Providing Freshness Guarantees for Outsourced Databases [*]

Min Xie[†] Haixun Wang[‡] Jian Yin[‡] Xiaofeng Meng[†]

[†]Renmin University of China
Beijing 100872, China
{xiemin,xfmeng}@ruc.edu.cn

[‡]IBM T. J. Watson Research Center
Hawthorne, NY 10532, USA
{haixun,jianyin}@us.ibm.com

ABSTRACT

Database outsourcing becomes increasingly attractive as advances in network technologies eliminate the perceived performance difference between in-house databases and outsourced databases, and price advantages of third-party database service providers continue to increase due to economy of scale. However, the potentially explosive growth of database outsourcing is hampered by security concerns, namely data privacy and query integrity of outsourced databases. While privacy issues of outsourced databases have been extensively studied, query integrity for outsourced databases has just started to draw attention from the database community. Currently, there still does not exist a solution that can provide complete integrity. In particular, previous studies have not examined the mechanisms for providing freshness guarantees, that is, the assurance that queries are executed against the most up-to-date data, instead of just some version of the data in the past. Providing a practical solution for freshness guarantees is challenging because continuously monitoring data's up-to-dateness is expensive. In this paper, we perform a thorough study on how to add freshness guarantees over proposed schemes (including authenticated data structure-based and probabilistic-based approaches) to provide integrity assurance. We implement our solutions and perform extensive experiments to quantify the cost. Our experiment results show that we can provide reasonable tight freshness guarantees without sacrificing much performance.

1. INTRODUCTION

Database outsourcing becomes increasingly attractive as network performance continues to improve and the cost continues to decrease. However, concerns about the quality of services, security in particular, hamper the growth of database outsourcing.

There are two aspects of security in database outsourcing: privacy and integrity. Much previous work has addressed the issue of safeguarding data privacy [1, 6]. Integrity, on the other hand, has received attention from the research community just recently [12, 13, 14, 10]. By integrity, we mean that query results returned by the service provider must be correct.

Current known schemes that provide query integrity over outsourced databases can be categorized into two types. The first type is based on authenticated data structures [10], and the second type uses a probabilistic approach [15]. These approaches have a limited scope – they only consider queries against read-only databases. A more challenging aspect of providing integrity assurance lies in providing freshness guarantees. By "fresh", we mean the outsourced data is up-to-date, that is, all database update operations have been correctly carried out by the service provider. Although researchers have started to realize that having freshness guarantee is important, there has not been any systematic study of this issue so far. In this paper, we perform extensive study on how to provide freshness guarantees.

We provide freshness guarantees for both types of approaches [10, 15]. For the authenticated data structure based scheme, we add timestamps to data signatures to provide freshness guarantees. For the probabilistic approach, we add fake insertion, deletion, and update operations to provide freshness guarantees. In both of the schemes, there is a trade off between the level of the freshness guarantees and the overhead.

Overall, we make four contributions in this paper. First, we perform a thorough and systematic study on how to provide freshness guarantees for outsourced databases. Second, we propose architecture to provide freshness guarantees for both authenticated data structure based and the probabilistic based approaches. Third, we implement prototypes of our solutions and perform experimental studies. We find that reasonable freshness guarantees only add around 5%−20% overhead for authenticated data structure and only 25% of overhead for probabilistic based approach. Moreover, we quantified the trade off between the level of freshness guarantees and the overhead, we found that we can reduce the overhead significantly for applications that can tolerate relatively low level of freshness guarantees.

The rest of the paper is organized as follows: In section 2, we cover the background of this topic. In section 3 and 4,

[*]This research was partially supported by the grants from the Natural Science Foundation of China under grant number 60573091; China 863 High-Tech Program(No. 2007AA01Z155); China National Basic Research and Development Program Project (No. 2003CB317000); Program for New Century Excellent Talents in University(NCET).

EDBT'08, March 25–30, 2008, Nantes, France.
Copyright 2008 ACM 978-1-59593-926-5/08/0003 ...$5.00.

we describe our architecture to provide freshness guarantees for authenticated data structure based approaches and probabilistic-based approaches. In section 5, we present our experiment results. Section 6 discusses related work and section 7 summarizes the paper.

2. BACKGROUND

In this section, we provide an overview of security issues in outsourced databases. Our focus is on query integrity of outsourced databases. We discuss several aspects of query integrity and describe primitives, mechanisms, and approaches to implement query integrity.

2.1 Security in Outsourced Databases

The main obstacle for wide acceptance and deployment of database outsourcing is security concerns. As the databases are hosted by third party service providers, it is no long reasonable to assume that the server hardware and software can be trusted. Service providers may peek into the confidential information or provide misleading query responses for various reasons. But more frequently, a third party service provider can host software from many parties and it could be difficult to administrate the hosting environment securely, which can open up vulnerabilities that may be susceptible of risks ranging from insider attacks, outside hacking, and human errors.

There are two aspects of security concerns: privacy and integrity. By privacy in database outsourcing, we mean that anyone who is not authorized to access, including those who control the server software and hardware, cannot peek into the confidential data stored in the database. By integrity in database outsourcing, we mean that servers cannot make up query results and query results must be generated by correctly processing of queries over the most up-to-date underlying data. In this introductory section, we first discuss privacy and then integrity.

2.2 Privacy

Data privacy is generally achieved through encryption. In outsourced databases, we have an additional requirement, that is, we must be able to execute queries efficiently. Various database encryption schemes [7, 2] have been proposed for efficient query processing. The ultimate goal of the research in this field is to make queries in encrypted databases as efficient as possible while preventing an adversary from learning anything about the data. Privacy research in outsourced database complements our work and our solutions can leverage progress in the privacy research.

2.3 Integrity

Although privacy in outsourced database has been extensively researched, database research community has just started to study integrity of outsourced database recently. There are three components in integrity of outsourced databases: authenticity, completeness, and freshness.

