RC24300 (W0707-044) July 5, 2007
Computer Science

IBM Research Report

Dual Encryption for Query Integrity Assurance

Haixun Wang, Jian Yin, Chang-Shing Perng, Philip S. Yu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Dual Encryption for Query Integrity Assurance

Haixun Wang Jian Yin Chang-Shing Perng Philip S. Yu

Abstract

In database outsourcing, an enterprise contracts its database management tasks to an outside database service provider to eliminate in-house hardware, software, and expertise needs for running DBMSs. This represents an attractive paradigm, especially for parties with limited abilities in managing their own data. Typically, the client applications want to obtain quality assurance (e.g., data authenticity and query completeness) of the outsourced database service at a low cost. Previous work on database outsourcing has focused on issues such as communication overhead, secure data access, and data privacy. Recent work has introduced the issue of query integrity assurance, but usually, to obtain such assurance incurs a high cost. In this paper, we present a new method called dual encryption to provide low-cost query integrity assurance for outsourced database services. Dual encryption enables "cross examination" of the outsourced data, which consists of the original data stored under a certain encryption scheme, and another small percentage of the original data stored under a different encryption scheme. We generate queries against the additional piece of data and analyze their results to obtain integrity assurance. Our scheme is provable secure, that is, it is impossible to break our scheme unless some security primitives can be broken. Experiments on commercial workloads show the effectiveness of our approach.

1 Introduction

Economic analysis shows that in the past five years, the cost of sending a terabyte of data across large geographic areas dip by 75%. The breakthrough helps accelerate the trend of information technology outsourcing. Recently, there is a growing interest in outsourcing database management tasks to an outside database service provider. The new paradigm has the apparent benefits of reducing in-house hardware, software, and human expertise costs for running DBMSs, and enabling businesses to concentrate on their core tasks.

Database outsourcing presents many challenges. These include traditional issues such as performance, scalability, and ease-of-use, which have been the topics of decades of database research, but have now gained a new dimension in the database outsourcing paradigm. The first concern raised by outsourcing is data security. We must ensure that the service provider, while providing query support to its clients, does not have access to the plain text content of the database. Several recent work [2, 18] addressed the issue of supporting encrypted queries over encrypted databases.

Data privacy is just one aspect of security. An equally important issue orthogonal to privacy is integrity. In the paradigm of database outsourcing, we use the term *integrity* or *query integrity* to refer to the validity of query results, in other words, we want to ensure that the results returned by the service provider for a user query are correct and complete. While the integrity of an individual tuple can be secured with a digital signature, query integrity has higher risk of being compromised. For example, attackers can return a subset instead of the complete set of tuples that satisfy a user query. In this paper, we aim at providing query integrity assurance at a low cost. To achieve this, we build on top of previous methods that support encrypted queries over encrypted databases. Closely related to our work is Sion et al's approach [25]. However, the adversary model in their approach is a little bit weak in the sense that they assume that the only goal of an attacker is to save some computation resource by processing the queries on an incomplete and hence small subset of the data instead of the whole dataset. In other words, the challenge token approach is ineffective if an adversary is willing to compute the complete query results and then delete some data from the query result before returning it to database clients. Such an adversary can be a malicious database service provider that tries to provide misleading query results or a hacker that breaks into the outsourced database machine which is not securely administrated.Moreover, the challenge token approach requires modifications in the DBMSs for query execution proofs.

Overview of Our Approach Our approach, in its simplest form, is similar to cross examination. The intuition is the following. Imagine we give the encrypted dataset T to two service providers A and B, and we assume they do not have any inter-communication. For every query q, we encrypt it and send it to both A and B. The results returned by A and B should be exactly the same after decryption. If they are not, then we know at least one of the service providers is not doing an honest job.

A clear advantage of the new scheme is that it does not require locally mirroring the outsourced database. Besides, the solution is conceptually straightforward, so is the analysis of integrity assurance. However, its success heavily depends on the assumption that the two service providers have no communication with each other. It becomes unrealistic, because in many cases, nothing more than minimal communication between A and B can defeat this approach. For instance, if A and B collaborate to return empty results to whatever queries they receive, then the user of the service cannot tell if it is true that no data satisfies the query. Although this attack is easy to foil, other problems may not be easy to solve. For instance, for a query q, A finds the correct answer ρ_A, and B finds the correct answer ρ_B. If A and B know that they are answering the same query, then A can replace T with ρ_A, and B can replace T with ρ_B, and execute later queries against ρ_A and ρ_B. Since the answers from A and B will always be the same, the user of the service cannot detect the breach of the query integrity. Furthermore, besides that the assumption of no communication between the two service providers is unrealistic, storing data at two service providers doubles storage, and other data management cost.

In this paper, we propose a new scheme called Dual Encryption for providing query integrity assurance. We select a small subset of T and encrypt it using two different keys. We show that we can use the two different encryptions to provide query integrity. In generally, it is difficult to argue the security of a scheme by showing that the scheme can withstand many types of secure attacks as it is difficult to exhaustively enumerate all the possible security attacks. To overcome this difficulty, we show that our scheme is provable secure. That is, the security of our scheme can be reduced to the security of some basic security primitives such as DES and any computational efficient algorithm that can break our scheme can also be used as a procedure to construct a computational efficient algorithm to break those security primitives. As those security primitives are being tested for many years and generally assumed to be secure in the security communities, our scheme is secure. Our approach incurs very small overhead, as our analysis indicates that with 10% of the data for dual encryption, we can already provide good integrity assurance.

