Send Orders for Reprints to reprints@benthamscience.ae

322 *The Open Cybernetics & Systemics Journal,* 2014, *8,* 322-329

Open Access

Multi-tenant Data Authentication Model for SaaS

Lin Li[1,3], Lanju Kong[1,2], Qingzhong Li[1,2*], Zhongmin Yan[1,2] and Hui Li[1,2]

[1]*School of Computer Science and Technology, Shandong University, Jinan, P.R. China;* [2]*Shandong Provincial Key Laboratory of Software Engineering, Jinan, P.R. China;* [3]*School of Mathematical Sciences, Shandong Normal University, Jinan, P.R. China*

Abstract: In SaaS, most tenants rely on the service provider for data maintenance and computation. As tenants no longer possess their application and data locally, it is of critical importance for the tenants to ensure that their data are being correctly stored and maintained. However, the customized multi-tenants sharing storage mode makes it hard for tenants to guarantee their data integrity because multiple tenants' data is stored in one physical universal table and different data types may be stored into a flex column based on tenants' customization. Meanwhile to ensure performances of query, adequate pivot table is set up. These introduce new challenges to data integrity protection for tenants. This paper presents a review of the state of the art solutions and recent patents in the fields of data authentication, and puts forward a multi-tenant data authentication model (MTDA). MTDA is a composite structure that constructs pivot authentication tree (PAT) on the pivot table and combines it with signature set (S-set) built on universal table to ensure that malicious insiders can't modify the data in pivot table and universal table. The main contribution of MTDA is it can guarantee the tenant query result in one tree travels and return the verification object, corresponding to the result on pivot table and universal table. We demonstrate effectiveness of our model compared with direct adoption of the MB tree based approaches on pivot table and universal table through the experiment. MTDA shows a better performance on VO verification.

Keywords: Authentication, database, data integrity, multi-tenant, SaaS.

1. INTRODUCTION

Software-as-a-Service, *i.e.* SaaS [1], is an emerging model that allows tenants to outsource computation and storage of their data to external service providers. By leasing the service and giving the data to the service providers, the tenants can be relieved of the burden of computation and storage and pay more attention to their business. However, the service provider malicious insiders may violate tenant data integrity for to some benefits, they may delete, modify, fabricate tenant's data for ulterior motives. Since the service provider may not be trusted, or may be compromised, tenants need to be affirmed that their query results have both correctness and completeness. Correctness implies that the result data records indeed be the tenant's legitimate original data, and that they have not been tampered with in any way. Completeness requires that no qualifying records have been omitted.

However, there are some obstacles for the existing method to apply suitably on tenant data authentication in SaaS. First, most SaaS service providers adopt the single instance multi-tenancy strategy to take full advantage of resources such as hardware and database, and multiple tenants data is stored in one physical table such as universal table which means different data types may be stored into a flex column based on tenants' customization [2-4]. Second, in order to guarantee performances of query operations in large multi-tenants data set, adequate pivot table [5-7] for tenant data are set up to speed up the query process. Those data stored in pivot table should also be included in the integrity consideration of tenants' data.

Existing methods for query result authentication fall into two categories. One is Merkle hash tree based approaches [8-13] the other is signature based [14-18]. And the MHT based approaches shows advantage over the signatures based, due to the efficiency of hashing computing compared to signature computing. However it is improper to build an authentication structure such as MHT directly on the multi-tenant universal table, for there may be different data types contained in one column and some tenant may not want the integrity guarantee. And it needs multiple MHTs to be set up to guarantee both the universal table and pivot table, which leads to double travels of the MHT on verification object (VO) set up and reconstruct. Compared to the MHT approaches, the signature based ones are easier to guarantee isolation between tenants on the universal table for they require signatures on every record, but it brings lots of signing work to the pivot table which may contain several-fold data records compared to the universal table. So it is inefficient to use MHT based or signature based approaches alone for the tenant data integrity in SaaS.

Based on recent methods and patents in the fields of data authentication and secure storage, this paper puts forward a multi-tenant data authentication model (MTDA). The main idea of MTDA is to constructs MHT tree based authentication structure Tenant Pivot authentication tree (PAT) on the

*Address correspondence to this author at the High-tech Industry Development Zone Shunhua Road 1500, Shandong University, Jinan Shandong, China 250101; Tel: (86) (86)18663719666; Fax: 0531-88390081;
E-mail: lqz@sdu.edu.cn

1874-110X/14 2014 Bentham Open

pivot table for each tenant to ensure that malicious insiders can't modify the index data in pivot table, and MTDA combine PAT with signature set (S-set) built on universal table to ensure the integrity of tenant query results. In order to meet the different integrity requirement of different tenants, MTDA is independent with the index structures built on pivot table by setting up separated authentication structures instead of coalescing as the MHT into the data index. The MTDA tree can guarantee the tenant query result in one tree travels, while return the VO corresponding to the result on pivot table and universal table. We demonstrate effectiveness of our model compared with applying the MB tree based approaches directly on pivot table and universal table through the experiment.

The rest of this paper is organized as follows. The next section covers related works. In Section 3 we present the system model as our work basis. Section 4 introduces the MTDA model and Section 5 presents the experiment. And Section 6 gives the conclusion of this paper.

