Anti-Tamper Databases: Querying Encrypted Databases

November 29, 2001

Abstract

With mobile computing and powerful laptops, databases with sensitive data can effectively travel everywhere. They can also be physically retrieved by malicious users who can employ techniques that were not previously thought of, such as disk scans, compromising the data by bypassing the database management system software or database user authentication processes.

A way to prevent, delay, or contain the compromise of the protected data in a database is to encrypt the data and the database schema, and yet allow queries and transactions over the encrypted data. Such encrypting may take the forms of low/medium/high security in response to users’ security needs as well as database access needs in terms of the efficiency of database accesses for queries and transactions. Clearly, there is a compromise between the degree of security provided by encryption and the efficient querying and accessing of the database. In this research, we propose to investigate the capabilities and limitations of encrypting the database data and/or the database schema in relational databases, and yet allowing, to the extent possible, efficient querying of the encrypted database.

TBD
Project Description

1 Introduction

Databases are complex software systems with advanced query languages allowing users to easily access and manipulate data. Towards this goal, database system software (DBMS) has two components:

(a) A data model, such as the relational data model [RG01] or the object-relational data model [RG01], which captures the semantics of the data in the database. The data model has “data” (e.g., employee name “john doe”) and schema which is “metadata”, i.e., data about data (e.g., table and attribute names in the relational data model). An integral component of the data model is the integrity constraints in the form of functional dependencies [Ullm89], referential integrity constraints, foreign key constraints, and general-purpose integrity constraints [RG01].

(b) A query language which allows arbitrary querying of the database, such as the SQL query language of relational and object-relational databases.

Commercial databases (e.g., ORACLE 8i [OR01] or the SQLServer [SQLS01]) are ubiquitous, widely used everywhere, and have advanced query engines. In the rest of this proposal, we concentrate only on relational databases and, to a lesser degree, object-relational databases even though other types of databases, namely, object-oriented databases employing the OQL query language are also in use.

With mobile computing and powerful laptops, databases with sensitive data can effectively travel everywhere. They can also be physically retrieved by malicious users, referred to as the adversary in this proposal. Present database security models [Pfle97, Den82] do not assume such possibilities, and need to be expanded. The two main security mechanisms that general-purpose commercially-available database management systems (DBMSs) employ are (a) user authentication, and (b) access control mechanism (e.g., read, write, grant, or revoke privileges, etc.) to data in the database. In today’s environments, we think that both of these mechanisms may fail, especially if the computer that hosts the database is physically captured by the adversary. Such a possibility allows the adversary to do a multitude of things that were not thought of before, such as scanning hard disks for data and/or passwords, or becoming a super user and bypassing the user authentication of a database, etc.

We believe that a good way to prevent, delay, or contain the compromise of the protected data in a database is to encrypt the database data and the database schema, and yet allow (a) queries over the encrypted data, and (b) the integrity constraint enforcement, i.e., when the encrypted database is completely decrypted, original integrity constraints are satisfied.

Clearly, the requirement of high level of security for the encrypted data conflicts with the requirements of query processing (let alone query optimization) and integrity enforcement over encrypted data. Thus, achieving high levels of security by encryption is not always possible; the encryption may take the forms of low/medium/high security, coupled with fast/medium/slow query processing/integrity enforcement in response to the users’ needs for

- Security of the anti-tamper database,
- Expressive power of the anti-tamper database queries and query processing times, and
- Expressive power of the anti-tamper database integrity constraints and integrity enforcement times.

In this research, we propose to investigate techniques for encrypting the database data and/or database schema, and the accompanying techniques to query the encrypted database and to enforce integrity constraints on the encrypted database, which we refer to as the anti-tamper database. The site (computer) in which the anti-tamper database resides will be called as the anti-tamper site. We make the following assumptions about the anti-tamper database:
The encrypting of the original database is transparent and unknown to the legitimate users of the database. That is, there is no extra burden placed on the legitimate users of the anti-tamper database.

As requested by federal laws (the Privacy Act of 1974), we assume that the general encryption mechanism is made available to the public, including the adversary. However, certain parameters of the encryption mechanism (e.g., private keys) are kept secret so as to make the protected information in the database secure.

We assume that the commercial DBMS system does not know that its data in the database is encrypted. This assumption is for convenience in that commercial DBMSs are complex proprietary software systems, and any security technique that is deployable without modifying a DBMS is more desirable over one that requires changes in the DBMS.

We employ the following computing architecture for our computing environment. Let the original database DB be encrypted into the anti-tamper database DBMOD (the “modified” database). Now, if the database DBMOD directly becomes available to the adversary, its contents are devoid of semantics and meaningless, and, therefore not directly usable by the adversary. We employ intermediary software, called the (Encryption/Decryption) Agent, which we assume to be secure. That is, the adversary cannot capture the Agent software code, and reverse-engineer its encryption/decryption algorithms. We will investigate two alternative architectures:

- The agent resides at a site different than the site of the anti-tamper database DBMOD, and has significant computational power and storage space. We refer to this site as a secure site, and to the Agent as a Secure-Site Agent. There may also be a secure DBMS at the secure site (to be employed in our procedures). We refer to this DBMS as DBMSSECURE.
- The agent resides at the site of the anti-tamper database, and has little computational power. We refer to this Agent as the Same-Site Agent.