Paper Organization The rest of the paper is organized as follows. In Section 2 we lay out the problem definition and study some background assumptions of database outsourcing. Section 3 analyzes a naïve approach for solving this problem. We present the dual encryption approach in Section 4. In Section 5, we show that our scheme, if implemented properly, is provable secure. We report experimental results in Section 6, and review related work in Section 7. We draw our conclusion in Section 8.

2 Problem and Background

In this section, we formalize the problem and introduce some background issues related to database outsourcing.

2.1 Problem Setting

A database owner outsources its data storage and management duty to a service provider. The encrypted data is accessed by multiple clients through secure channels. Besides normal data management services, the user of the service often want assurances of the quality of the service. In this paper, we focus on query integrity assurances, that is, how do users obtain assurances as to whether the query results are correct and complete?

Note that the users (e.g., cellphone or PDA devices) may have very limited computation power and data storage capacity, which means, i) they cannot execute expensive queries locally, and ii) they cannot store any significant amount of the outsourced data locally. Thus, the only way of obtaining any integrity assurance of the query results is through asking queries and analyzing results. To make it possible, we rely on two things. First, the database owner can embed additional information in the outsourced data. Second, the clients may construct additional queries against the original data as well as the embedded information stored at the service provider. We require that the storage overhead and the query overhead are kept at minimum.

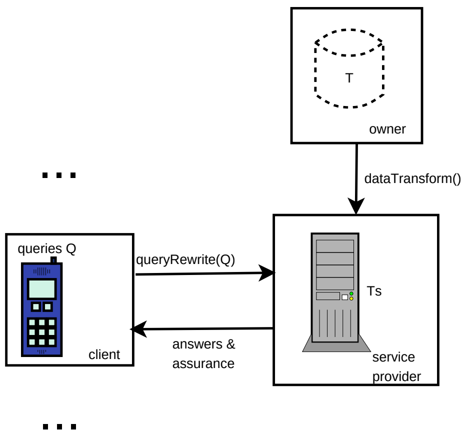


Figure 1. Database Outsourcing

In Figure 1, we illustrate the setting of the database outsourcing environment. Let T denote the data to be outsourced. The data T is to be pre-processed, encrypted and stored at the service provider. We use $\texttt{dataTransform}()$ to denote the pre-processing and encryption process, and we use $T_s = \texttt{dataTransform}(T)$ to denote the data stored at the service provider.

We rewrite a set of queries $\mathcal{Q} = \{q_1, \cdots, q_u\}$ against T to queries against T_s. Let $\texttt{queryRewrite}()$ denote the query rewriting process, and $\texttt{queryRewrite}(\mathcal{Q}) = \{r_1, \cdots, r_v\}$ denote the queries sent to T_s. The service provider, on receiving $\{r_1, \cdots, r_v\}$, returns results $R = \{\rho(r_1), \cdots, \rho(r_v)\}$ to the client, where $\rho(r_i)$ is the result for query r_i. From R, the client will derive the results of $\mathcal{Q}$ as well as the confidence of their integrity.

Thus, the problem of query integrity assurance comes to the following. Construct the two functions, `dataTransform`() and `queryRewrite`(), such that for a batch of queries, the confidence that the results returned by the service provider are correct and complete is beyond a user-specified level, and the overhead of data storage and query processing is as low as possible.

2.2 Background

In this section, we introduce some known issues and techniques related to the database outsourcing problem.

Querying Encrypted Data

Encryption is a well established technology for protecting sensitive data [15, 24, 27]. The data at the service provider is encrypted. This creates a lot of challenges for providing query support over encrypted data. For example, to support range queries, we must ensure that the encryption of numerical values are order-preserving so that we do not have to decrypt the data before evaluating the queries. We also need to provide index so that queries over encrypted data can be carried out in an efficient way. This has been a focus of much recent work. For instance, Hacıgümüş et al [18] explored techniques to execute SQL queries over encrypted data. It is later improved by OPES [2], which returns exact answer instead of superset of answers that require filtering on the client side.

Our approach builds on top of existing techniques that support querying over encrypted databases. We assume the underlying encryption scheme already takes care of issues such as privacy protection, and is robust to a variety of attacks (such as data distribution attacks) against encrypted databases. Thus, we only focus on new security concerns introduced in providing query integrity assurance.

PKI, Secret Key, and Signature

In cryptography, a public key infrastructure (PKI) is an arrangement which provides for third-party vetting of, and vouching for, user identities. User identities, or signatures, can be generated with either PKI or a secret key. PKI-based signature separates the ability of verifying a signature from signing a signature. Public key is used for verifying a signature and the private key is used for generating signature. In our case, we argue that using the secret key approach is sufficient, as we trust the owner of the database and the users of the database to keep a secret key from attackers. The advantage of the secret key approach is that the overhead is significantly less.

One-way Hash Functions

Our approach relies on a standard security technique, the one-way hash functions [4]. A one-way hash function H takes a variable-length input string x and converts it into a fixed-length (usually 128 bits) binary sequence $H(x)$. It is designed in such a way that it is hard to reverse the process, that is, for a given value x' it is computationally infeasible to find a string x such that $H(x) = x'$. For some one-way hash functions, it is also computationally infeasible to find two strings x and y, such that $H(x) = H(y)$. These properties make one-way hash functions a central notion in public-key cryptography.

Our approach adopts dual encryptions of the data, i.e., we select a small subset of the data and encrypt it using two different keys. For any given record, the user of the service can find out easily whether it has been selected for dual encryption, and if it is, which of the two keys it is encrypted with. These information however must be kept away from the service provider, or potential adversaries. We use one-way hash functions to achieve this, and the details are discussed in Section 4.

3 Data Correspondence

In this section, we introduce the concept of *data correspondence*. It reveals a major difficulty that a cross examination based approach must overcome in order to provide query integrity assurances.