2. RELATED WORK

Nowadays, most of the existing methods for data integrity verification fall under two categories -MHT-based approaches and signature aggregation. Reference [10] utilize the Merkle hash tree to provide authentication. The owner builds a Merkle tree on the records in the database, based on the query attribute. Similar to the original MHT, the root digest is signed by the owner. The server returns the query result along with the necessary hashes for the client to reconstruct the root of the Merkle tree to verify the query results. And it proposes MB tree which combines B+-tree with MHT. Reference [11] introduces EMB tree for one-dimensional queries over disk-resident data. However those indexing authentication structures are not suitable to multi-tenant, because they incorporate the MHT into the data index while in SaaS the data index is shared by multiple tenants and some of them may not want the integrity guarantee, even all the tenants need the guarantee it is hard for those applied directly on the shared storage because there may be different data types contained in one column. Reference [12] proposes a partially materialized digest scheme in which they split the authentication structure from the data index and extend their work to the spatial database, but it does not apply for the multi-tenant circumstance.

The work in [15] proposed two signature schemes that enable aggregating multiple individual signatures into one unified signature, verifying which is equivalent to verifying ALL individual component signatures. The size of the aggregated signature equals that of a single plain digital signature (which is constant), irrespective of either the database size or the query reply size. The condensed multiple signatures can be verified almost as fast as an individual signature. Reference [16, 18] introduce approach based on signature aggregation and chaining to achieve authentication of query replies. With the database ordered on an attribute, the owner hashes and signs every three sets of consecutive data records. Posed a range selection query on the attribute, the server returns the qualifying data, along with hashes of boundary record. The signatures of all the result records are aggregated and placed into the verify object. Signature aggregation has a smaller proof and is amenable to concurrent updates. Reference [14] constructs a signature aggregation protocol that provides freshness and guarantees authentication for the basic relational operators. Compared to the MHT approach, the signature aggregation makes it easier to guarantee isolation between tenants for they require signature of every record, but it needs to chain the searching attribute to protect the completeness, if there are multiple searching attribute it need lots of consecutive data records to check the completeness. In our scenario, we need to consider the pivot table integrity along with the tenant data, since the pivot table stores the tenant index data only and there may be several attributes stored into pivot table, using signature method to protect the pivot table will lead to several-fold signatures and as to the tenant data in the universal table which leads to lot of resources wasted on storing those signatures.

Beyond those, Reference [8] focuses on the authentication of long-running queries on outsourced data streams and presents a CADS model to achieve correctness and completeness on continuous data streams. Reference [9, 19] mainly aims at handling aggregation queries (*e.g.*, SUM, MAX, etc.) and promoting authentication structures on SUM queries. Reference [20] inserted certain fake tuples into the real data and verified query integrity by checking the fake tuple in the result. Reference [21] presented the dual encryption approach, where certain data are encrypted with different keys and query integrity could be checked by "cross examination". All of these works well in their respective scenarios but could not be applied to the multi-tenant storage ideally. Reference [21] presented a formal security definition of query integrity in outsourced dynamic databases. Reference [22] focuses on the case that service providers are not always trustworthy and promote a meta-data driven data chunk based secure data storage model for SaaS to ensure the data integrity. But it also did not give an appropriate solution on how to guarantee both the pivot table and tenant data.

Besides these works, there are many patents on data authentication and secure storage. US Patent Application 20080115194 "Authentication of modified data" [23] introduces an authentication method that uses authentication information to authenticate the modified data. US Patent Application 20100031048 "Data Authenticator" [24] make use of user encoded result to authenticate target data based on signature. US Patent Application 20120222127 "Authenticating a web page with embedded javascript "[25] presents a method for detecting if a digital document (*e.g.* an HTML document) is changed by others than authenticated script code, the digital document can then at any time be compared with the most recent snapshot to verify if it is authentic. The patents on SaaS storage contain US Patent 8280874 [26] which puts forward a multi-tenant database using dynamic tuning of database indices, JP Patent 2013168044 [27] aiming to provide a SaaS management system capable of facilitating the management of the SaaS by a system manager. US Patent Application 20110126168 [28] provides a cloud platform for managing resources wherein the SaaS applications and customer data are stored logically and physically independent of the computing resources. US Patent 7693970 [29] presents secure shared storage infrastructure accessible by more than one customer in isolation.