Authenticity means that the query result is generated from original database tuples. Authenticity is generally achieved by attaching a signature to a tuple or part of a tuple. When a query is processed by the server, signatures are returned along with the content of the query result to the client. As the signature is unforgeable, the client can check the signature for authenticity.

Completeness means that a query is executed over all the data and the query result thus contains all the tuples that satisfy the query. Note that it is possible to meet authenticity requirements without meeting the completeness requirement by executing the data over a subset of the data.

Freshness basically means that the queries are executed over the most up-to-date data. It is challenging to provide freshness guarantees because old data are still valid data at some point of time.

Up to this point of time, there does not exist any system that can provide freshness guarantees even though there are several systems that can provide both authenticity and completeness guarantees. Those systems either use the authenticated data structure based approach or use the probabilistic based approach. In this paper, we investigate how to add freshness guarantees to both of the approaches. Having a basic understanding of the two approaches is essential for understanding the rest of the paper. In the following two subsections, we provide a detailed description of the two approaches.

2.4 Authenticated Data Structure Based Approaches

In authenticated data structure based approaches, completeness guarantees are provided by signing the sorted list of date attribute values. In response to a query, the tuples along with a part of the signature on the sorted list are returned to the client. The client can verify that the query result contains all the tuples matching the selection criteria by checking the part of the signature.

We illustrate how this approach works by a simple example. Suppose that a table contains 8 tuples, t_1, t_2, …, t_8, the value of attribute A of those tuples are a_1, a_2, …, a_8, and the list of a_1, a_2, …, a_3 is in the ascending order. Further suppose that a client C issues a query that selects all the tuples whose attribute A value is between a_s and a_e. Assume that $a_3 < a_s < a_4$ and $a_5 < a_e < a_6$. The query result will be a_4 and a_5. In a naïve method, the server can send back the whole signed list of a_1, a_2, …, a_8 and the client can verify that only a_4 and a_5 meet the selection criteria. However, sending the whole attribute value list and checking the whole list can be prohibitively expensive. Note that it is not necessary for the client to know the whole list to verify that t_4 and t_5 are the complete query result. It is sufficient for the client to know that part of sorted list from a_3 to a_6 is a_3, a_4, a_5, a_6. This can be implemented by attaching a pointer to each attribute value a_i that points to the attribute value that is immediately next to a_i and signing the pointer. In our case, a_2 points to a_3, a_3 to a_4, a_4 to a_5, and a_5 to a_6. These signed pointers are returned to the client along with the tuples t_4 and t_5. The client can then verify that there does not exist any tuple whose attribute A value is between a_4 and a_5 and there does not exist any tuple whose attribute value is bigger than a_5 or smaller than a_4 and in the range between a_s and a_e.

The most recent work in this area is by Li et. al. [10]. In that work, the authors use PKI to authenticate the database. To reduce the cost of signature while preserving the ability of verifying parts of the data, a Merkle hash tree can be built upon the underlying data and only a signature of the root node is needed for the authentication.

The drawback of this approach is that it can only handle simple queries. In a complex query such as a join with the condition that two attribute values are equal, the complete list of attribute values is needed to verify the completeness, which makes this approach impractical.

2.5 The probabilistic approach

In the probabilistic approach [15], we inject a small number of *fake tuples* into the outsourced database. Data encryption ensures that the service provider cannot differentiate fake tuples from the original tuples in the database. A query sent to the service provider is executed against the entire database, and fake tuples that satisfy the query will show up in the query result. By analyzing these fake tuples, the system can provide probabilistic guarantees that the service provider is doing a correct and honest job.

An interesting question is what kind of fake tuples should be mixed up with the original data in the outsourced database. In order to analyze the query result, the client must be able to know what are the fake tuples in the query result. One approach is to create a copy of the fake tuples and store them on the client side. While queries are executed by the service provider against the outsourced database, the client executes the same query on the local data and compare both the results. However, this approach requires that the client manages a local database of the fake tuples, which undoes the purpose of database outsourcing.

A better approach is to use a set of deterministic functions to generate the fake tuples [15]. Although tuples generated by deterministic functions may exhibit certain patterns, it is provable that the service provider cannot distinguish fake tuples from original tuples in the database. Thus, instead of maintaining a copy of fake tuples on the local site, the client remembers the deterministic functions. Given a query and the deterministic functions, the client can find the fake tuples generated by the deterministic functions that satisfy the query.

We illustrate how the probabilistic based approach works with an example. We assume that a query has the following form.

SELECT * FROM T
WHERE $T.A$ BETWEEN a_1 AND a_2 AND
$T.B$ BETWEEN b_1 AND b_2 AND ...

Assume that the server returns R_Q for a query Q. The client knows that R_Q should include certain fake tuples by checking the deterministic fake tuple generating function. Then, if any of them is absent, we know immediately that there is an attack.

Auditing whether all fake tuples covered by Q appear in R_Q can be a costly process, for the client needs to join R_Q with its own "copy" of fake tuples to get the result. To alleviate the cost, we use the *header* column information for each tuple t to easily find out the total number of fake tuples returned by the server for query Q.

Let $C_s(Q)$ be the set of fake tuples in R_Q, and let $C_c(Q)$ be the tuples computed on the client that satisfy Q. Note that it is easy to ensure that no duplicates exisit in $C_s(Q)$ and $C_c(Q)$. We have the following conclusion:

THEOREM 1. *If* $|C_s(Q)| = |C_c(Q)|$, *then* $C_s(Q) = C_c(Q)$.

PROOF. Assume to the contrary $C_s(Q) \neq C_c(Q)$. As $|C_s(Q)| = |C_c(Q)|$, $\exists t \in C_s(Q)$ such that $t \notin C_c(Q)$. But $t \in C_s(Q)$ means t is a fake tuple, whose authenticity is guaranteed by the encryption and the one-way hash function, and since t satisfies Q, t must appear in $C_c(Q)$. □

Theorem 1 enables the client to audit the *completeness* of R_Q by counting the tuples, which avoids the join operation. Now, if $|C_s(Q)| \neq |C_c(Q)|$, we know immediately there is a problem.