In the naïve setting, two sites are involved in a cross examination. More specifically, assume we store the same dataset T at two service providers A and B. We denote the encrypted data at A as T_A, and that at B as T_B, and we assume that A and B do not have intercommunication. Every user query q is encrypted and sent to both of the service providers. If both A and B execute the queries and send back the results honestly and correctly, then the results should be exactly the same after decryption.

The feasibility of this approach depends on the assumption that the two service providers cannot communicate with each other. Under this assumption, the best way it can cheat is to "guess" the answers. However, as long as one of them is "guessing", it will be very unlikely that the results they provide for a query will turn out to be the same. To see this, assume dataset T contains N_T tuples. For any query that finds tuples satisfying a certain predicate, the probability that A and B return the same answer is 2^{-N_T}, if one or both of them is guessing randomly.

However, the no-communication assumption is too strong to realize in practice. This is exacerbated by the fact that minimal communication between the two service providers can break the scheme. To see this, we define *data correspondence* between two copies of encrypted data.

Definition 1. *Assume T_A and T_B are encryptions of T. Let $D_A \subseteq T_A$ and $D_B \subseteq T_B$. We say D_A correspond to D_B if they are encryptions of the same subset of data in T.*

Intuitively, two datasets correspond to each other if they are different encryptions of the same plain text data. Data correspondences are a severe security risk for the cross examination approach in providing query integrity assurances.

Property 1. *If the two service providers discover that D_A correspond to D_B, they can evade the integrity check by simply doing one of the following*

1. remove D_A and D_B from T_A and T_B, or

2. replace T_A and T_B with D_A and D_B

and then use the new T_A and T_B to answer future queries.

The proof is trivial because T_A and T_B are the same data under different encryptions, so any query against T_A and T_B should return the same results. It is a clear integrity breach because queries are now running against subsets of the original data.

Unfortunately, it is very easy for the service providers to find such correspondence as long as they have minimal communication with each other. For instance, the empty-result attack we mentioned earlier corresponds to the 2nd form of attacks where $D_A = D_B = \emptyset$. Other non-trivial data correspondence can be discovered by analyzing the queries the two service providers receive from the user. For instance, if A and B knows that they are answering a same query t, then they can use the query results of t as D_A and D_B, and carry out the above attacks.

In this paper, we introduce the dual encryption model based on the intuition of cross examination. Clearly, our major challenge lies in finding a solution that can handle data correspondence with low cost. In addition, we must avoid another severe drawback of the naïve cross examination approach – it needs two service providers, as it doubles the cost of database outsourcing, and introduces extra overhead of communication.

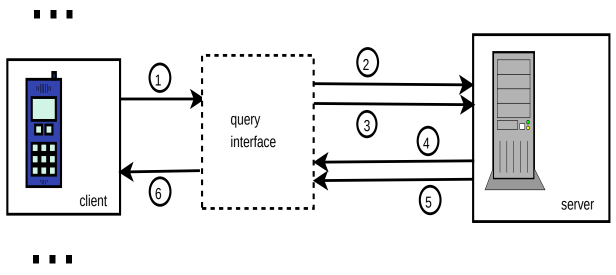
4 The Dual Encryption Approach

In this section, we describe a solution to providing query integrity assurances for database outsourcing.

4.1 Overview

In our approach, the database owner outsources its data T to the service provider through dual encryption, namely, a primary encryption and a secondary encryption. More specifically, we encrypt the entire data T using a primary encryption key k, and we encrypt a selected, small subset of T using a secondary encryption key k'. The encrypted data are merged and stored at the service provider as a single piece. The detail of the encryption process is discussed in Section 4.2.

User queries go through a query interface before being sent to the service provider. The query interface performs query rewriting and monitors query integrity by analyzing query results returned by the service provider. More specifically, given a batch of queries $\mathcal{Q} = \langle q_1, \cdots, q_u \rangle$, the query interface sends $\mathcal{Q}^k = \langle q_1^k, \cdots, q_u^k \rangle$ to the service provider, where q_i^k denotes the query q_i encrypted using the primary encryption key k. Based on $\mathcal{Q}$, the query interface generates a new batch of queries and sends them to the service provider after applying the secondary encryption. We use $\mathcal{Q}^{k'} = \langle r_1^{k'}, \cdots, r_v^{k'} \rangle$ to denote these queries. From the results of $\mathcal{Q}^k$ and $\mathcal{Q}^{k'}$, the query interface derives the correct answers of $\mathcal{Q}$ as well as the assurance of their integrity, and sends them to the user. Figure 2 illustrates the role of the query interface. We discuss query rewriting in detail in Section 4.3.



(1)	query sequence $\mathcal{Q} = \langle q_1, \cdots, q_u \rangle$
(2)	primary queries $\mathcal{Q}^k = \langle q_1^k, \cdots, q_u^k \rangle$
(3)	secondary queries $\mathcal{Q}^{k'} = \langle r_1^{k'}, \cdots, r_v^{k'} \rangle$
(4)	answers to primary queries $\langle \rho(q_1^k), \cdots, \rho(q_u^k) \rangle$
(5)	answers to secondary queries $\langle \rho(r_1^{k'}), \cdots, \rho(r_v^{k'}) \rangle$
(6)	query answer $\langle \rho(q_1), \cdots, \rho(q_u) \rangle$

Figure 2. Our approach

4.2 Dual Encryption

In databases, tuple encryption promises to disassociate a tuple's encrypted text from its plain text content. Dual encryption exploits this promise to provide integrity in addition to privacy.

The idea is to encrypt some tuples in a database with two keys. Given a tuple r in the original database T and two encryption keys k and k', the encrypted database T_s can contain both $E_k(r)$ and $E_{k'}(r)$, which are the encrypted texts of r using key k and k' respectively.