For both architectures, user queries are processed as follows:

a. The user forms a query Q(DB) against the original database DB, and submits it to the Agent

b. The Agent rewrites the original query Q(DB) into either

   i. A single query Q(DBMOD) against the encrypted database DBMOD or

   ii. A set Qs of k different, k≥1, queries Q1(DBMOD), …, Qk(DBMOD), and submits Q(DBMOD) or the query set Qs, respectively, to the DBMS of the anti-tamper database, referred to as DBMSMOD.

c. The DBMSMOD processes the query Q(DBMOD) or the query set Qs, and returns the output O(DBMOD) or the output set Os, respectively, to the Agent.

d. In the case of a single output O(DBMOD), the Agent decrypts the output O(DBMOD) into O(DB), the legitimate output of the original query Q against the original database DB, and returns it to the user. In the case of the multiple output set Os, the Agent decrypts each output in the output set, performs additional computations with the decrypted output set to obtain the answer, and returns the answer to the user.

We give an example.

**Example 1.** Consider a relational employee database with the relation EMPLOYEE (Id, Salary) where Id and Salary are integer-valued employee id and employee salary attributes. Assume that f() and g(), functions from integers to integers with inverses f^(-1)() and g^(-1)() respectively, are used to encrypt employee salaries and employee ids, and the relation EMPLOYEE and the attribute names Id and Salary are encoded as the characters R, A, and B. Then the SQL query
Q(DB): \[ \text{SELECT EMPLOYEE.Id FROM EMPLOYEE WHERE Salary=a} \]
which returns the employees with salary a, is rewritten by the Agent into the SQL query
Q(DBMOD): \[ \text{SELECT R.A FROM R WHERE B=f(a)} \]
and submitted to the DBMSMOD to be executed against the database DBMOD. Assume that the output O(DBMOD) of
the query Q(DBMOD) is a single tuple with value y. Then the value y is decoded by the Agent as \( g^{-1}(y) \), and returned
as the output O(DB) of the query Q(DB).

We use the notion of key from cryptology for both functions \( f() \) and \( g() \) as follows: The function, say, \( f(x) \) and
its inverse \( f^{-1}(x) \) are in the form \( E(K, x) \) and \( D(K, x) \) where \( K \) is the secret key, and \( D(K, E(K, x)) = x \). That is, \( D \)
decrypts \( x \) encrypted by \( E \) using the same key \( K \).

We refer to the process of transforming the SQL query of the original database (i.e., Q(DB)) into
the SQL query of the encrypted database (i.e., Q(DBMOD)) as the \textit{query rewriting process}. 

Note that the \textit{user site} in this architecture can be the secure site, the anti-tamper site, or yet another
third (secure) site. If the user site is the third site, we assume that there is a secure channel of
communication with the Agent site and the user site. If the user and the Agent are both at the anti-tamper
site then the output of a query returned to the user can be compromised (until it is destroyed by the user)
if captured by the adversary.

Our approach to query processing, as described above, can be characterized as \textit{“no decrypting at
the anti-tamper site”}. For feasibility considerations, there is another alternative which uses private-key-
based decryption and user-defined procedures \([HS93, H94]\) at the anti-tamper site, illustrated with an
example.