There are two advantages of using probabilistic based approaches. First, probabilistic based approach are server transparent in the sense that we do not need to modify database servers. Second, unlike the existing authenticated data structure based approach, probabilistic based approach can handle complex queries such as joins efficiently, which makes this approach more applicable.

3. AUTHENTICATED DATA STRUCTURE BASED APPROACH

In an authenticated data structure based approach, the whole database and the sorted list of each attribute are certified with signatures. When the server processes a query for a client, the server returns the query result along with a part of the signatures to allow a client to verify authenticity and completeness of the query result.

In order to provide freshness guarantees, it is no longer sufficient to verify that the signatures are valid. We also need to verify the signatures are the signatures of the most up-to-date data. The clients needed to be constantly notified of the current signatures of the data. Authenticated data structure based approaches typically aggregate the signatures or use Merkle hash tree to reduce the cost of generating and verifying signatures. We call the aggregated signature or the signature of Merkle hash tree's root node the root signature. It is sufficient for the client to know the current root signature.

3.1 The Basic Scheme

As the underlying data changes over the time, their aggregated signature also changes. Assume that a client C_A updates the database at time t that results in a signature S_t. The client can generate a signature $Certificate_{C_A}(S_t, t)$ and put it into the outsourced database to indicate that the current signature at time t is S_t. When another client retrieves the query result, it also retrieves the signature.

Note that if the database server is trusted, we only need to place a certificate whenever the root signature changes. However, this opens up potential attacks when the database server is not trusted. A malicious server can simply turn away all the updates on the data and new certificate.

To counter this kind of attack, we also assign an expiration time, δ, to such a certificate. So a client needs to place a certificate on the current signature every δ units of time no matter the signature has been changed or not. When another client sees a certificate on (S_t, t), it assumes that S_t is valid until $t + \delta$, which ensures that a client can see all the updates that happen δ units of time before the current time. δ can be tuned to provide various degree of freshness guarantees. Note that the clients need to have correct local time to make this scheme work. However, correct local time is required for many other applications such as the make utility.

3.2 Multiple Updating Clients

The basic scheme is sufficient if there is only one client that updates the database. When multiple clients update the database, the basic scheme cannot provide freshness guarantees.

For example, suppose we have three clients C_A, C_B, and C_C. C_A first makes an update which results in a signature S_1. Then C_B makes another update which results in a signature S_2. Assume that both of the clients then become idle. Each of the client would continue to certify the signature that results from their last update every δ units of time. When client C_C accesses the database, it would not be able to tell which signature is the latest signature.

To address this issue, we propose the following scheme: when a client certifies a root signature, it also provides a reference to the latest signature from another client. In our example, C_B certifies the tuple (S_2, S_1). When another client C_C accesses the database, it can follow the reference to figure out the latest signature.

4. THE PROBABILISTIC APPROACH

In this section, we present a probabilistic approach for integrity auditing of update operations[1].

4.1 Outline

Recall that for read-only databases, we create a set of fake tuples and embed them into the database [15]. As the service provider or malicious attackers cannot distinguish a fake tuple from a real tuple, we are able to obtain a probabilistic integrity assurance for SELECT queries.

It is conceivable that there is a relationship between the percentage of fake tuples in the database and level of assurance we can provide [15]. Figure 1 shows that for a dataset of $N = 1,000,000$ tuples, with an additional of 5% to 50% fake tuples, the probability of escaping detection when 1 to 100 tuples are deleted from the database. It shows that the probability of escape decreases sharply when the number of fake tuples or deletion increases. In particular, when the fake tuples are more than 10% of the original data, and more than 50 tuples are deleted, it is close to impossible for the attacker to escape from being caught by the randomized approach.

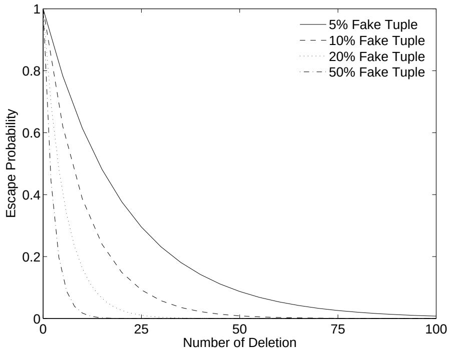

Figure 1: Probability of malicious attacks escaping detection. The database has 1,000,000 records.

Our approach for providing freshness guarantee is based on the same mechanism. When fake tuples in the outsourced database change over time in a deterministic way, it allows us to check whether update queries are performed correctly by the service provider. Note that our algorithm cannot provide freshness guarantee when there exists a malicious client that collaborates with the malicious server. However, in the outsourced database model, clients are from the enterprise that outsourced the database service and thus are assumed to be in a trusted domain. Moreover, we do not consider range deletion in this paper.

Architecture

We are interested in providing query integrity assurance for insert and delete operations. Before we dive into details, we reveal the architecture of data services outsourcing. In Figure 2, we show multiple clients, one trusted proxy server, and one service provider. In this multiple client scenario, the trusted gateway exists to coordinate clients for deterministic auditing, which we will discuss in detail. Usually, the trusted gateway forwards all clients' queries to the server at the service provider side, but at proper time slot decided by our algorithm, it also sends some queries to the service provider for the purpose of auditing. Note that all data services are provided by the server. Neither the client nor the trusted gateway maintains any database on its own.

Our Approach

To audit the integrity of INSERT and DELETE operations, we create fake INSERTs and DELETEs. These audit operations insert new fake tuples into the outsourced database and delete old ones from the outsourced database. By checking whether fake tuples that should be in the outsourced database actually show up in query results, we can find out whether previous fake operations have been correctly and honestly carried out. Furthermore, if we ensure that neither malicious attackers nor the service provider can distinguish

[1] In this work, we focus on auditing INSERT and DELETE operations in outsourced databases, UPDATE is considered as a combination of INSERT and DELETE.