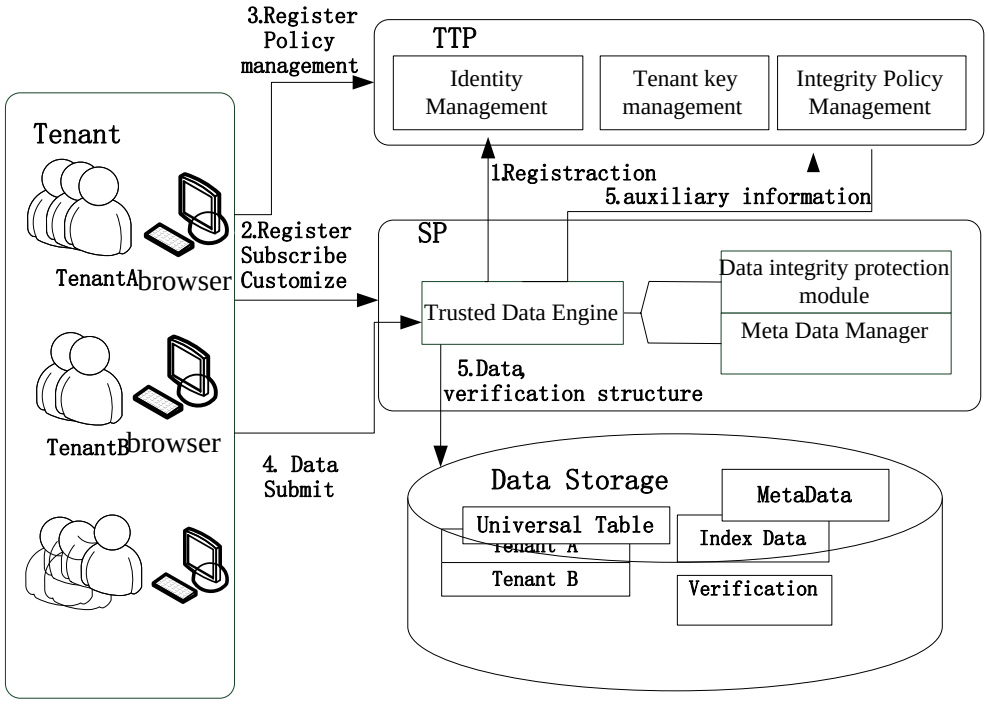

Fig. (1). System architecture.

Base on the above mentioned works, we know the MHT base approach and signature aggregation have their disadvantages to apply to multitenant sharing storage and MTDA combines them together to ensure tenant query integrity. For the tenant data in universal table, MTDA signatures each record and does not link the signatures on searching attribute, this can ensure the data correctness but not the completeness. The data completeness can be guaranteed by the pivot table where MTDA creates separate PATs on each query attribute. Based on the correctness and completeness of the pivot table, MTDA can ensure the completeness of tenant data.

3. PRELIMINARY

This section gives the system model, attack model and problem description as our work basis.

3.1. System Model

The system model for SaaS includes three entities: tenant, trusted third party and the service provider. The system architecture is shown in Fig. (**1**).

Tenants: an entity that customizes and consumes SaaS applications provided by service provider and relies on the service provider for data maintenance and computation.

The trusted third party (TTP): The TTP which has expertise and capabilities that tenant do not have is used to assist the tenant to manage their secret key information and provides integrity policy for data integrity verification of tenants. TTP mainly contains identity management, tenant key management and the integrity policy management. Identity management is used for prohibiting untrustworthy entity from getting tenant's secret key and the integrity policy. Secure tenant key management assists the tenants to manage their public key and secret key. Integrity policy management stores the customization integrity policy of the tenants.

Service provider (SP): an entity, which has significant computing resource and storage space to maintain the tenants' applications and data storage.

3.2. Attack Model

We assume that the *SP's* are not necessarily trusted because of the possibility of the malicious insider. Based on the researches and patents on trust platform [30-33], we assume that the SaaS platform can be trusted, that is, the applications implementation mechanism on the Service provider such as data engine shown in Fig. (**1**) is trusted. And we explore an integrity protection module in platform. This module can assist tenants on their data integrity customization and verify the data integrity with the help of the trusted third party. Besides, we assume that all communications go through a secure channel between the *SP*, *TTP* and tenants.

Based on the above assumptions, we concentrate on the analysis of malicious behavior from the *SP* malicious insiders which means that the Data storage of the Service provider may be violated by the malicious insider. For example, insiders may delete the record of a tenant in universal table, change the data item in pivot table or universal table or forge some non-existent record to tenants' data hosted by *SP* storage, which violates tenant data integrity.

On the SP side, the trusted data engine (DE) contains metadata manager and data integrity protection module (the other functions of the data engine have little to do with our research, so we ignore them here). The metadata manager manages the customization information of tenants based on their logical view. The data integrity protection module is used to establish the authentication structure of the tenants