**Example 2.** Assume that the EMPLOYEE relation has the integer-valued attribute ProjId, which specifies the
current project that the employee is working on. Attribute ProjId is encrypted using the encryption function \( E(K, x) \)
where \( x \) takes its value from ProjId attribute values, and the key \( K \) at the Agent is such that \( D(K, E(K, x)) = x \) where
\( D() \) is the decryption function. Assume that the user u poses the SQL query
Q(DB): \[ \text{SELECT EMPLOYEE.Id FROM EMPLOYEE WHERE Salary > 100000 AND (EMPLOYEE.ProjId /2)*2 = EMPLOYEE.ProjId} \]
which returns the employees who make more than $100K and work in projects with even-numbered project ids.
Assume that this query cannot be rewritten into a single anti-tamper database query because the encryption function
\( E(K, x) \) is such that there is no way of rewriting the predicate
\( (\text{EMPLOYEE.ProjId} /2)*2 = \text{EMPLOYEE.ProjId} \)
into a predicate for DBMOD. However, at query evaluation time, ProjId can be decrypted using (a transformed
version of ) the key \( K \). Assume that the relation EMPLOYEE and the attribute names Id, Salary, and ProjId are
encoded as the characters R, A, B, and F; and \( f() \) is the encryption function for the Salary attribute. Then
(1) the Agent first splits the query Q(DB) into two queries:
Q1(DB): \[ \text{SELECT EMPLOYEE.Id FROM EMPLOYEE WHERE Salary > 100000} \]
and
Q2(DB): \[ \text{SELECT EMPLOYEE.Id FROM EMPLOYEE WHERE (EMPLOYEE.ProjId /2)*2 = EMPLOYEE.ProjId} \]
(2) The agent rewrites Q1(DB) and Q2(DB) as
Q1(DBMOD): \[ \text{SELECT R.A FROM R WHERE B > f(1000)} \]
Q2(DBMOD): \[ \text{SELECT R.A FROM R WHERE (Decrypt(V(K), R.F)/2)*2=Decrypt(V(K), R.F)} \]
and sends both queries to the anti-tamper database (DBMSMOD) for evaluation.
Here Decrypt is a user-defined procedure stored at DBMSMOD which implements the decrypting function D( ). The value of V(K), sent by the agent, is sufficient for the user-defined procedure Decrypt() to decrypt a given encrypted ProjId value. V(K) is retained at the anti-tamper site during the evaluation of Q2(DBMOD) for Decrypt() to repetitively decrypt the encrypted ProjId values. That is, the evaluation of Q2(DBMOD) takes place by first decrypting each R.F value as ProjId at the anti-tamper site, and then evaluating the SQL predicate \((\text{ProjId} / 2)^2 = \text{ProjId}\).

3 The agent then collects the outputs \(O_1(\text{DBMOD})\) and \(O_2(\text{DBMOD})\) of queries \(Q_1(\text{DBMOD})\) and \(Q_2(\text{DBMOD})\), respectively; decrypts \(O_1(\text{DBMOD})\) into \(O_1(\text{DB})\), takes the union of \(O_1(\text{DB})\) and \(O_2(\text{DBMOD})\), and sends the result to the user as the answer to the original query \(Q(\text{DB})\).

The approach of Example 2, while it works, has problems. First, if the adversary captures the anti-tamper database while the query \(Q_2(\text{DBMOD})\) is in the process of being evaluated, V(K) will be compromised and all ProjId attribute values will be compromised. However, to the anti-tamper database software DBMSMOD, V(K) is a temporary value employed only during query processing, and, assuming that DBMSMOD is instructed to destroy all temporary data related to a query after the query is processed, the anti-tamper database is secure—except for the time window of processing the query \(Q_2(\text{DBMOD})\). The second problem is, this approach is potentially much more costly in time as (a) extra decrypting takes place at the anti-tamper database, and (b) the Agent needs to have high computational power and, possibly, secondary storage capabilities since it needs to take a union operation of possibly two very large data sets.

Assume that the Agent is located at the secure site, and DBMSSECURE is available at the secure site as well. If the problems of the approach of Example 2 are not acceptable, there is always the base approach where the agent requests using the query \(Q_3(\text{DBMOD})\) a complete copy of R to the secure site, decrypts and stores it into DBMSSECURE, and runs \(Q_2(\text{DB})\) against DBMSSECURE. However, this approach, while secure, is even more time-costly than the approach of Example 2 as all of R is shipped to the secure site and then stored to another database. Nevertheless, this approach forms the base approach against which the costs of all approaches need to be compared in order to evaluate how much time improvement they provide with respect to the base approach.

Let us briefly review the database integrity constraint enforcement issues for the encrypted data. First, retaining in the anti-tamper database referential integrity constraints and foreign keys of the original database translates into using the same encryption function for values appearing in different attributes of different relations, which clearly reduces the level of security. Second, retaining in the anti-tamper database the functional dependencies of the original database is more serious in that the encryption technique is forced into retaining the properties of functional mappings (e.g., one-to-one, onto, total, etc.) as defined by functional dependencies. Third, general-purpose integrity constraints (for relational databases) are 1st order predicate calculus formulas that must always evaluate to true when instantiated with the existing tuples of relations. Therefore, for the anti-tamper database, satisfying an integrity constraint of the original database translates into satisfying the “rewritten” 1st order predicate calculus formula for the anti-tamper database, which in turn means that the rewriting of the formula for DBMSMOD must exist.

The rest of the proposal is organized as follows. In section 2, we briefly review results from prior NSF support. In Section 3, we survey the related literature. Section 4 discusses the research issues in anti-tamper databases. Section 5 is the proposed research plan.

---

1 V(K) is a transformed form of the key K so that K cannot be compromised during its transfer to and its use at the anti-tamper site. And, V(K) is “uni-directional” (i.e., V(K) maps to multiple keys) so that compromising V(K) does not lead to the compromise of K.

