragments, and let n_{i_j} be the cardinality of F_{i_j} ($j = 1, \ldots, p$). The protection level of $F_{i_1}, \ldots, F_{i_p}$ is defined to be $1 - n'/(n_{i_1} \cdots n_{i_p})$, where n' is the cardinality of the projection of R onto the attributes of $F_{i_1}, \ldots, F_{i_p}$.

The protection level required for each subset of fragments may be defined by setting threshold values during the dissociation process sketched in Section 4.1. Or one may simply adopt a single threshold for all the possible combinations of fragments.

These thresholds must be upheld during both the initial design and in subsequent query processing. During the initial design, it must be ensured that for every subset of fragments

[2]Note that it may be misleadingly high, as just associating a few of the fragments may be worrisome.

the protection level exceeds the threshold. During query processing, when the semi-join strategy is used, care must be taken not to exchange subsets with small cardinalities, as this may result in decreased protection levels (as was illustrated in the example).[3]

In both the initial design and subsequent query processing, whenever protection levels fall below the threshold, cardinalities may be artificially increased by adding *spurious tuples*, thus improving protection levels. (1) In the initial design, spurious tuples may be added to fragments as necessary. The id's of these "bogus" tuples must be kept on the client, to ensure that this information is not included in final answers. (2) During query processing, when the client receives a set of tuple id's from one fragment and responds with a corresponding set of tuple id's of another fragment, the outgoing set may be enlarged with additional tuple id's from the second fragment. These may be either "real" or "bogus" tuples; however, the client must log these additions, to ensure that they do not taint final answers. Clearly, the use of spurious tuples increases the cost of query processing.

5.2 Multi-relation Databases and Join Queries

The architecture we described assumed that the database has only one relation. However, the extension to several relations is relatively simple, and we sketch it here briefly. Each of the database relations is dissociated into a set a fragments and a cipher, and the client stores all the ciphers. However, extra care must be taken to protect key or foreign key relationships among the different relations, as necessary. For example, assume two database relations $\mathit{Employee} = (\underline{\mathit{Eid}}, \mathit{Ename}, \mathit{Salary})$ and $\mathit{Assignment} = (\underline{\mathit{Eid}, \mathit{Project}}, \mathit{Performance})$ with the requirement that performance information be kept separate from salary. If the two relations are considered individually, then it may appear that no dissociation is necessary; i.e., each relation will require a single fragment containing all its attributes. Yet, when considered together, the common attribute *Eid* allows performance and salary to be associated. A simple solution is to remove *Eid* from the relation in which it is a foreign key. Thus, the fragments would be $F_1 = (I_1, \mathit{Eid}, \mathit{Ename}, \mathit{Salary})$, and $F_2 = (I_2, \mathit{Project}, \mathit{Performance})$. This solution depends on the fact that all employee id's in *Assignment* appear in *Employee.*

Consider now database queries that involve joins among relations. The techniques described in Section 4.2 are applicable without significant changes. A query that requires a join $R_1 \bowtie R_2$ will be decomposed to a query that joins fragments of R_1, fragments of R_2, and a fragment of R_1 with a fragment of R_2. The joins may be based on either the frugal or the semi-join approaches.

5.3 Comparison

How does the blind custodians architecture compare with other methods? In Section 3 we discussed the two main alternatives: field-level encryption that uses encryption functions

[3]The frugal join strategy has no affect on the protection level.

with exacting properties [8], and a combination of tuple-level encryption with index-like structures [9]. Below, we briefly discuss the blind custodians architecture vis-a-vis these two alternatives

We observe two important performance criteria for a non-trusting database service: (1) the family of queries that can be processed should be as general as possible, and (2) the service provider should do as much of the work as possible, and the amount of data transmitted should be as low as possible.[4]

Clearly, a strong encryption function that is a homomorphism for general queries would provide an ideal solution, as it would fully satisfy both requirements. Note that because the encrypted answers sent to the client would be exact, all work (except for decryption) would done at the server, and data transmission would be minimal. Unfortunately, such ideal encryption functions are not available. Hence, the main disadvantage of the approach advocated in [8] is that it severely limits the generality of queries that can be processed. And if more general queries are attempted, then the burden of processing shifts drastically to the client.

A similar disadvantage is also apparent in the index-based architecture [9]. To increase query processing capabilities, indexing structures must be devised for most every attribute; and even then some queries (e.g., certain statistical queries) may require the entire set of tuples to be sent to the client. Consequently, clients must have substantial database management capabilities to process the tuples they receive.

In contrast, the blind custodian architecture has a clear advantage in the first of the two criteria, as it places no restrictions on the types of queries allowed. It is difficult to estimate its relative performance with respect to the second criteria, as it depends strongly on the *profile* of the queries submitted, and the *implementation* of the other architectures (i.e., the types of encryption functions adopted in the first approach, and the extent of indexing performed in the second approach).