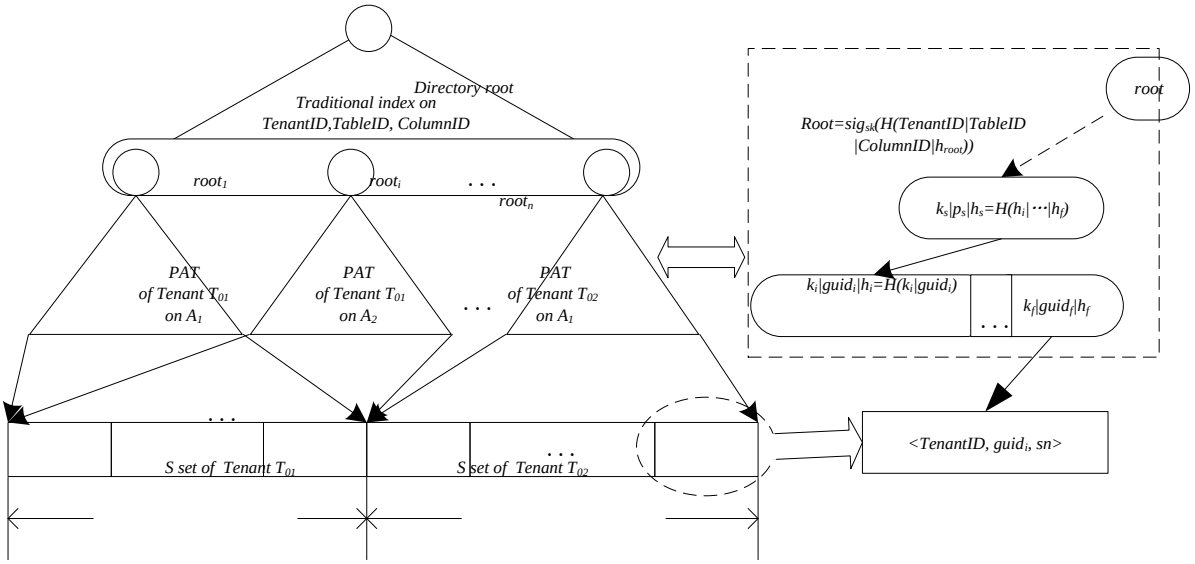

Fig. (2). Multi-tenant data authentication model for SaaS.

data though the customization of tenant data and verify the data integrity with the help of a trusted third party.

The Data storage stores the metadata, universal table, index data and the verification structures of tenant's data. Metadata is used to store the customization information of tenant. Based on the metadata data engine can convert the tenant logical view into the physical storage view in transparency. The universal table is a physical sparse table that stores all the tenant data in one table on the data node. Among which, GUID (Globally Unique Identifier) is treated as the physical primary key to achieve the rapid location in the record level. SP copies field data to be indexed into the pivot table and creates physical indices such as B+ tree on the table to accelerate the query speed. This index model could support customization and isolation characteristics of multitenant applications. The verification structures stores the verification structures of the tenant. In this paper, the verification structures of a tenant is built on the physical pivot table and sparse table records and stored in an external file on the data node.

3.3. Problem Description

Given a tenant T, suppose tenant *T* has a logical view $R(A_1, A_2, ..., A_n)$ and tenant customizes A_1 as the search attribute and registers query on it (Here we mainly aim at the case of searching key data type that does not have duplicates). The physical view in shared table corresponding to *R* containing records as $r(guid, T, value_1, value_2, ..., value_n)$, while $value_1, value_2, ..., value_n$ corresponding to $A_1, A_2, ..., A_n$. The values of A_1 are stored in pivot table as record *t(indexID, value, guid)*.

In this paper we consider equality and range selections. Equality selections are treated as a special case of range selections, so we focus on the latter. Suppose tenant T request a query *Q* such as $(SELECT * FROM R WHERE q_l< A1<q_u)$, where $q_l(q_u)$ is the lower(upper) bound of *Q*. The set of tuples that satisfy the query predicate is denoted by *SAT(Q)*, and the final answer returned is *ANS(Q)*. The process of queries of tenant in SaaS can be defined as follows: When tenant pose queries *Q* to SP, the data engine takes charge of query transformations and submits those queries to the data node: first data engine registers the query QP on the pivot table to get the middle result set, *Set(QP)*; then data engine registers query, QS, on the sparse table based on *Set(QP)* and obtains the result set *ANS(Q)*. So the DE obtains the *ANS(Q)* along with the VO. VO enables the data engine to verify the correctness and completeness of *ANS(Q)*. If the result set is legitimate, the data engine returns those data to the tenant through the application, else the data engine rejects the result set.

The problem of authenticating queries of tenant in SaaS is to define the appropriate verification structures on the sharing storage mode that ensures the data engine to verify the correctness and completeness of *ANS(Q)*, in other words *ANS(Q)* has been indeed computed solely from *SAT(Q)*. Because the sharing storage mode leads to a situation that the same column may contain numerous data types of multiple tenants' and tenant data loses their logical semanteme in shared table. So we may not be able to determine the consecutive pairs of tuples in shared table which have a definite order in tenant logical view given a searching attribute. To resolve this problem, we take a roundabout solution that combines the tenant data in sparse table with the index data together to establish the tenant data authentication which will be discussed in detail in the next section.

4. MULTI-TENANT DATA AUTHENTICATION MODEL

In this section, we proposed a solution for tenant data authentication in SaaS called tenant composite data authentication model(MTDA). Conceptually, MTDA is a composite tree and consists of two layers: the upper is *PAT* which is a MHT based authentication structure building for pivot table and the lower is *S set* with the corresponding signatures of the records in the universal table. The exact structure of MTDA is shown in Fig. (**2**).