2 This is the case for commercial databases if the user requests that all cache contents are cleared after every query evaluation.
2 Results from Recent NSF-supported work

In *Constructing and Manipulating Presentations in Multimedia Databases* (NSF Award IRI-96-31214; 8/96-8/1999), we incorporated multimedia presentations (modeled as presentation graphs) into multimedia database systems. We developed three different techniques for novice, intermediate, and advanced users of multimedia, and a distributed multimedia database system with (a) Automated Presentation Assembler, called VIDQUERY [OHK96, HakOz96, HakO97, HKO99, Hakk97, Ramach98, Parekh98, Yang99, Eric00, Dennis00], (b) Presentation Player for presentation construction, storage, and delivery from local/remote servers, and (c) A distributed multimedia storage manager and server, VStore, designed for serving presentations over a local-area network. We designed, implemented, and evaluated two multimedia presentation retrieval and manipulation languages: GOQL [SOO99] and GVISUAL [LSB99, LSBAOO00, LeeO98, LSBOO96, LO96], to query and manipulate multimedia presentation graphs with respect to content information. The multimedia system ViSiOn, which supports the storage, delivery, querying, manipulation and playout of presentation graphs, was built [Cao98, Touma98, Reng97, Ortega97, Yao97, AIH98, Lee98, Cai98, Lei00]. We have designed and implemented a buffer manager to serve presentation graphs and also to investigate efficient algorithms [BO98, BO99] for the admission control of presentation graphs. We devised an efficient index structure (for images), which is used for large metric spaces [BO99, BO97]. In [O+a00, O+b00], we introduce an electronic book data model containing (a) topic objects and (b) instructional resources, called instruction module objects, which are multimedia presentations possibly capturing real-life lectures of instructors. The electronic book automatically constructs the “best” user-tailored lesson (as a multimedia presentation) for users.

This grant graduated five PhD (one female) students and 19 MS students. The publications include papers in IEEE TKDE (5 papers), ACM TODS (1 paper), VLDB Journal (3 papers), ACM Multimedia Systems Journal (2 papers), Information Sciences (1 paper), and J. of Multimedia Tools and Applications (2 papers).

Add the results from the current NSF grant.

3 Related Work in Database Security and Cryptology

A cryptographic strategy must of necessity involve a trade-off between security and functionality. At one extreme, security may be achieved by complete encryption of the data file. The client begins a session by entering a pass-phrase; then the data is decrypted and the client works with the data. At the conclusion the data is re-encrypted. Even this model does not provide absolute security, since the data is vulnerable during the period in which it is decrypted. The client must end each session with the re-encryption step. To the extent that this is onerous there may be the temptation to leave the file decrypted.

A second approach is to employ record-based encryption. Here each record is encrypted as a unit and only a particular record or set of records need be decrypted at any given time. However, this technique makes search queries impossible without decrypting all records.

A third approach is field-level encryption [Den85]. This offers a lower level of security, since it allows ciphertext searching, but it adds considerably to the ease of use of the data. Techniques which provide multiple-valued encryption while preserving unique decryption, can be used to thwart such searches. In other words, the same value appearing in the same field of two different records should be encrypted differently. Then if the contents of one record are compromised no inferences can be drawn about the content of other records. Denning achieves this in [Den85] by deriving different encryption keys for each data element from a single master key.

A fourth approach is to use an algorithm for encrypting data which always encrypts identical values in
corresponding fields of distinct records in the same way, thus deliberately violating the security prescription described above. This clearly increases the vulnerability of the data to cryptanalysis. On the other hand, it now allows for the possibility of certain search queries being performed without decrypting the data. Specifically, we can perform a query of the type described in example 1. Note that it is certainly possible to use this approach only for selected fields in which the utility of searching without decryption outweighs the loss to security. Furthermore, the correct choice of encryption method should insure that if the entries in one record are compromised, only those records whose corresponding field values agree with those of the compromised record will be (partially) compromised.

If, however, we wish to perform other queries, such as the query described in example 2, it is necessary to preserve the order structure of the data. Thus, we propose to develop encryption techniques which are order-preserving. Such encryption falls squarely outside the usual bounds of secure cryptographic techniques, since the ability to perform inequality queries must reveal information about the ciphertext. Nevertheless, this weak form of encryption may provide sufficient security for some applications, while significantly increasing the usability of the data.

4 Proposed Research

This research has four different components: (i) encryption techniques that retain data semantics, (ii) inference control (information leakage) analysis, (iii) query rewriting techniques for nested queries and aggregation operations, and (iv) efficient query processing with encrypted data.

4.1 Encryption Techniques That Retain Data Semantics

We will design and evaluate encryption techniques for database data of different data types, that allow query rewriting on the encrypted data, and yet provide provable security guarantees. All of the encryption techniques to be developed will be cognizant of

(1) Query expressive power and integrity constraint specification needs of the database, and

(2) Feasibility and efficiency of query processing and integrity enforcement over encrypted values.