We must overcome the data correspondence problem. The rationale is that, if the attacker does not know data correspondence, that is, he does not know that $E_k(r)$ and $E'_k(r)$ are different encryptions of the same original tuple r, then deleting tuples[1] from the encrypted database will likely lead to a situation where one tuple, say $E_k(r)$, is deleted while its corresponding tuple, $E'_k(r)$, remains, which enables us to detect the attack.

We discuss our dual encryption scheme in detail in two parts, data transformation and query encryption.

4.2.1 Data Transformation

Dual encryption can build on top of any encryption scheme that supports queries over encrypted data. These include schemes proposed in the recent work [18, 2]. We assume the underlying encryption scheme already provides good security (e.g., protection against data distribution attacks) for querying over encrypted databases.

[1]The attacker can also modify or add tuples. But these attacks are easily detected as described below.

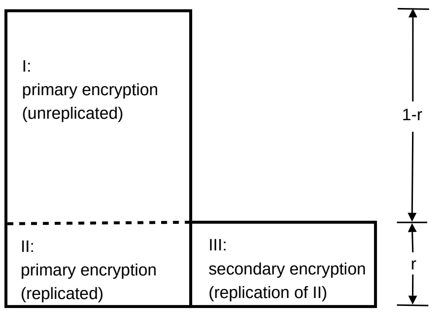


Figure 3. Dual Encryption

For our dual encryption approach, we encrypt the original dataset T with a primary key k, and we replicate r percent of T and encrypt the replication using a secondary key k'. We denote r as the replication factor. The encrypted dataset T_s contains data of two different encryptions, and we store T_s at the service provider. As shown in Figure 3, we can view T_s as composed of 3 parts that are mixed together. Part I and II correspond to the primary encryption of T, and part III corresponds to a different encryption of part II. Because tuples of the three parts are mixed together, given any tuple t in T_s, we cannot tell which part t belongs to. In particular, the domain of the two encryptions can overlap, that is, two different tuples, one encrypted with k, the other k', may have the same encrypted text.

Nevertheless, in order to assess query integrity from the answer returned by the service provider, for any tuple t in the answer, the client need to know:

1. whether t is a valid tuple of T, and
2. how t is encrypted (in particular, whether t has a corresponding tuple encrypted using a differet key).

Once the client knows that t is valid, and t satisfies the query condition, then the client knows that t is a correct answer to the query. On the other hand, knowing how t is encrypted is important in evaluating query integrity, which we discuss in Section 5.

In order to have the above information, we attach additional information to each tuple in T. We call it the *dual* information. We use secret key and one-way hashing to generate the *dual* information for each tuple t. Assume a secret key e is shared by the database owner and the users of the database service. For any tuple t, its dual information t_{dual} is computed by a one-way hash function $H()$ as follows.

$$t_{dual} = \begin{cases} H(e,t) & : \ t \in \text{ part I} \\ H(e,t)+1 & : \ t \in \text{ part II} \\ H(e,t)+2 & : \ t \in \text{ part III} \end{cases} \qquad (1)$$

According to the property of one-way hashing, given t, it is easy to compute $H(e,t)$. The user of the service, on receiving t as an answer from the service provider, can easily check if the value of its *dual* information is among $\{H(e,t), H(e,t)+1, H(e,t)+2\}$. If it is not, we know t is not a valid tuple in the original dataset T. If it is, we know how it is being encrypted, that is, whether it is encrypted with a primary key, and whether it is one of the replicated tuples.

Only the client has the knowledge of how a tuple is encrypted, and this will enable us to assess query integrity, which we discuss in Section 5. For the service provider or any adversary, since H is a one-way hash function, it is computationally infeasible to find out the secret key e given any tuple t and its t_{dual} value. Thus, it is unlikely that an adversary can generate a valid t_{dual} without knowing e. The probability that a random guess of t_{dual} happens to be one of the three valid values for t is $3/2^{128}$, as one-way hash functions typically convert the input into a binary string of 128 bits. Thus, it is difficult for the adversary to change the content of the database, or to distinguish a tuple encrypted with the primary key from a tuple with the secondary key.

We can show that the dual encryption scheme is provable secure against data correspondence attacks when the adversaries are computationally bounded. More specifically, if we assume that the underlying encryption algorithm is a pseudo-random function or a pseudo-random permutation in the sense that the best adversary only has advantage ϵ than a random guess [17, 20, 8], then we can show, by contradition, that no algorithm can have advantage by ϵ' than randomly guessing at corresponding tuples, where ϵ' is a function of ϵ. We omit the proof here due to lack of space.

4.2.2 Query Encryption

In order to query encrypted data, the query itself must also be encrypted. Our method builds on top of previous approaches that support querying over encrypted data [18, 2]. For example, if the underlying encryption scheme is order preserving, our scheme supports range queries against outsourced data. Similar to previous work on integrity assurances [22, 25], we are concerned with identity queries whose result is a subset of T. This means we consider only queries testing equality and other logical comparison predicate clauses.