PAT is an extended MHT with fanout f ($f \geq 2$) built for each tenant's searching attribute. Each leaf entry of PAT cor-

responds to the data record, *t,* in pivot table with the form <*k, guid, h* > where *k* is the searching key of PAT, *guid* is used to locate the signature that corresponds to *k* and the hash value $h=H(k|guid)$. For the internal nodes of PAT, each nodes is a triplet < *k, p, h*>, where *k* is the search key of the first record in the subtree of this node, *p* is a pointer to the corresponding child node and $h=H(h_1|...|h_f)$, where $h_1,..., h_f$ are the hash values of the node's children. Here we pick up those repetitions in pivot table like t*enantID, TableID* and *ColumnID* as public information and put the public information into PAT's *root*, *root*=< *TenantID, TableID, ColumnID k ,p, Root*>, where *Root* is the signature of PAT and $Root = sig_{sk}(H(TenantID|TableID|ColumnID|h_{root}))$, h_{root} is the digest of the PAT root, | denotes string concatenation and *sk* is tenant's private key and stored in TTP.

The *Signature set* **S set** contains all the signatures of the universal table with the form <*TenantID, GUID, SN*>, where the *TenantID* indicates the owner of the signature, *GUID* is the search key value and *SN* represents the digital signature. Here the *GUID* is corresponding with the leaf node of PAT. When tenant data stores into universal table *via DE*, the *DE* computes the signature $sn_{ri.guid}$ of each record r_i in universal table, $sn_{ri.guid} = sig_{sk}(h(guid|TenantID|tableID|a_{1i}|a_{2i}|...|a_{ni}))$. Based on the signature in *S*, the DE can verify the correct and provenance integrity of *ANS(Q),* but it can't discover if the malicious insider deletes the tenant record in universal table, for the records of *ANS(Q)* may be scattered in the share table based on the query attribute.

The PAT is, respectively, built on the searching columns on pivot table for each searching key. As the example shown in Fig. (**2**), there has two tenants with *TenantID* T_{01} and T_{02}. Tenant T_{01} specify the column A_1 and A_2 as the searching key, while the Tenant T_{02} specify the column A_1 as the searching key. The MTDA creates three PATs respectively on those columns, as shown in the red, orange and green dotted box and PAT of T_{01}-A_1 an T_{01}-A_2 share the signatures set of *Tenant* T_{01}. From Fig. (**1**), we can see that multiple PATs could be set up on the pivot table to adapt the query conditions of different attribute, while all the PATs of a tenant shares the *SignatureSet*. The server combines these individual signatures into a single aggregated signature and returns the result set comprised of the tuples along with the aggregated signature. Upon receipt, the tenant simply verifies the latter. This organization form could effectively adapt the changes of the tenant query conditions on different searching attribute in index table while introduce fewer influence to the *S*, because the PAT and *Signature Set* are two independent structures associated by *GUID*.

4.1. Verification Object Construction and Authentication of MTDA

As shown in Fig. (**2**), suppose tenant T_{01} register a range query $Q : [q_l,q_u]$ on A_1. On receiving the query request of the DE, the data manager performs two top-down traversals to locate the leaf node that is immediately before q_l and after q_u in the PAT, respectively. Those leaf are necessary to enforce the *set(QI)* has completeness and to ensure the DE dose not omit results at the range limits. Then the data manager locates the signature of *Set(QS)* based on PAT in the *signature set* S and computes the *VO* of the data result set *ANS(Q)* for the DE to check the integrity of the query results.

Table 1. VO construction algorithm.

```
Algorithm 1 MTDAVO(Query Q; PAT T ;Signature Set S; String VO)
Begin
VO=Null;
P=Null;
RangeTAB(T.root,[ql,qu])
Append boundary leaf node to VO
Append Root to VO
For every element in P Append sn to VO
 RangeTAB(Node N, Range R)
Aggregate signature returned by S
Begin
Append { to the VO
For each entry e in N
 If N is a intermediate node
  If e intersects the query range
   RangeTAB(e.prt, R)
  Else append e.h to the VO
 Else if e is leaf node
  Append t correspond to e.k to the VO
Append e.guid to P
Append } to the VO
End
End
```

Specifically, the *VO* includes: (1) the digest the pruned internal node of PAT, (2) the record of pivot table in every visited leaf node, (3) the lower (upper) boundary of an internal node, here we use {indicate the lower boundary and } present the upper; (4) the Root of PAT , and (5) the aggregated signatures of all the records that satisfy the *Q* in *S*. The VO construction algorithm is shown in Table **1**.

When the DE verifies the *VO*, it first reconstructs the root digest h'_{root}. Based on the h'_{root} and *Root*, the DE can establish if *Set(QI)* is correct and complete. For the result in *Set(QS)*, first the DE checks if all the *GUID* in *Set(QI)* appears in *Set(QS)*, then computes the digest of the record in *Set(QS)* and authenticates with the signatures of all the records that satisfy the *Q* in *S*, the verification algorithm is shown in Table **2**.

4.2. The Correctness and Completeness of MTDA

Based on the signature involved in VO, the data engine can verify the correct and provenance integrity of *ANS(Q),* but it can't discover if the malicious insider deletes the tenant record in shared table for the records of *ANS(Q)* may be scattered in the share table based on the query attribute.

As the query process of *QS* can be treated as equal-join query between index table and shared table with join condition *Indextable.GUID= Sharedtable.GUID* in their respective attribute. Because the *GUID* attribute is the globally unique identifier for record level rapid positioning, it is a one-to-one correspondence between Index table and Shared table. Based on the query process, we get the following conclusion.