Clearly, there is a compromise between the above-listed “database requirements” and the level of security provided by the encryption technique. And, the database requirements will sometimes dictate encryption functions that provide low levels of security in that, with small amounts of a priori knowledge, the users may be able to break the encryption. We view these cases as valid alternatives for database owners who do not wish to compromise the query expressive power or the query processing efficiency and effectiveness, but still would like to have an added, albeit low, degree of security. We discuss query processing and integrity enforcement needs (item (2) above) in section 4.4. Next, we discuss item (1), i.e., query expressive power and integrity constraint specification needs, and their implications on the encryption to be employed.

SQL, the industry standard for relational database query languages, is a declarative language with the expressive power (of a safe version [Ullm89]) of relational (first-order predicate) calculus. SQL employs in its query specification the characteristics of the data, as defined by its data type. As an example, with data of type integer and real, queries can contain arithmetic comparison operators; with data of type string, queries can contain string comparison and substring selection operators; and, with graph data of (user-defined abstract data) type graph, queries can contain, say, the (user-defined method

---

3 Relational calculus can specify queries with infinite output and/or infinite evaluation times [GT87, OW89]. Safety requirement eliminates such queries.

4 SQL of relational databases is actually more expressive due to aggregate functions and other features such as constraint specification. SQL’99 of object-relational databases is even more powerful with high-order calculus equivalency and recursive queries.
of ) graph containment operator. Therefore, the encryption technique to be designed, in addition to being secure, must take into account the data type T of the data being encrypted so that the operators available to T will also be available—in a transformed form—in the data type of the encrypted data. This means that, to give an example, for spatial database data, if a 3-D object O is being encrypted, for the query “rotate O around axis X by Y degrees” to be rewritable to the encrypted version of O, the operator “rotate” should also be available for the encrypted object.

The question then becomes “for the available data types (and their operators) in relational and object-relational databases, how can an encryption technique retain the original data semantics of the encrypted data?” We list the available data types in databases:

1) Primitive data types, i.e., integers, reals, and strings (relational databases).
2) Structured (also called, collection) data types such as sets, bags, sequences, arrays, graphs, etc., each with their own operators (object-relational databases).
3) User-defined (abstract) data types (ADTs) and user-defined methods as operators of ADTs (object-relational databases) such as (a) jpeg with crop, rotate, etc., or (b) mpeg with get-frame, etc. (object-relational databases).

In addition, data types form other semantically meaningful structures such as inheritance hierarchies with their own requirements; and, the encryption must be cognizant of the data type involvement in an inheritance hierarchy.

Next, we illustrate with two examples the issues with the primitive data types integers, reals, and strings and arithmetic comparison operators <, >, ≤, and ≥ as well as the arithmetic expressions on integers and reals.

Arithmetic comparison operators <, >, ≥, and ≤ (but, not = and ≠) of primitive data types depend on the total ordering of attribute values. For attributes of these types, if the operators <, >, ≥, and ≤ are to be employed in a query on the encrypted data, the encryption technique must “preserve” the ordering of attribute values.

Example 3. Consider the Salary attribute of the EMPLOYEE relation. The SQL query

```
SELECT * FROM EMPLOYEE WHERE Salary > 100000
```

returns those employees making more than $100K annually. For the Agent to rewrite this query as

```
SELECT * FROM R WHERE B > f(100000)
```

the Salary attribute should be encrypted using an encryption function that preserves the ordering of Salary values.

Def’n (Order-Preserving Encryption). Consider attribute A of relation R, and the encryption function f() for A values. The encryption function f() is order-preserving if when a > b for any two A values a and b in the original database DB then f(a) > f(b) in the anti-tamper database DBMOD.

In other words, the encryption of attribute A retains the ordering in A. Order-preserving encryption brings a new dimension to encrypting attributes of type integer, real, or string.

Integer and real attributes can also participate in arbitrary arithmetic expressions of SQL queries. Let us consider one of the simplest arithmetic expressions, namely, |a – b|.

Example 4. Assume that the relation EMPLOYEE has the attribute Age that lists the age of an employee, which is encrypted as attribute C using the encryption function f(). Consider a query where the user wants to know the names of pairs of employees who have an age difference of 10 or more years, which is expressed in SQL as

```
SELECT E.id, T.Id FROM EMPLOYEE E, EMPLOYEE T WHERE (E.Age – T.Age) > 10
```

When the encryption function f() is distance-preserving, this query can be rewritten as

```
SELECT E.A, T.A FROM R E, R T WHERE r. (E.C – T.C) > 10
```
where \( f(x) = rx + c \).

**Def’n (Distance-Preserving Encryption).** The encryption function \( f() \) is *distance-preserving* if, for any two attribute \( A \) values \( a \) and \( b \) where \( |a - b| = k \), we have \( |f(a) - f(b)| = r.K \), where \( r \) is a constant and \( |.| \) is absolute value.