How much work is done at a blind custodian client? It can be described as a “light” database management system. Among other tasks, it should be able to convert queries to appropriate execution plans, join relations through their ciphers, and apply final extractions. Except for the ciphers, data is only cached temporarily at the client, and only limited storage capabilities are therefore necessary. Of course, the client system does not need to manage functions such as backup, recovery, or transaction synchronization.

6 Conclusion

We outlined an architecture for a database service that provides confidentiality by means of information dissociation. The essential paradigm of our architecture — decompose the

[4]Note that the relative amount of work done at the server and the volume of data transmission are strongly related: More work accomplished at the server implies less data transmitted, and vice versa.

database to fragments and then transform queries on the original database to queries on the fragments — is similar to that of *distributed databases*, with a notable difference: The *motivation* for the decomposition is different. In distributed databases decomposition is dictated by requirements such as (1) data must be stored only in the computers of their owners, (2) data is preferably stored in computers that access them frequently, and (3) data could be replicated to provide redundancy and to reduce transmission; whereas here, the decomposition is motivated by the need to protect the data from the server, while letting the server store as much information as possible.

The discussion in this paper is only preliminary and many issues still have to be addressed in appropriate detail. Several of these research issues have already been given limited treatment earlier, and we mention here three additional issues,

Queries. We considered at some detail queries that are join-selection-projection expressions. Other important query operations include set operations (e.g., union or difference) and statistical functions (e.g., count or average). We conjecture that these operations can be accommodated in the architecture without requiring any modifications. Indeed, as the architecture is analogous to a distributed database, every query should be feasible, the only constraint being that its execution plan should maintain the requisite level of protection.

Protection. We analyzed protection levels under the naive assumption that no external knowledge is used in attempts to gain hidden information, and our protection analysis assumed *uniform* probability distribution functions. In various circumstances, external information available to the server may allow it to infer a non-uniform probability distribution function that is much closer to the actual function. For example, there may be 40 different values of *YearBorn* and 20 different values of *YearHired*, but of the 800 combinations, some combinations may be known to have probabilities that are much higher than those of other combinations. Such knowledge may lower substantially the protection level. Additionally, the cardinalities of some domains may be misleadingly high. For example, there may be 3,000 different salary values in the database, yet for practical purposes one may consider similar all salaries that round to the same $1,000, resulting in a much smaller number of "significantly different" values. These and other issues require a more elaborate analysis of protection levels.

Encryption. A basic feature of the blind custodians architecture is that it does not involve encryption. The architecture attempts to protect relationships, while assuming that there is no harm in disclosing the values in the database. Yet there may be circumstances in which even the domain values should not be made public. This could be achieved by substituting the domain values with identifiers and associating the identifiers with the actual values by means of a new client relation. This solution is not attractive because it increases the storage requirements on clients beyond the essential ciphers. Alternatively, we could use field-level encryption to hide values when necessary.

References

[1] A. Abou El Kalam, Y. Deswarte, G. Trouessin, and E. Cordonnier. A generic approach for healthcare data anonymization. In *Proceedings of WPES 04, the 2004 ACM Workshop on Privacy in the Electronic Society*, pp. 31–32, 2004.

[2] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. H. Spafford. *Secure Outsourcing of Scientific Computations*, Volume 54 in Advances in Computers, pp. 215–272. Elsevier, 2001.

[3] C. Boyens and O. Gunther. Trust is not enough: Privacy and security in ASP and Web service environment. In *Proceedings of ADBIS 02, Advances in Database and Information Systems*, LNCS No. 2435, pp. 8–22. Springer, 2002.

[4] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill, 1984.

[5] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proceedings of the 10th ACM Conference on Computer and Communication Security*, pp. 93–102, 2003.

[6] M. Fischmann and O. Gunther. Privacy tradeoffs in database service architectures. In *Proceedings of BIZSEC 03, the First ACM Workshop on Business Driven Security Engineering*, 2003.

[7] O. Goldreich. *Foundations of Cryptography, Volume II: Basic Applications.* Cambridge University Press, 2004.

[8] G. Ozsoyoglu, D. A. Singer, and S. S. Chung. Anti-tamper databases: Querying encrypted databases. In *Proceedings of the 17th Annual IFIP WG11.3 Working Conference on Database and Application Security*, 2003.

[9] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings SIGMOD 02, International Conference on Management of Data*, pp. 216–227, 2002.

[10] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On databanks and privacy homomorphisms. In R. D. DeMillo, editor, *Foundations of Secure Computations*, pp. 169–177. Academic Press, 1978.

[11] J. D. Ullman. *Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[12] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Proceedings of CRYPTO 00, 20th Annual International Cryptology Conference*, LNCS No. 1880, pp. 36–54. Springer, 2000.