Table 2. The verification algorithm.

```
Algorithm 1 CAS verification(VerificationObject VO, Set(QS))
Begin
h'root= Reconstruct(VO)
verify h'root with VO.Root or reject
For every t in VO
If t.GUID contained in Set(QS) or reject
For every r in Set(QS) verify r.digest with sn or reject
//
Reconstruct (VerificationObject VO)
Begin
  S=Null
  While VO≠ Null
    Remove next entry e from VO
    If e is a hash value Append e to S
    If e is a point of record r Append h(r) to S
    If e is (, Append Reconstruct h'root(VO) to S
    If e is ), return hash(S)
End
```

Theorem 1 If the *Set(QI)* is correct and complete, any deletion on the shared table of *Set(QS)* can be checked by compare *Set(QI)* with *{GUID| Set(QS)}*, if *Set(QI)={GUID| Set(QS)}*, we can say that *Set(QS)* is complete.

Proof: Consider the contrary, suppose the Set(QS) is complete, but Set(QI) ≠ {GUID| Set(QS)}. Consider the two situations: a, Set(QI)< {GUID| Set(QS)} ; b, Set(QI)> {GUID| Set(QS)}.

Against a, it means $\exists$ r, r$\in$Set(QS) and r.guid$\notin$Set(QI), r is omitted from Set(QI) which means Set(QI) is incomplete, it conflicts with the precondition that Set(QI) is correct and complete.

Against b, it means $\exists$ *r, r*$\notin$*Set(QS)* and *r.guid*$\in$*Set(QI),* because precondition that *Set(QI)* is correct and complete, r's absent from *Set(QS)* is conflict with the assumption that the *Set(QS)* is complete.

So give the precondition that *Set(QI)* is correct and complete, we can check the completeness of *ANS(Q)* by comparing *Set(QI)* with *{GUID| Set(QS)}*.

The correctness of the index data is guaranteed by PAT due to the security of collision-resistance hash functions and the public key digital signature for the hash value of the root node. Completeness can be assured by the sorted binary sets and the boundary binary set that enclose the select range. Based on the PAT, we can ensure the correctness and completeness of *Set(QI)*. And according to Conclusion 1 and PAT, we can check the completeness of *ANS(Q)* by compare *Set(QI)* with *{GUID| Set(QS)}*.

4.3. Data Update

In MTDA the data update contains three kinds of circumstance: data insert, delete and modify. For data deleted, we can mark on the tree and set to delete information and don't adjust the structure; for data insert, we insert in the leaf node with new hash and signature and then bottom-up change the path node until the root. For data modification, there are two types of modification: one is only the universal table that does not involve the pivot table and the other involves both. For the former category, we only need to update the signature of the corresponding to the modified record while the later need to amend the *PTA tree* and *S-set* at one time and insert belongs to the later one.

4.4. Cost Analysis of MTDA

Suppose there are *N* records in Tenant logical view R and the *ANS(Q)* has N_Q records, $|k|$, $|p|$, $|h|$ and $|sn|$ denote the size of the searching key, pointer, hash value and signature.

Storage cost the storage cost of PAT is:

$$C_s^{MTDA} = (\frac{Nf - 1}{f - 1})(|k| + |p| + |h|) + 2 \cdot |sn| \quad (1)$$

VO Construction Suppose δ_i is the total number of query results contained in the left boundary leaf node of the query sub tree, μ_i on the right boundary of the sub tree, the size of the VO is:

$$\begin{aligned} C_s^{VO} &= (\lfloor logN \rfloor - \lfloor logN_Q \rfloor) \cdot (f - 1) |h| \\ &+ \sum_{i=0}^{\lfloor logN_Q \rfloor - 1} (2f - d_i - m_i) |h| + 2|sn| \\ &+ (f - d_{\lfloor logN_Q \rfloor} - m_{\lfloor logN_Q \rfloor}) + 2(N_Q + 2)|h| \end{aligned} \quad (2)$$

Verification cost Given the *VO*, *DE* has to compute the missing digests and combine them with the *VO* to retrieve the root of the *PTA tree*. This procedure involves calculating $2 \cdot |N_Q| - 1$ hashes on top of *ANS(Q)*, and combining them with the *VO* digests, incurring a maximum of $2 \cdot (d_{PTA\ tree} - 1)$ additional digest computations. Finally, *DE* has to verify whether the computed *PTA tree* root matches the one returned in the VO, using the tenant's public key. Letting the verification cost of signature be C_v and hash cost be C_h. The total verification cost is:

$$\begin{aligned} C_c^{VO} &= \left(\sum_{i=1}^{\lfloor logN_Q \rfloor} f^i + \lfloor logN \rfloor - \lfloor logN_Q \rfloor \right) \cdot c_h \\ &+ N_Q \cdot c_h + 2 \cdot c_{sn} \end{aligned} \quad (3)$$

5. EXPERIMENT

We perform an experiment to demonstrate our analysis of the multi-tenant data authentication scheme approach. Based on the meta-data driven multi-tenancy storage model, we construct the shared database storage model with MySQL 5.6.14 and the development environment is Eclipse-SDK-4.3.1-win 64 Bit, operating system is Windows XP Professional Service Pack 3, CPU is Inter Core (TM)2 2,33GHz, and the memory is 2Gb. We utilize RSA signatures that are typically 128 bytes in size and SHA1with 20-byte outputs.