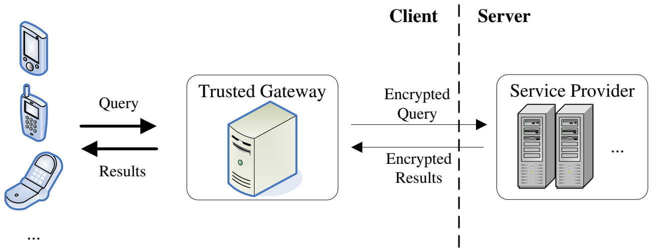

Figure 2: The architecture of query integrity auditing.

fake operations from real operations, we can provide a probabilistic assurance of all INSERT and DELETE operations.

To make the above approach work, we are faced with the following challenges. At any time t, in order to perform integrity audit, each client must know what are the current fake tuples in the outsourced database. This means each client must know the accumulated effect of all fake INSERTs and DELETEs starting from time 0. Furthermore, we must ensure our scheme is provable secure. In particular, malicious attackers cannot distinguish fake operations from real operations. We give the security proof in this section.

4.2 Deterministic Fake Operations

Recall that for static databases, in order to avoid storing a copy of fake tuples on the client side, we use a function determined by a randomly chosen key to generate a deterministic set of fake tuples. The benefit is that we only need to remember the randomly chosen key instead of the tuples generated by the function.

We want to apply this technique on non-static databases. To provide freshness assurance, we not only embed fake tuples in the outsourced databases, but also send fake INSERT and DELETE operations to the service provider to manipulate the fake tuples. As a result, the set of fake tuples in the outsourced database changes over time.

As for static databases, to provide probabilistic integrity assurance, we need to know what are the fake tuples in the database. But we do not want to store a local copy of fake tuple, nor do we want to record all historical fake INSERTs and DELETEs to derive the current fake tuples.

Our approach is to devise a mechanism to make all fake operations "deterministic". Because of the determinacy, each client inherently knows i) when fake operations take place, and ii) what are the fake tuples being inserted or deleted by each fake operation. This tells each client what are the current fake tuples in the outsourced database. In comparison, for static databases, we only need one snapshot of the fake tuples in the outsourced database, as they do not change over time. In our approach, we rely on determinacy to encode the entire history of fake operations and their effects.

More specifically, our scheme is a triple $(\mathcal{FS}, \mathcal{T}, \mathcal{H})$, where

- $\mathcal{FS} = \{f_1, \cdots, f_n\}$ is a set of functions, where each f_i is determined by a randomly chosen key, and generates a deterministic set of fake tuples.
- $\mathcal{T}$ is a function determined by a randomly chosen key, and it prescribes when fake operations are sent to the service provider. More specifically, let t_i be the last time we submit fake operations, then the next time we submit fake operations will be $\mathcal{T}(t_i) = t_{i+1}$.
- $\mathcal{H}$ is a function determined by a randomly chosen key, and it maps a function $f \in \mathcal{FS}$ and a time stamp t to a real value between 0 and 1, that is $\mathcal{H}(f, t) \in [0, 1]$. It decides when f is used to generate fake tuples.

Intuitively, at any time, the fake tuples in the outsourced database are generated by a subset of functions in $\mathcal{FS}$. At certain time slots decided by function $\mathcal{T}$, we turn on or turn off certain functions decided by function $\mathcal{H}$. The effect is that we will generate a deterministic sequence of fake INSERTs and DELETEs at deterministic time slots to change the fake tuples in the outsourced database in a deterministic way.

Specifically, the current fake tuples are collectively generated by a subset of functions in $\mathcal{FS}$. If function f_i is in the subset, we say f_i is *in effect*, meaning the current outsourced database contains fake tuples generated by f_i. Function $\mathcal{T}$ generates a deterministic series of time slots $\{t_1, \cdots, t_k, \cdots\}$ where the content of the subset changes. The function $\mathcal{H}$ decides, at each time slot t_k, which functions of $\mathcal{FS}$ will become in effect, and which will cease to be in effect. Let $\theta \in [0, 1]$ be a user-defined threshold. At time t_k, function f_i become or keeps in effect if $\mathcal{H}(f_i, t_k) \geq \theta$. It is clear that parameter θ can effectively control the size of total fake tuples in the outsourced database.

Thus, between time slots t_k and t_{k+1}, the fake tuples in the outsourced database should be those generated by functions in

$$C(t_k) = \{f_i | f_i \in \mathcal{FS}, \mathcal{H}(f_i, t_k) \geq \theta\}$$

Clearly, as long as the clients remember $\mathcal{FS}$, $\mathcal{T}$, and $\mathcal{H}$, they can derive the current fake tuples easily.

To enforce such changes in the outsourced database, at each time slot t_k, when the functions in effect change from $C(t_k)$ to $C(t_{k+1})$, the trusted proxy sends a series of INSERT operations to add fake tuples generated by function f_j where $f_i \in C(t_{k+1}) - C(t_k)$, and a series of DELETE operations to remove fake tuples generated by function f_j where $f_j \in C(t_k) - C(t_{k+1})$. Thus, by making the fake tuple generation functions valid and invalid over time, we have a deterministic way to audit update operation.

Algorithm 1 outlines the procedure. In Algorithm 1, at each time slot designated by $\mathcal{T}$, we check if a function f is turned on or off by $\mathcal{H}$. The set `EffectiveSet` records all functions that are in effect. Whenever functions are added to or removed from `EffectiveSet`, the trusted gateway will send fake insert or delete operations to the service provider to add or remove corresponding fake tuples. Thus, the fake tuples in the outsourced database should always be those generated by the functions in `EffectiveSet` and nothing more, as long as all fake operations are performed correctly by the service provider.