More specifically, let q be an identity query. It should have the following form [22]:

q: SELECT * FROM T WHERE $predicate$;

Each literal of *predicate* is in the form of:

$$a_i \text{ cond } v_i$$

where a_i is an attribute, v_i a value in the domain of a_i, and $cond$ is an operator such as $=, >, <, \geq, \leq$. For example, q can be the following query:

q: SELECT * FROM T WHERE $a_1 = 2$ AND $a_3 < 100$;

The above query q on table T will be encrypted when it is sent to the server. Since we have two keys, k and k', the above query can be transformed into two different forms:

q^k : SELECT * FROM T_s
WHERE $a_1 = E_k(2)$ AND $a_3 < E_k(100)$

$q^{k'}$: SELECT * FROM T_s
WHERE $a_i = E_{k'}(2)$ AND $a_3 < E_{k'}(100)$

Note that the service provider executes a query, regardless how it is encrypted, against the entire T_s that contains data encrypted in two different ways. Note that for security reasons, we may choose E_k and $E_{k'}$ whose domains overlap. For example, there may exist values x and y such that $E_k(x) = E_{k'}(y)$, where $x \neq y$. Thus, query $q^{k'}$ may return tuples encrypted with key k as well. These tuples are to be filtered out during decryption according to Eq 1.

Next, we discuss how to take advantage of the relationship between the results of q^k and $q^{k'}$ to determine whether the service provider carries out the query in honestly.

4.3 Cross Examination

The purpose of dual encryption is to allow for sophisticated cross examination. We have mentioned in Section 4.2.1 that dual encryption is secure against data correspondence attacks: with the assumption that the underlying encryption mechanism is provable secure, the best adversary has no advantage more than ϵ' over random guessing in finding corresponding tuples. However, queries posed by the clients may reveal critical information to allow the service provider to detect data correspondence.

Query Correspondence Attacks

To understand the new challenge, we first analyze a simple approach which performs cross examination on dually encrypted data. Assume a user has a batch of queries $\mathcal{Q} = \langle q_1, \cdots, q_u \rangle$. For each query $q_i \in \mathcal{Q}$, the user sends q_i^k to the service provider and gets back result $\rho(q_i^k)$. Later, to check the integrity of $\rho(q_i^k)$, it sends $q_i^{k'}$ to the service provider. The user then evaluates the "trustworthiness" of the service provider by analyzing $\rho(q_i^k)$ and $\rho(q_i^{k'})$, which involves checking for every replicated tuple $t \in \rho(q_i^k)$, whether t's replication is in $\rho(q_i^{k'})$.

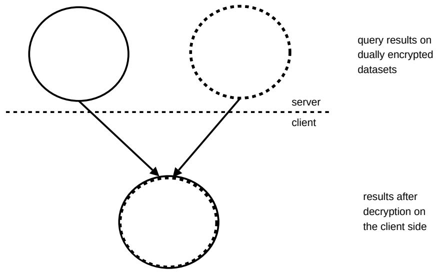

query results on dually encrypted datasets
server
client
results after decryption on the client side

Figure 4. Identical Queries

We illustrate the simple approach in Figure 4. The two queries, q_i^k and $q_i^{k'}$, are semantically identical. In particular, if every tuple is replicated (r=1), then after decryption, $\rho(q_i^k)$ and $\rho(q_i^{k'})$ will correspond to the same set of tuples. This induces a security risk. If an adversary finds out that q_i^k and $q_i^{k'}$ are different encryptions of a same query, it knows immediately that $\rho(q_i^k)$ *correspond* to $\rho(q_i^{k'})$, which enables the adversary to launch data correspondence attacks. The risk is high because although queries are encrypted, the encryptions are applied on the values in the queries not the queries themselves. For instance, given q_i = "SELECT * FROM T WHERE $a_i = 3$", the encrypted queries q_i^k and $q_i^{k'}$ differ only in the way of encrypting the value 3. Thus, it is possible to discover the correspondence between two queries by performing simple syntactical checking.

In our approach, the query we issue has different semantics for the query whose integrity we want to evaluate, that is, given a query q_i, we derive a checking query q, such that

$$\rho(q^{k'}) \neq \rho(q_i^k) \qquad (2)$$

To use query q to assess the integrity of query q_i, we require that the result of q and the result of q_i have certain overlap, that is,

$$\rho(q^{k'}) \cap \rho(q_i^k) \neq \emptyset \qquad (3)$$

Eq 2 and Eq 3 are the only conditions for applying our approach. This is illustrated in Figure 5.

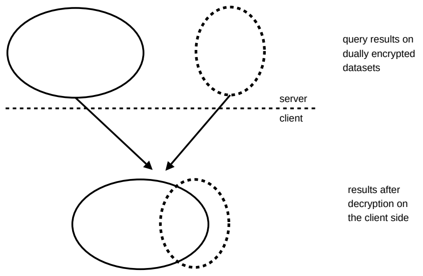

query results on dually encrypted datasets
server
client
results after decryption on the client side

Figure 5. Our Approach

It follows that if we can ensure the result of q overlaps with the result of multiple queries, we can use q, a single query, to evaluate the integrity of multiple queries, instead of generating a checking query for each of them. The benefits of evaluating the integrity of multiple queries at a time is two-fold. First, it increases the difficulty for an adversary to match a checking query with the original query. This reduces the risk of the data correspondence attack. Second, because there are fewer checking queries, the overhead of query processing is reduced.

Distribution Attacks

Assume for a batch of queries $\mathcal{Q} = \langle q_1, \cdots, q_u \rangle$, the client generates a single checking query q. It then sends $\hat{\mathcal{Q}} = \mathcal{Q}^k \cup \{q^{k'}\}$ to the server. There is still a subtle potential risk, which we describe below.

By studying the distribution of query results, an adversary may discover data correspondence with high probability. Let t be a tuple in the answer set of $\mathcal{Q}$ and let s denote the probability that t also satisfies q, in other words, s denotes the level of overlap. Recall that the replication factor is r, then the probability that $\rho(\hat{\mathcal{Q}})$ also contains t's corresponding tuple is sr. If sr is larger than the probability that $\rho(\hat{\mathcal{Q}})$ contains a random tuple, then the adversary may discover data correspondence after seeing a large number of queries.