In general, an encryption function \( f(x) \) that is distance-preserving for integers and reals has to be affine, i.e., \( f(x) = Ax + c \), where \( c \) is a constant and \( A = r \) if \( x \) is a scalar, and, more generally, \( A \) is a multiple of an orthogonal transformation when \( x \) is a vector attribute. Of course, such a transformation affords a very small level of security.

In this project, starting with primitive data types and the corresponding operators available in SQL and continuing with the data types in (2) and (3) above, in the given order, we will investigate encryption techniques that will allow efficient rewriting of queries, and yet are provable secure.

### 4.1.1 Open-Form Encryption Based on a Secret Key

In example 4, we have used \( f(x) = Ax + c \) which is a *closed-form encryption function* with a closed-form inverse. We may not always have a closed-form inverse \( f^{-1}() \) for the encryption function \( f() \), in which case we refer to such encryption functions as *open-form encryption functions*. That is, for encrypting the original database \( DB \) initially, and for decrypting query outputs as well as for intermediate processing by the agent (see section 4.3), the system uses a decryption algorithm (instead of a closed-form inverse function), possibly employing a secondary-storage-based data structure (such as a B+ tree [RG00]) so that, given the encrypted value \( v \) in \( DBMOD \), the decryption algorithm locates the original value \( f^{-1}(v) \) of the original database \( DB \). Next, we discuss order-preserving and open-form encryption based on a secret key, for integer attributes and for fixed-precision real attributes.

#### A. Integer Attributes

Consider an integer-valued attribute with values \( X \), to be encrypted using order-preserving encryption. Let us assume that the encryption function \( f(X) \) and its inverse are in the form of \( E(K,X) \) and \( D(K,Y) \), respectively, where \( K \) is the secret key. We would like to define a family of functions \( Y=E(K,X) \) and \( X=D(K,Y) \), where \( X,Y \), and \( K \) are nonnegative integers:

1. \( K \) is the secret key. Given \( K \), \( E \) and \( D \) should be efficiently computable.
2. For all \( X, Y, K \), we have \( D(K, E(K,X)) = X \). That is, \( D \) decrypts any number \( X \) encrypted by \( E \) using the same key. We will write \( E_k(X) \) for \( E(K,X) \) and \( D_k(Y) \) for \( D(K,Y) \); then we want \( D_k(E_k(X)) = X \).
3. It should be hard (in some sense) to find \( X \) from \( Y \) or \( Y \) from \( X \) in the absence of knowledge of \( K \), even assuming complete knowledge of the functions \( E \) and \( D \).
4. If \( X < X' \) then \( E(K,X) < E(K,X') \) for any \( K \). (Order-preservation)

Now let's suppose that the domain for our function \( E_K \), is the integers from 1 to \( N \), and the range is the integers from 1 to \( M \). This is not necessary but it will be convenient for the description of the system, and any other set of values can be converted to this by an encoding formula. For fixed \( K \), let \( y_n = E_k(n) \) for \( 1 \leq n \leq N \). Define a new sequence \( z_n \) by

\[
z_1 = y_1 - 1, \quad z_{n+1} = y_{n+1} - y_n - 1.
\]

In other words, the \( z_i \)'s are the differences (minus 1) between successive values of the encryption function \( E_K(n) \). Then condition 4 above is equivalent to the statement

\[
z_n \geq 0 \quad \text{for all } n
\]

If we have any order-preserving function we can define the corresponding sequence \( z_n \); and the converse is also true: given a sequence \( z_1, z_2, \ldots, z_n \) of nonnegative integers, there is a uniquely determined order-
preserving function for which \( z_i \)'s are the differences. Furthermore, we have the following algorithms for computing the encryption function \( E_K(n) \) and its inverse \( D_K(y_n) \):

**Encryption:**
\[
E_K(1) := 1 + z_1; \quad E_K(n+1) := E_K(n) + 1 + z_{n+1};
\]

**Decryption:**
Input: \( Y \)
Output: \( D_K(Y) \)
begin
\( i := 0; \quad W := Y; \)
while \( W > i \) do begin
\( i := i + 1; \quad W := W - z_i \) endwhile;
if \( W = i \) then return \( D_K(Y) := i \) else output "Failure";
end.

Our goal is to construct a family of functions indexed by a key \( K \), which contain as little information as possible. One way to achieve this is to generate a pseudorandom sequence \( z_i \) depending on an initial seed \( K \), and then use the algorithms above to define the encryption and decryption functions. The practicality of this scheme depends on the time it takes to execute the algorithms. Apart from the sequence generation, the only operation is addition, so the above decryption algorithm is feasible at least for reasonably small values of \( N \). For our running example such as salary, where if the salary is in hundreds, we can assume \( N < 20000 \), this would be simple to implement. For large \( N \) values such as \( N = 10^{12} \), one can use secondary storage-based indexing structures such as B+ trees, through which the decryption at the agent can be done fast. We will derive cost formulas for agent-based decryption employing indexing structures. Finally, there are plenty of well-known algorithms for generating pseudorandom integers efficiently which have known security properties.