As we show in Figure 1, in order to achieve good integrity

Algorithm 1 Update Auditing Algorithm

```
$t = \mathcal{T}(0)$
EffectiveSet = {}
while true do
    if $current\_time = t$ then
        for all $f \in \mathcal{FS}$ do
            if $\mathcal{H}(f,t) \geq \theta$ and $f \notin$ EffectiveSet then
                EffectiveSet ← EffectiveSet $\cup\{f\}$
                for all Tuple $t$ generated by $f$ do
                    INSERT $t$
                end for
            else if $\mathcal{H}(f,t) < \theta$ and $f \in$ EffectiveSet then
                EffectiveSet ← EffectiveSet $-\{f\}$
                for all Tuple $t$ generated by $f$ do
                    DELETE $t$
                end for
            end if
        end for
        $t \leftarrow \mathcal{T}(t)$
    else
        wait()
    end if
end while
```

assurance, the number of fake tuples we embed into outsourced databases usually account for 10% to 20% of the entire data [15]. Adding or removing large amount of fake tuples can be a costly operation that will not finish instantly. Also, if the total number of functions in $\mathcal{FS}$ is small, then it is also likely that there might exist some hiatus where the database contains no fake tuples. Clearly, during such hiatuses, no auditing can be performed.

To solve this problem, we partition the feature space into a set of grids, and for each grid, we use an independent triple $(\mathcal{FS}, \mathcal{T}, \mathcal{H})$ to generate fake tuples and perform auditing. A range query may access multiple grids in the feature space, and auditing is performed in each grid independently.

The benefit of griding is obvious. Switching to use another set of fake tuples is much easier and more efficient. Globally, it will be much more unlikely that there exist a hiatus during which the outsourced database does not contain any fake tuples.

4.3 Designing deterministic functions

Given the triple $(\mathcal{FS}, \mathcal{T}, \mathcal{H})$, each client knows what are the current fake tuples, and how they evolve in the outsourced database. However, what is the criteria in choosing what $\mathcal{FS}$, $\mathcal{T}$, and $\mathcal{H}$ to use?

We should choose functions such that the service provider or the malicious attackers cannot tell fake tuples from original tuples, and fake operations from real operations. The problem of choosing the right set of $\mathcal{FS}$ has been discussed in [15]. The set $\mathcal{FS}$ can contain any family of functions, e.g., linear functions, quadratic functions, etc. It is conceivable that tuples generated by such functions exhibit certain patterns. However, with encryption, it is proved that it is computationally infeasible to detect such patterns [15]. As far as $\mathcal{H}$ is concerned, we can just use a pseudorandom function which turns on and turns off functions in $\mathcal{FS}$ at given time slot in a uniform way.

We focus on finding the right function $\mathcal{T}$. Function $\mathcal{T}$ determines the distribution of fake INSERTs and DELETEs over time. If the distribution differs significantly from the distribution of user INSERTs and DELETEs, then there is a big chance that the server or the malicious attackers can detect such patterns and tell fake operations from real ones. This enables the attackers to cheat and elude our detection: they can carry out only fake operations and ignore real ones.

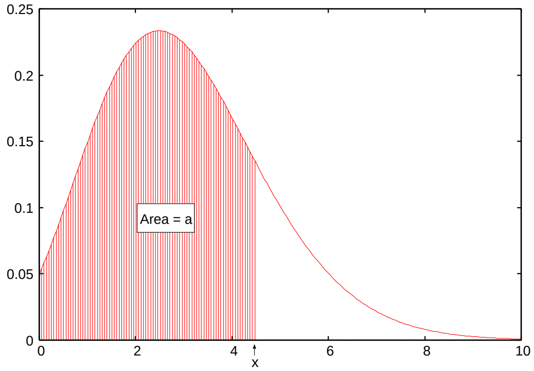

Figure 3: The next deterministic fake insertion/deletion time.

Thus, it is essential that the distribution generated by function $\mathcal{T}$ is similar to the distribution of real INSERTs and DELETEs. In the following, we show how this is achieved. In the next section, we give a security proof of our method.

Assume we know the distribution of real INSERTs and DELETEs. The distribution can be in the form of a histogram which records the number of INSERTs and DELETEs for any interval of time. For simplicity of discussion, let us assume that it is a Poisson distribution. Figure 3 shows a Poisson distribution with $\lambda = 3$.

How do we generate a series of INSERTs and DELETEs whose distribution is close to that shown in Figure 3? Given a secret key, and assume the i-th time slot is t_i (and also assume $t_0 = 0$), each client uses the following steps to generate the $(i+1)$-th time slot t_{i+1}:

1. Use the secret key to encrypt number i. Assume the results of encryption distribute uniformly between $[0, 1]$, then this essentially gives the client a pseudo random value a in the range of $[0, 1]$.

2. Since the curve in Figure 3 represents a probability density function, the total area under the curve is 1. Find x such that $\int_0^x \frac{\lambda^t e^{-\lambda}}{t!} dt = a$, that is, a is the area under the Poisson curve between 0 and x.

3. $t_{i+1} \leftarrow t_i + x$

Given that the secret key is known to each client, the clients generate a same series of time slots. Also, it is easy to show that the time intervals in the series of timestamps $\{0, t_1, t_2, \cdots, t_k \cdots\}$ generated in the above steps observe the distribution shown in Figure 3. We can easily substitute the Poisson distribution in our discussion by a real distribution represented by histogram. In the next section, we show that our approach is provable secure – it is computationally infeasible to tell fake INSERT and DELETE operations from real operations.

4.4 Security Proof

Here we give a proof of our scheme based on a simple model. With this simple model, our data outsourcing scheme is shown to be a provably secure scheme, that is, the security of our scheme can be reduced to the security of the underlying encryption primitives. The underlying security primitives used in our scheme includes pseudo random generators. Our proof uses the standard techniques used in the provable security literature [4, 3].