This issue can be resolved by considering the size of $\rho(\mathcal{Q}^k)$ before we decide if we should send a checking query now or we should delay the checking query. If $|\rho(\mathcal{Q}^k)|/|T_s|$ is no less than rs, then the checking query poses no risk, because a random tuple has as high probability as a corresponding tuple to appear in $\rho(\hat{\mathcal{Q}})$. Note that we can always construct additional queries so that for any tuple t covered by the checking query, a deterministic set of tuples is queried and returned with probability higher than rs.

5 Security Proof

We first prove the security properties of our scheme in an ideal setting, where each tuple is encrypted with DES.

We assume some basic security properties of these primitives as in most security literature. Our system can then be shown to be provable secure [10, 7, 11, 6, 5, 9, 26]. That is, the security of our system can be reduced to the security properties of the underlying crypto primitives that we employed to build our system.

First, we introduce some standard crypto primitives used in provable secure literature. These primitives can be built with the standard DES symmetric key encryption [9].

A mapping $F : K \times X \rightarrow Y$ where k is chosen uniformly randomly from the key space K is said to be a (q, t, ϵ)-pseudorandom function if there does not exists an algorithm $\mathcal{A}$ that can ϵ-distinguish this function from a truly random function $U : K \times X \rightarrow Y$, where U is chosen uniformly random from all the set of random functions that map X to Y for each chosen k in K, with no more than q queries and t computation.

The following notations are used in our proof and in the performance analysis.

T	original table
N_T	number of tuples in T
r	replication factor
T_s	table stored at the service provider
$\mathcal{Q}$	a batch of queries against T
q	a query generated to evaluate $\mathcal{Q}$'s integrity
s	q's selectivity ($s = \frac{\lvert q\cap\mathcal{Q}\rvert}{\lvert\mathcal{Q}\rvert}$)
s'	$\mathcal{Q}$'s selectivity ($s' = \frac{\lvert q\cap\mathcal{Q}\rvert}{\lvert q\rvert}$)

Table 1. Notation

Theorem 1. *There does not exist an adversary algorithm that can succeed in selecting m tuples that can be deleted without being detected with a probability significant higher than*

$$\sum_{m=u+2v, u\geq 0, v\geq 0} g(u)f(v)$$

with $t + c$ computation and $q + T_N + rT_N$ queries.

Proof. (Sketch) We prove this by contradiction. We assume there exists an algorithm $\mathcal{B}$ that can successfully choose m tuples that can be deleted without being detected from an encrypted database. We then construct an algorithm A that breaks F, that is, we show F is not a (t, q, ϵ)-pseudorandom function.

We construct an algorithm A that works as follows. Algorithm A takes a function as the input. This function can be either a truly random function U or a pseudorandom function F.

A then apply the function to all the tuples T in the database with the primary key as the first argument to the function and then apply the function to the replicated set of tuples T' with the secondary key as the first argument to the function. We call the function application here encryption. The results are combined and inserted into the database table. We then invoke the adversary algorithm $\mathcal{B}$ to select m items.

There are two ways to select m items. First, a tuple, t_1, resulting from applying one functions to a non-replicated tuple can be selected. We call this selection*non-replication selection*.Second, we can select two tuples: one is encrypted with the primary key; the other is the same tuple encrypted with the secondary key. We call this selection *replication selection*.

First, to successfully carry out *non-replication selection* of u tuples, all the u tuples must come from the tuples that have been encrypted with the primary key only. If the map function is truly random, there is an equal chance for a tuple to be mapped to any tuple in the encrypted table. Since there are $(1 - r)N_T$ tuples encrypted with the primary key only, therefore there are $\binom{(1-r)N_T}{u}$ ways of picking u tuples encrypted with the primary key only. As there are $\binom{(1+r)N_T}{u}$ ways of picking any u rows from the entire table T_s, the probability is

$$g(u) = \frac{\binom{(1-r)N_T}{u}}{\binom{(1+r)N_T}{u}}$$

Second, to successfully carry out *replication selection* of $2v$ tuples, the algorithm must pick v tuples encrypted with the primary key and the corresponding v tuples encrypted with the secondary key. There are $\binom{rN_T}{v}$ ways of picking v tuples in T_s that are encrypted with both the primary key and the secondary key. Thus, the probability of carrying out u successful replication selections is

$$f(v) = \frac{\binom{rN_T}{v}}{\binom{(1+r)N_T}{2v}}$$

Finally, if m tuples are picked without possibly being detected, there must be v replication selections and u non-replication selections such that $m = 2v + u$. Thus, the probability of deleting m successfully is

$$\sum_{m=u+2v,u\geq 0,v\geq 0} g(u)f(v)$$

As we assume that algorithm $\mathcal{B}$ can select m tuples with a probability

$$\sum_{m=u+2v,u\geq 0,v\geq 0} g(u)f(v) + E$$

where E is significantly bigger than (t, q, ϵ). After the algorithm $\mathcal{B}$ outputs m tuples, algorithm A takes over and verify whether these tuples can be deleted without detection, if so, it output 1 and outputs 0 otherwise. Hence, we have

$$\begin{aligned} Adv_A &= Pr[A(F) = 1] - Pr[A(U) = 1] \\ &= \sum_{m=u+2v,u\geq 0,v\geq 0} g(u)f(v) + E - \\ & \quad \sum_{m=u+2v,u\geq 0,v\geq 0} g(u)f(v) \\ &= E > \epsilon \end{aligned}$$

which contradicts the fact that F_k is a (t, q, ϵ)-pseudorandom function. □

Furthermore, it is easy to see that

$$g(u) = \frac{((1-r)N_T)((1-r)N_T - 1) \cdots ((1-r)N_T - u + 1)}{((1+r)N_T)((1+r)N_T - 1) \cdots ((1+r)N_T - u + 1)}$$

Since

$$\frac{((1-r)N_T)}{((1+r)N_T)} > \frac{((1-r)N_T - x)}{((1+r)N_T - x)}$$

for x in $1, \cdots, u-1$, we have

$$g(u) < (\frac{(1-r)N_T}{(1+r)N_T})^u = (\frac{1-r}{1+r})^u$$

Note also this bound is quite tight for large T_N.