5.1. Query Performances

In this experiment, we test the query performances influence to T_{01} of the index authentication scheme to multitenant sharing database. We compare the pivot based index models

with the case that builds *MB trees* on pivot table with searching key A_1. We set up a Tenants T_{01} with a data set cardinality with *300k* records with size of 1024 bits and range queries with selectivity σ varying between 10% and 50%. The result shows that the query response of independent authentication model is about three times faster than index authentication scheme, shown in Fig. (**3**). So for the diversity of tenant requirement, it is inappropriate to apply generic index authentication scheme for tenants.

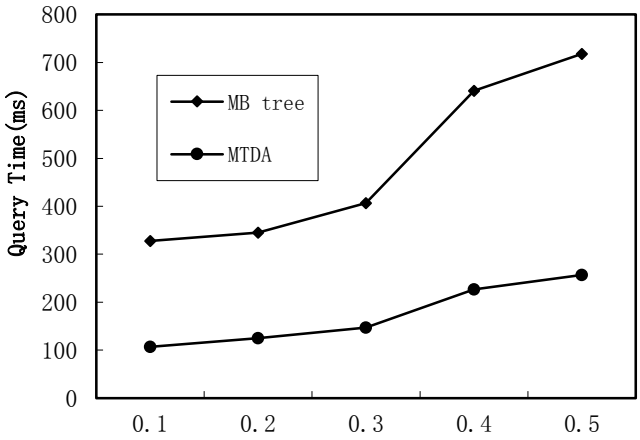

Fig. (3). Query performances.

5.2. Construct Cost

In this experiment, we set up a Tenants T_{01} with data set with 100k record and they customize the same application while T_{01} specify A_1 A_2 as the searching key. We stores T_{01}-A_1, T_{01}-A_2 into pivot table. We compare our models MTDA with the case *T* that builds MB tree on universal table with searching key and pivot table. We specify the fanout of MB tree and PAT as f=10 in our experiment. The set up cost of them is shown in Fig. (**4**). Since in MB tree, in order to support simple range queries on multiple single attribute, hash trees for all possible attributes orders of relation must be precomputed [16]. The construction cost of MB tree solution is growing with the specified index attributes numbers, so is pivot table authentication. But the construction cost of the universal table of S set is constant because S set is shared by multiple searching attributes.

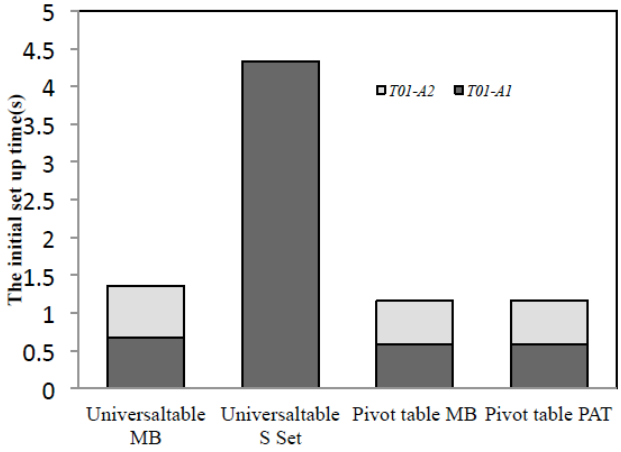

Fig. (4). The initial set up cost of MTDA and MB on Universal table and Pivot table.

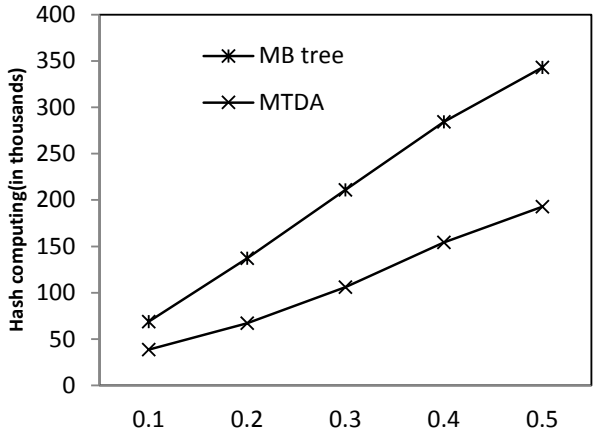

Fig. (5). Total hash operations of VO verification.

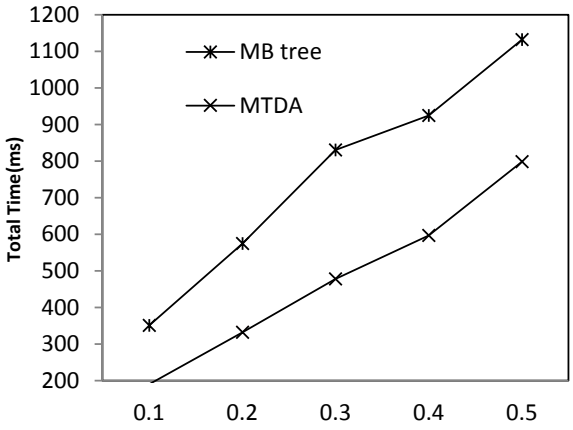

Fig. (6). VO verification Time.