We first introduce some necessary definitions for developing our theorem and proof.

DEFINITION 1. *Function family*
A function family F is a finite collection of functions together with a probability distribution on them. All functions in the collection are assumed to have the same domain and the same range.

There is a set of "keys" and each key names a function in F. We use F_k to denote the function selected by key k in the function family F.

DEFINITION 2. *ϵ-distinguisher*
Suppose that F_0 and F_1 are two function families. Let $\epsilon > 0$ and let f_0 and f_1 be two functions selected from F_0 and F_1 uniformly randomly. A distinguisher $\mathcal{A}$ is an algorithm; given a function, $\mathcal{A}$ outputs 0 or 1 as it determines whether the function is from F_0 or F_1. We use $Adv_{\mathcal{A}}$ to denote $\mathcal{A}$'s advantage in distinguishing F_0 from F_1.

$$Adv_{\mathcal{A}} = |Pr[\mathcal{A}(f_0) = 1] - Pr[\mathcal{A}(f_1) = 1]|$$

We say algorithm $\mathcal{A}$ is an ϵ-distinguisher of F_0 and F_1 if $Adv_{\mathcal{A}} > \epsilon$.

DEFINITION 3. *(t, ϵ)-pseudo random*
A function family $F : U \to V$ is (t, ϵ)-pseudorandom if there does not exist an algorithm $\mathcal{A}$ that can ϵ-distinguish a pseudorandom function from a truly random function. Here A is allowed to use no more than t computation time.

A stream cipher can be abstracted as a pseudorandom generator for uniform distributions. More complex distribution can be constructed by applying function map to this basic pseudorandom number generator.

In our scheme, the time slots for fake insert and delete operations are selected by a pseudorandom function G. We show that an adversary cannot distinguish such operations from real insert and delete operations from the clients. Specifically, let R be the time distribution of real insert and delete operations. Our goal is to show that our scheme is secure even if we only know the exact distribution R approximately. The following theorem shows that an approximate distribution can still work if the divergence from the true distribution is small enough.

THEOREM 2. *If G is a $(t, \epsilon)-$secure pseudorandom function then G and R is $(t, e + E_G) -$indistinguishable, where $E_G = \int_0^T (g(x) - r(x))p(x)dx$, where g is the probability density function of the output of G, r is the probability density function of the output of R, $p(x) = 1$ if and only if $g(x) > r(x)$, and $p(x) = 0$ otherwise.*

PROOF. (Sketch) Define G' as the truly random distribution with the same probability density function as G. The goal is to show that G is indistinguishable from R. We first show that G is indistinguishable from G'. We then show that it is difficult to distinguish G' from R.

For any given algorithm A that distinguish these distribution, let x in X denote a time point, 0 be the start time point, and T be the end time point.

$$\begin{aligned} AdvA &= |Pr[A(G') = 1] - Pr[A(R) = 1]| \\ &= |\int_X Pr[A(G') = 1|Output(G') = x]Pr[Output(G') = x]dx \\ &\quad - \int_X Pr[A(R) = 1|Output(R) = x]Pr[Output(R) = x]dx \end{aligned}$$

This function is maximized when $A(G') = 1$ and $A(R) = 0$ whenever $Pr[Output(G') = x] > Pr[Output(R) = x]$, which equals to $AdvA \leq \int_0^T (g(x) - r(x))p(x)dx$. This is exactly E_G.

$$\begin{aligned} AdvA &= |Pr[A(G) = 1] - Pr[A(R) = 1]| \\ &= |Pr[A(G) = 1] - Pr[A(G') = 1] + \\ &\quad Pr[A(G') = 1] - Pr[A(R) = 1]| \\ &\leq |Pr[A(G) = 1] - Pr[A(G') = 1]| + \\ &\quad |Pr[A(G') = 1] - Pr[A(R) = 1]| \end{aligned}$$

By definition, $|Pr[A(G) = 1] - Pr[A(G') = 1]| \leq \epsilon$. Thus, $AdvA \leq \epsilon + E_G$. □

5. EXPERIMENTS

In this section, we evaluate the performance of our integrity auditing scheme.

5.1 Experiment setup

We use two Pentium IV 2.0GHz PCs. Each of them has 512MB RAM and a 80GB Ultra-ATA/100 hard disk. One simulates clients and the other simulates the server. The two machines are connected with a local Ethernet network running at 100MBps. We use IBM DB2 v8.2 to store data on the server side. We use an existing library for authenticated index structure [9]. Authentication index based experiments are coded in C++ and compiled with GCC 3.4.4. Probabilistic based experiments are developed in JAVA with the JDK/JRE 1.5 develop kit and Runtime Library.

Data setup. We generate a synthetic dataset in our experiments to evaluate the performance of the authentication data structure based approach and the probabilistic approach. The synthetic dataset is composed of 200,000 randomly generated one dimensional numerical tuples.

Fake tuple setup. We use equal-width histogram with 1000 buckets to catch the distribution of the synthetic data value. And an initial set of 2000 deterministic linear functions are randomly selected to generate fake tuples.

Query setup. Query used in our experiments are composed of three parts, unit selection query, of which the value predicate interval has the same width as a histogram bucket, randomized point insertion query and randomized point deletion query.

5.2 Server Side Performance Study

In this section, we compare performance of our probabilistic integrity auditing scheme against the authenticated index based auditing scheme at the server side.

In our experiment, we measure the throughput and CPU utilization at the server side for both the authenticated index based approach and our probabilistic approach for a workload of 300 query requests.

In figure 4, we show the server side throughput performance with the number of requests we simultaneously send to the server varying from 10 to 20.