Similarly, we have

$$f(v) < (\frac{2v}{(1+r)N_T})^v (\frac{v}{(1+r)N_T - v})^v$$

As $f(v) \ll g(u)$ for $u, v \leq m$ as N_T is big, $Y(m) \approx g(m)$. Thus, the probability of not being detected always approaches 0 rapidly for large T_N.

6 Experimental Evaluation

In this section, we evaluate the security and the performance overhead of the dual encryption scheme.

Datasets The data we use in our experiments is derived from the database in the SPECjAppServer benchmark [14], which intends to model real-world E-commerce workload and is widely accepted for evaluating the performance of J2EE servers. The database in this benchmark has two domains: dealer and manufacture. We use the customer table in the dealer domain for our experiments. The database in this benchmark is scaled with a parameter called `InjectionRate`. The number of tuples in this table is $7500 * \texttt{InjectionRate}$. In our experiment, we use an `InjectionRate` of 400, which gives us 3 millions tuples.

Storage Overhead In order to secure privacy and integrity in database outsourcing, the dual encryption scheme incurs some storage overhead. These include i) the overhead of storing encrypted tuples instead of plaintext tuples, ii) the storage overhead for the additional *dual* column, and iii) the storage overhead for the subset of the data replicated for secondary encryption. However, i) is common to any scheme that tries to protect data privacy. The *dual* column serves as signatures for tuple authentication. In dual encryption, the signature is overloaded as an indicator of how tuples are encrypted, and it does not create further space overhead. Thus, the only overhead introduced for providing integrity assurance is iii), i.e., the r percentage of the original data replicated for secondary encryption.

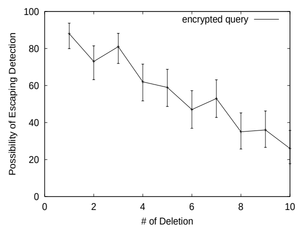
Security Against Data Alternation Attacks Recall that in data deletion attacks, an attacker randomly deletes m tuples from the outsourced table. For each value of m (ranging from 1 to 30), we repeat our experiment 100 times, and count the times that the attacks can escape detection. We also calculate the confidence interval for the probabilities by using exact Binomial distribution instead of using normal distribution to approximate binomial distribution. Figure 6 shows the probability of successful attacks for duly encrypted data with replication rates fro 10% to 50%. As we have expected, at only 10% overhead, the probability drops rapidly when m increases. Moreover, the probability of carrying out successful data deletion attacks approaches 0 as the attack tries to delete more than 15 tuples even if we use a replicate rate of merely 10%. Note that the data has 3 million tuples, and 15 out of 3 million (0.0005%) is a very small number.

Security Against Query Result Alternation Attacks We evaluate the rate of successful query result deletion attacks. We assume that the original query is of the form

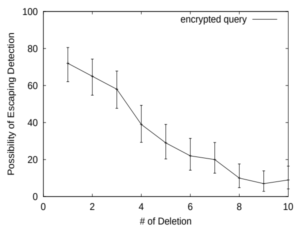
SELECT * FROM T WHERE $A_i = v$;

For every 10 queries $\mathcal{Q} = \langle q_1, \cdots, q_{10} \rangle$, we form a checking query that covers their query results. The checking query is a relaxed query based the above queries. The queries are executed against T with 3 million records.

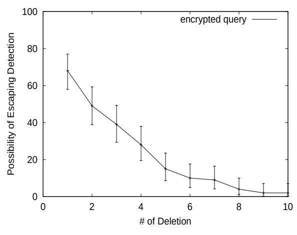
We then simulate query result deletion attacks. We randomly delete m tuples, and check if we can detect the dele-



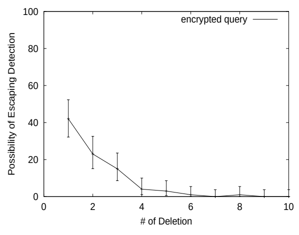
(a) $r = 10\%$



(b) $r = 20\%$



(c) $r = 30\%$



(d) $r = 50\%$

Figure 7. Ratio of attacks that escape a single check.

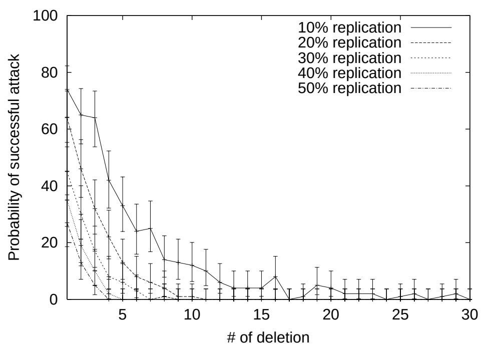


Figure 6. Data Deletion Attacks

tion by analyzing the query results. For every m, we repeat the process 100 times and find the average detection probability and the confidence interval. As we can see from the curves in Figure 7, the probability of escaping detection rapidly approaches 0 as the number of deletion increases.

7 Related Work

There is an extensive body of reseach in traditional DBMS security [12, 19, 23]. However, these studies mainly focus on providing access control to sensitive data. Although access control mechanisms are effective in the traditional database setting, it is inapplicable in the database outsourcing paradigm where the data and the query execution are hosted by a third-party service provider in a different administration domain.