The approach summarized in this section is a straightforward way to design a scheme whose security can be determined. There is of course a lot of work that needs to be done. It is clear that any order-preserving function leaks information, so the question is how much information is leaked. This will depend on the distribution of the \( z_i \). We may use \( z_i \)'s that are identically distributed in some range, but this does not appear optimal. For example, it might be better if each \( z_i \) is uniformly chosen from an interval depending on the sum of the previous \( z_i \). Here is one such algorithm:

1) Generate a sequence of random integers \( y_i \) in the range \([1, M]\).
   Assume that we have a family of pseudorandom functions \( R[K,n] = x_n \), where \( K \) is the secret key and \( n = 1, 2, 3, \ldots \). Typically this generates a stream of random bits, which can then be used to produce random integers; see Chapter 5 of the Handbook of Applied Cryptography [Ref].
   There are a large number of pseudorandom number generators with useful security properties. See, for example, M.
   Blum and S. Micali [BM84] for an early example. Here, the key serves as the initiator of the sequence.

2) Define \( z_i \) by the rules:
   \[
   S_1 := z_1 := y_1; \quad \text{for } k \geq 1 \text{ do begin } A_k := M - S_k; \quad z_{k+1} := \text{Int} \left[ y_{k+1} \cdot A_k / M \right]; \quad \text{endfor} \quad S_{k+1} := S_k + z_{k+1};
   \]
   The encryption function is then defined by \( f(k) = S_k \).

   For distance-preserving encryption, \( z_i \)'s are identically distributed in some range, but this may leak more information, and needs to be investigated.

**B. Real Attributes**
The above-summarized encryption cannot be directly extended to real-valued attributes. As an example, consider a real-valued attribute \( B \) with every \( B \) value having \( p \) fractional bits (i.e., fixed precision) and with magnitude values up to \( 2^{64} \) (i.e., two words). Converting \( B \) values into integers requires multiplying \( B \) values by \( 2^p \), and then dealing with the range \( N \) of \( 2^{64+p} \), which may simply be too large for open-form decryption at the Agent. One approach we will investigate is to decrypt the fractions (converted into integers) and the magnitudes separately; however, such an approach severely limits query expressibility and query processing capabilities of the database.

Discuss what to do.

4.1.2 Closed-Form Encryption

A. Integer Attributes

?? I need some help here.

B. Real Attributes

Add stuff from Sun’s report.

C. String-valued Attributes

The least secure cryptographic scheme for strings is a monoalphabetic cipher [Ref], that is, a cipher in which each occurrence of a symbol \( X \) is replaced by a symbol \( Y=E(X,K) \) (where \( K \) is the key). While it offers minimal protection, it does have the advantage that substring searches can be performed without decryption. The agent merely encrypts the substring and submits it to the database program for the search. The fact that such searches can be performed is precisely the weakness that allows the cryptanalyst to break the cipher by statistical analysis.

A more secure cryptographic scheme is the polyalphabetic cipher [Ref]. Here a different encryption \( E(X,i,K) \) is performed on the symbol \( X \) depending on its position \( i \) in the string. While such a cipher is still vulnerable to statistical attack, it is considerably more secure due to the disruption of substring frequencies. It is no longer possible to perform arbitrary substring searches, since the position of the substring within the string affects its encryption. It is still possible to search for prefixes without decryption. String-length is also unaffected by such encryption.

The most secure field-based encryption scheme uses the record number \( i \) and field position \( j \) to determine the encryption of the string [Den85]. This is also the scheme in which almost no searches can be done without decryption, although string length can still be preserved if this is desirable.

4.2 Inference Control Analysis

We will develop inference control techniques to

- Decide whether the adversary can or cannot infer the employed database encrypting techniques through additional information such as partial a priori information about the data in the database, or through scanning the data in the database physically, etc. And
- Employ algorithms/techniques to prevent users from inferring the information in the database.

Funda will supply this part.

4.3 Query Rewriting Techniques for Aggregate Functions and Nested Queries

It is easy to prove that
**Claim:** When the encryption function \( f() \) is order- and distance-preserving, all SQL queries of the original relational database DB expressible in safe relational calculus can be rewritten into SQL queries of the encrypted database DBMOD.

The claim fails however for SQL queries of relational databases involving aggregate functions, and for SQL queries of object-relational databases involving derived or complex types (as opposed to primitive types integers, reals, and strings). We will revisit derived and complex data types later in section 4. Let us review the case of SQL queries with aggregate functions where, in some cases, a given query to the original database translates into multiple queries to the encrypted database plus additional computation on the part of the Agent.