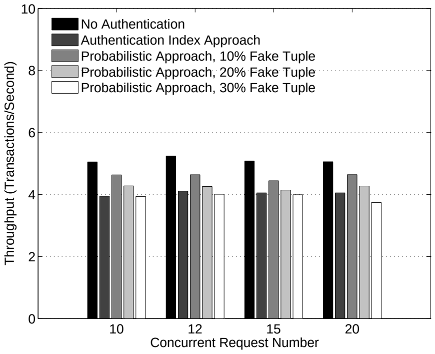

Figure 4: ThroughPut Experiment

From the figure, it can be easily seen that our probabilistic auditing scheme has only small overhead over throughput when we have a reasonable amount of fake tuples inserted into the database. We can improve security performance of our probabilistic auditing approach by increasing the number of fake tuples, but as a result we have to sacrifice query performance as a cost. So basically, according to different application requirements, we can trade-off between efficiency and the level of integrity assurance by adjusting the number of fake tuples we have in the database.

We also analyzed the CPU utilization performance of both approaches at the server side. In order to have a fair comparison, we use the same work load, and ensure that the time each approach uses to finish the whole workload is roughly the same under different request rate. To achieve this goal, we can simply adjust the time interval between two continuous requests. And then we analyze the average CPU utilization during the execution of the workload. Experiment result is shown in Figure 5.

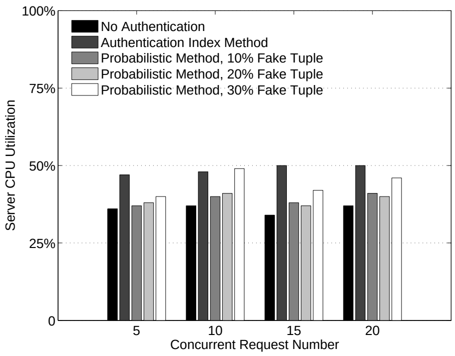

Figure 5: Server Side CPU Utilization Experiment

From the figure we can see that when only a reasonable amount of fake tuples present in the database, our probabilistic auditing scheme will not cause much burden for the server side. And besides our probabilistic auditing scheme is server transparent, which means we do not need to implement none standard authentication structures on the server side, we can easily get an conclusion that our probabilistic approach is both efficient and practical to implement.

5.3 Client Side Performance Study

We also evaluate the client side performance of authentication index based approach and our probabilistic approach.

Figure 6 shows the average processing latency of client side with different request rate. From the figure, we can easily find out that the additional inserted fake tuples have only a small affection on the overall performance. And when a reasonable amount of fake tuples are inserted into the table, our probabilistic based approach have better client side performance than authentication index based approach.

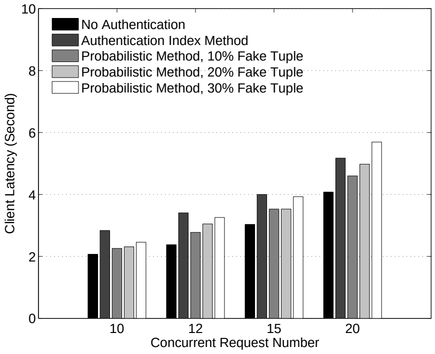

Figure 6: Client side overall performance experiment

And from figure 7, which shows detailed client side processing latency information, we can figure out that our probabilistic auditing approach have very low client side verification cost, and the synchronization cost caused by our deterministic updating processing also has a very small affection on the overall performance.

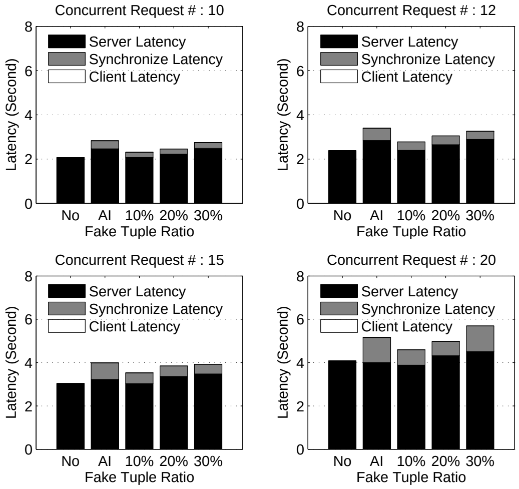

Figure 7: Client side performance study

5.4 Freshness auditing cost study

Here in this section, we study the cost of performing freshness auditing of both of the approaches. We setup a dataset of 200000 tuples, and measure the cost of freshness auditing using the average computation cost for both schemes after a set of tuples have been updated.

In figure 8, we show after a certain number of tuples have been inserted, the computation cost needed of both approaches to guarantee the freshness. For authentication index based approach, we may need to update the index accordingly, recompute the digest, encrypt and send to the server side after every insertion. And for our probabilistic based approach, we need to insert a set of fake tuples into the database along with the normal insertion. From the figure, it's easy to figure out that our probabilistic approach may have comparable computation cost with authentication index based approach when the fake tuples we insert have a reasonable size. As the number of fake tuples increases, we may have a high insertion cost, but remember the more fake tuples we insert, the more secure our approach will be. In practice, our approach can offers users flexible choices to trade off between performance and security according to the application scenario.

Similarly, In figure 8, we show after a certain number of tuples have been deleted the computation cost needed for both approaches. Our probabilistic based approach need to delete fake tuples while performing normal deletions, but unlike insertion, deletion is a high cost operation as it needs scanning the whole table to locate all the target tuples to be deleted. So a set of point deletion may cause great performance degration. A simple solution is to group several deletions into a single one using "OR" predicate. In figure 9, for probabilistic based approach, we experimented using different group size, like we group every 20(40) deletions into a single deletion. An extreme case is that we group all fake deletions into a single one, as shown in figure 9, we get a very good performance gain.