There have been some studies focusing on the privacy aspect of outsourced database. Hacıgümüş et. al. [18] proposed a method to query the encrypted database and Agrawal et. al. [2] proposed an order-preserving encryption scheme for numeric values. These methods enable direct execution of encrypted queries on encrypted datasets. In our study, we use these methods, particuarly the order-perserving encrytion, to ensure that we can ask identity queries over data of different encrytions. Recently, there has been much work on security issues for data mining application. For example, the so-called privacy-preserving data mining [3, 1] focuses on potential security hazards posed when an adversary has data mining capabilities.

In the database outsourcing environment, most of the studies on the integrity aspect of data security do not consider query completeness. Premkumar et. al. [16] proposed a scheme to use Merkel trees to authenticate databases that are published by a third party. Mykletun et. al. investigated the feasibility of using various signature schemes including BGLS and Merkle Trees to authenticate returned results. In many cases, we can use less expensive symmetric signature schemes for authenticity as the clients in our outsourced database model and the owners of the data are trusted. In other words, we do not separate the ability of reading the data and creating the data. Only in the case that we want to have two seperated groups of clients, one group can only read the data and the other can also create the data, we need a PKI-based non-symetric signature scheme.

Mykletun et. al. [22] were the first to identify the query completeness as one aspect of intergity, but they did not investigate possible solutions. Probably the most related work to this study is described in Sion [25]. In that study, Sion proposeed to use Challenge Token as a query execution proof technique to ensure that a query is processed over the entire dataset. Their adversary model is not as general as ours in that the only purpose of an adversary to produce incomplete results is to reduce resource consumption. Their approach is not applied to the adversary model where an adversary can first compute the complete query result and then delete the tuples specifically corresponding to the challenge tokens. Moreover, their approach requires software modification on database servers to add query execution proof while our approach does not require any modification of DBMS software.

Our study falls into the general category of research on computing over a set of untrusted hosts. Our approach can be thought of as encrypting a computation twice and then decrypting the results to check whether they are matching. Several algorithms have been proposed to encrypt computation [13, 28]. However, in general, these algorithms are only constructed to demonstrate theoretical possibilities and are not intend to be implemented at efficiency level accepted by any real-world system.

Many practical systems also resort to relax the assumption about the trustworthiness of the hosting environment. The most common relaxation is to assume that the number of untrusted machines only constitutes a minority of hosts [21, 29]. Majority vote is used to select the correct computation results. In general, the assumption that the trustworthy hosts constitute the majority is hard to check and verify. Moreover, a computation must be repeated for many times to derive the results in majority voting. By focusing on a special type of computation, i.e., queries in databases, we are able to eliminate the assumption and the drawbacks associated with it.

8 Conclusion

Recently, there is a growing interest in database outsourcing. Providing verifiable quality-of-service assurances is to the interest of both the service providers and the users of the service. We are concerned with the issue of query integrity, that is, whether the results provided by the service are correct and complete. Previous approaches require the database servers to be modified and thus can be difficult to deploy. Our approach is database server transparent in the sense that it does not require database servers to be modified. Moreover, our approach is efficient and it only introduce small overhead.

References

[1] Dakshi Agrawal and Charu C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *PODS*, 2001.

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In *SIGMOD*, 2004.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *SIGMOD*, pages 439–450. ACM Press, May 2000.

[4] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: A survey. Technical Report 95-09, Department of Computer Science, University of Wollongong, 1995.

[5] Mihir Bellare. Practice-oriented provable security. In Ivan Damgård, editor, *Lectures on Data Security*, volume 1561 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 1998.

[6] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *FOCS*, pages 394–403, 1997.

[7] Mihir Bellare, Roch Guérin, and Phillip Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *CRYPTO*, pages 15–28.

[8] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. *Lecture Notes in Computer Science*, 839:341–358, 1994.

[9] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[10] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO*, pages 232–249.

[11] Mihir Bellare and Phillip Rogaway. Provably secure session key distribution: the three party case. pages 57–66, 1995.

[12] Elisa Bertino, Sushil Jajodia, and Pierangela Samarati. A flexible authorization mechanism for relational data management systems. *ACM Transactions on Information Systems*, 17(2):101–140, 1999.

[13] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO*, pages 87–119, 1987.

[14] Standard Performance Evaluation Corporation. The SPECjAppServer benchmark. http://www.spec.org/jAppServer/, 2004.

[15] D. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[16] Premkumar T. Devanbu, Michael Gertz, Charles U. Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security*, 2001.

[17] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792C807, October 1986.

[18] Hakan Hacıgümüş, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database service provider model. In *SIGMOD*, 2002.

[19] Sushil Jajodia, Pierangela Samarati, V. S. Subrahmanian, and Eliza Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD Record*, pages 474–485, 1997.

[20] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373C386, 1988.

[21] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. In *Proc. of annual ACM symposium on Theory of computing*, pages 569–578, 1997.

[22] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and integrity in outsourced databases. In *NDSS*, 2004.

[23] Matunda Nyanchama and Sylvia L. Osborn. Access rights administration in role-based security systems. In *IFIP Workshop on Database Security*, pages 37–56, 1994.

[24] B. Schneier. *Applied Cryptography*. John Wiley, 2nd edition, 1996.

[25] Radu Sion. Query execution assurance for outsourced databases. In *VLDB*, 2005.

[26] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[27] D. R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 2nd edition, 2002.

[28] Andrew C. Yao. Protocols for secure computation. In *Proc. of IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[29] J. Yin, J. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault tolerant services, 2003.