**Example 5.** Assume that the relation EMPLOYEE and the attribute name Salary are encrypted as characters R and B, and the attribute Salary is encrypted using an order-preserving encryption function \( f() \). Then, the SQL query

\[
\text{SELECT AVG (EMPLOYEE.Salary) FROM EMPLOYEE WHERE Salary > 100000}
\]

will need the evaluation of two queries:

\[
Q_1 (DBMOD): \quad \text{SELECT SUM(R.B) FROM R WHERE B > f(100000)}
\]

and

\[
Q_2 (DBMOD): \quad \text{SELECT COUNT(R.B) FROM R WHERE B > f(100000)}
\]

And, the Agent will compute the answer as a function of the responses \( O_1 \) and \( O_2 \) to \( Q_1, Q_2 \) respectively, and \( f() \). The answer is given by \( f^{-1}(O_1/O_2) \). This formula works because the special form of \( f() \) is such that it preserves arithmetic means.

We make two observations. The query rewriting approach of example 4 will fail if \( f() \) is not additive, i.e., \( f(a+b) = f(a) + f(b) \). That is, for any \( x \) and \( y \) values in the Salary attribute, the additivity property guarantees that \( f^{-1}(f(x) + f(y)) = f^{-1}(f(x + y)) = (x + y) \). For example, the function \( f(x) = x^2 \) is not additive, and cannot be used as the encryption function here. Second, open-form encryption functions by definition do not satisfy the additivity property.

**Example 6.** Consider the query in example 4. Now, assume that we have used an “instance-based” encryption; i.e., for each salary value \( a \), there is a distinct \( f(a) \) value, not expressible in a closed form and maintained in a secondary data structure at the Agent. In this case, the query rewriting scenario of example 4 fails, and \( Q_1 (DBMOD) \) is forced to return all of the salary values in the desired range as

\[
Q_1 (DBMOD): \quad \text{SELECT R.B FROM R WHERE B > f(100000)}
\]

and the Agent will need to (i) decrypt all of the values in the output \( O_1 (DBMOD) \), (ii) sum up the decrypted values into \( X \), and (iii) compute the output of the query as \( X/O_2 \). This scenario will incur heavy time delays since DBMOD salary values have to be shipped to the Agent, and the Agent needs to perform (possibly, a disk-based) addition, independent of the database query optimization process.

Thus, one important goal in query rewriting is to have minimal performance degradation in evaluating the new set of queries of the encoded database DBMOD. Therefore, the query rewriting techniques to be developed will be cost-based against expected query workloads of the database.

### 4.4 Query Processing with Encrypted Data

Database query engines are complex software systems that rely on statistics gathered on the performance of the queries, and multiple tree-structured or hashing-based indexes. Encrypting the data in the database will change both the statistics, and the indexing techniques needed by the query engines for query processing. In this research, we will investigate what is currently available in commercial systems as well as possible improvements to the existing systems that support encrypted database querying. We give three examples.

**Example 7.** Consider an attribute \( A \) where, for query processing efficiency, the DBMS deems it necessary to construct and maintain an index structure, say, a B+-tree index [RG00] on \( A \). Now assume that \( A \) is encrypted using
the encryption function $f()$ that does not satisfy the order-preservation property. This means that the DBMS cannot employ the B+-tree index as a tool to efficiently process queries, and the encryption function $f()$ is not acceptable as an encryption function.

**Example 8.** Assume that the attribute $B$ needs to have an inverted index [RG00] for efficient query processing and the encryption function $f()$ encrypts each value of a real-valued attribute $B$ into a vector with three elements, in order to avoid precision and overflow errors [Gold91] during encryption. This means that the functionality of the inverted index must be provided with the converted vector-valued encrypted $B$ attribute.

**Example 9.** Assume that we have an integer-valued attribute $A$ used in arithmetic expressions in SQL queries and whose domain values are very large, and any encryption technique used to encrypt an attribute $A$ value into another integer value results in integer overflows. Thus, assume that we encrypt a given $A$ value $x$ into a 2-tuple $(v_1, v_2)$, and we employ a query rewriting technique that is capable of rewriting any query $Q$ on the original database into a single query on the anti-tamper database. To support query processing on the anti-tamper database, the database administrator or the legitimate user of the original database may want to create a B+-tree index on encrypted $A$ values (which are now 2-tuples). To improve query optimization, the design of this B-tree index should be changed in light of the fact that the 2-tuple is an artifact of the encrypting, and, a reference to the 2-tuple is really a reference to an original $A$ value.

Note that the decrypting is directly proportional to the size of the query output, and can also be computationally very expensive. It is always much faster to decrypt if the decrypting function is in closed-form and involves only additions and deletions, as opposed to multiplications and divisions. And, it is more costly if the decrypting function involves square root operations, as opposed to multiplications and divisions. In comparison, decrypting values encrypted by open-form encryption is likely to be more costly as it usually will involve secondary-storage-based indexing structures. Thus, our query processing cost functions will take into account the expected size of the query output as well as the type of the inverse function used for decrypting.

## 5 Proposed Research Plan

TBD
6 References


[OR01] ORACLE 8i, St. Edition, commercially available DBMS.


[Ullm89] Ullman, J.,