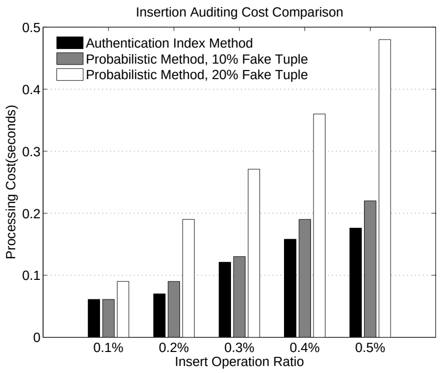

Figure 8: Insertion auditing cost

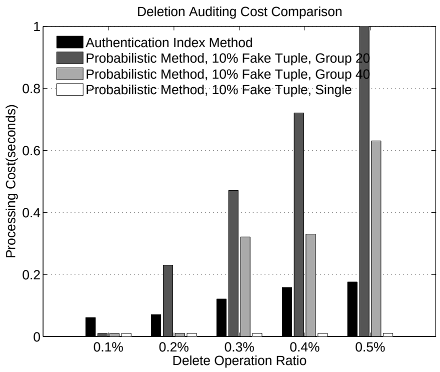

Figure 9: Deletion auditing cost

6. RELATED WORK

There are two security concerns in database outsourcing: *data privacy* and *query integrity*. Although data privacy has been extensively studied, so far there does not exist any system that provide complete integrity assurance.

Hacigümüs et. al. [8] first studied security issues in the scenario of database outsourcing. That work focuses on the *privacy* aspect of the outsourced database, in particular, efficiency of various encrypting schemes using both hardware and software encryption. However, that work does not consider the problem of data integrity.

The pioneering work on the problem of integrity [5, 12] focuses on the authentication of the data records, that is, the *authenticity* aspect of the integrity. Devanbu et. al. [5] authenticates data records using the Merkle hash tree [11], which is based on the idea of using a signature on the root of the Merkle hash tree to generate a proof of *correctness*. Mykletun et. al. [12] discussed and compared several signature methods which can be utilized in data authentication, and they identified the problem of *completeness*, but did not provide a solution.

Some recent work [13, 10, 14] studied the problem of auditing the *completeness* aspect of the integrity. By explicitly assuming an order of the records according to one attribute, Pang et. al.[13] used an aggregated signature to sign each record with the information from two neighboring records in the ordered sequence, which ensures the result of a *simple selection query* is continuous by checking the aggregated signature. But it has difficulties in handling *multipoint selection query* of which the result tuples occupy a non-continuous region of the ordered sequence.

Besides, it can only handle a subclass of join operations efficiently, the *primary key/foreign key* join, because that the result of the join forms a continuous region of original ordered data can only be assured in this case. Other work [5, 10] uses Merkle hash tree based methods to audit the *completeness* of query results, but since the Merkle hash tree also uses the signature of the root Merkle tree node, a similar difficulty exists. The network and CPU overhead on the client side can be prohibitively high for some types of queries. In some extreme cases, the overhead could be as high as processing these queries locally, which can undermine the benefits of database outsourcing. Moreover, to ensure freshness, an additional system is needed to deliver the most up-to-date root signature to all the clients in a reliable and timely manner. It is unclear where such a system can be placed in an outsourced database environment, while the freshness problem could be solved naturally in our approach without additional requirements. In our approach, we just need to vary the inserted tuples according to the current time. We will prototype this functionality in our future work.

Sion [14] introduces a mechanism called the *challenge token* and uses it as a probabilistic proof that the server has executed the query over the entire database. It can handle arbitrary types of queries including joins and does not assume the underlying data is ordered. But their scheme cannot detect all malicious attacks, for instance, when the service provider computes the complete result but returns part of it for sake of business profit from a competition rival. Xie et. al.[15] proposed an improved probabilistic based approach that addresses those drawbacks. In that approaches, fake tuples are added to the encrypted outsourced databases. Clients check the fake tuples in query results for integrity.

7. CONCLUSION

The potentially explosive growth of database outsourcing is hampered by security concerns: privacy and integrity. Although privacy in outsourced databases has been extensively researched, there still does not exist a practical system that can provide complete integrity guarantees before this work. The freshness aspect of integrity guarantees is missed from all the previous systems. In this work, we perform a systematic study on how to add freshness assurance to two types of approaches for database outsourcing integrity: authenticated data structure based and probabilistic based. We have shown that we can provide complete integrity guarantees without sacrificing much performance.

8. REFERENCES

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

[2] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In Gerhard Weikum, Arnd Christian König, and Stefan Deßloch, editors, *SIGMOD Conference*, pages 563–574. ACM, 2004.

[3] Mihir Bellare. Practice-oriented provable-security. In Eiji Okamoto, George I. Davida, and Masahiro Mambo, editors, *ISW*, volume 1396 of *Lecture Notes in Computer Science*, pages 221–231. Springer, 1997.

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

[5] Premkumar T. Devanbu, Michael Gertz, Charles U. Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In Bhavani M. Thuraisingham, Reind P. van de Riet, Klaus R. Dittrich, and Zahir Tari, editors, *DBSec*, volume 201 of *IFIP Conference Proceedings*, pages 101–112. Kluwer, 2000.

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

[7] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki, editors, *SIGMOD Conference*, pages 216–227. ACM, 2002.

[8] Hakan Hacigümüs, Sharad Mehrotra, and Balakrishna R. Iyer. Providing database as a service. In *ICDE*, pages 29–. IEEE Computer Society, 2002.

[9] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. *Authenticated Index Structures Library.* http://www.cs.fsu.edu/ lifeifei/aisl/index.html.

[10] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *SIGMOD Conference*, pages 121–132. ACM, 2006.

[11] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989.

[12] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. In *NDSS*. The Internet Society, 2004.

[13] HweeHwa Pang, Arpit Jain, Krithi Ramamritham, and Kian-Lee Tan. Verifying completeness of relational query results in data publishing. In Fatma Özcan, editor, *SIGMOD Conference*, pages 407–418. ACM, 2005.

[14] Radu Sion. Query execution assurance for outsourced databases. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 601–612. ACM, 2005.

[15] Min Xie, Haixun Wang, Jian Yin, and Xiaofeng Meng. Integrity auditing of outsourced data. In *VLDB*, 2007.