5.3. VO Verification

In this experiment, we create workloads of random 100 range queries with selectivity σ varying between 10% and 50% on data cardinality *300K* on $T_{01\text{-}A1}$. *MTDA* show a better performance on VO verification, because in the case *T*, they have to set up separate MB trees on universal table and pivot table which leads to double reconstruction of MB tree, while in *MTDA*, they only need to travel the tree once and combine with one aggregated signature to verify tenants data. Fig. (**5**) shows the total hash operations and Fig. (**6**) gives the total *VO* verification times. It shows that *MTDA* have an advantage on the query conditions changing to different searching attribute, compared with MB tree approaches.

6. CONCLUSION AND FUTURE WORK

In SaaS, applications and databases are both hosted at the service providers, tenant query result authentication become the biggest challenge caused by the untrustworthiness of service providers. In this paper, we put forward a multi-tenant data authentication model *MTDA*. *MTDA* can accommodate the multi-tenant properties perfectly by establishing isolated authentication structures for each tenant based on their integrity demands. There remains some problem of MTDA for the future work such as the tenant dynamic data

operation and multiple attribution query authentication. To combine data integrity with data privacy in SaaS is a challenging problem which remains to be solved.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENT

This work is supported by National Key Technologies R&D Program No.2012BAH54F01; National Natural Science Foundation of China under Grant No.61272241, No.61303085; Natural Science Foundation of Shandong Province of China under Grant No.ZR2013 FQ014; Science and Technology Development Plan Project of Shandong Province No. 2012GGX10134; Independent Innovation Foundation of Shandong University under Grant No. 2012TS075, No.2012TS074; Shandong Province Independent Innovation Major Special Project No. 2013CXC30201.

REFERENCES

[1] Wikipedia. Software as a service. http://en.wikipedia.org/wiki/Software_as_a_service.

[2] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, J. Rittinger, "Multi-Tenant Databases for Software as a Service," Schema-Mapping Techniques. SIGMOD, 2008.

[3] J. L. Beckmann, A. Halverson, R. Krishnamurthy, J. F. Naughton, "Extending RDBMSs To Support Sparse Datasets Using An Interpreted Attribute Storage Format," ICDE, 2006.

[4] E. Chu, J. Beckmann, J. Naughton, "The MDAe for a Wide-Table Approach to Manage Sparse Relational Data Sets," *SIGMOD*, 2007.

[5] C. D Weissman, S. Bobrowski, "The Design of the Force.com Multitenant Internet Application Development Platform," *SIGMOD*, 2009.

[6] L. Kong, Q. Li, Y.Shi, X. Wang, "A research on SaaS-oriented multitenant indexing model based on keyvalue pairs," *Chinese J. Comput.*, 2010.

[7] C. Pang, Q. Li, L. Kong, "An Index Model for Multitenant Data Storage in SaaS", *WAIM 2013, LNCS 7923*, vol. 7923, pp. 423-428. 2013.

[8] S. Papadopoulos, Y. Yang, D. Papadias, "Continuous authentication on relational streams", *VLDB J. (VLDB)*, vol. 19, no. 2, pp. 161-180, 2010.

[9] F. Li, M. Hadjieleftheriou, G. Kollios, L. Reyzin, "Authenticated Index Structures for Aggregation Queries," *ACM Trans. Inf. Syst. Secur. (TISSEC)*, vol. 13, no. 4, pp. 32, 2010.

[10] P. T. Devanbu, M. Gertz, Charles U. Martel, S. G. Stubblebine, "Authentic Third-party Data Publication", *DBSec*, pp. 101-112, 2000.

[11] F. Li, M. Hadjieleftheriou, G. Kollios, L. Reyzin, "Dynamic authenticated index structures for outsourced databases", *SIGMOD*, pp.121-132, 2006.

[12] K. Mouratidis, D. Sacharidis, and H. Pang, "Partially materialized digest scheme: An efficient verification method for outsourced databases," *Int. J. Very Large Data Bases*, vol. 18, no. 1, pp. 363-381, 2009.

[13] W. Wei, T. Yu, R. Xue, "iBigTable: practical data integrity for bigtable in public cloud", *CODASPY*, pp. 341-352, 2013.

[14] H. H. Pang, J. Zhang, K. Mouratidis, "Scalable verification for outsourced dynamic databases," *PVLDB*, vol. 2, no. 1, pp. 802-813, 2009.

[15] E. Mykletun, M. Narasimha, G. Tsudik, "Authentication and integrity in outsourced databases," *TOS,* vol. 2, no. 2, pp.107-138, 2006.

[16] M. Narasimha, G. Tsudik, "Authentication of outsourced databases using signature aggregation and chaining," *DASFAA*, pp. 420-436, 2006.

[17] H. H. Pang, K. L. Tan, "Authenticating query results in edge computing", *ICDE*, pp. 560-571, 2004.

[18] H. Pang, A. Jain, K. Ramamritham, and K. L. Tan, "Verifying completeness of relational query results in data publishing," *ACM SIGMOD*, pp. 407-418, 2005.

[19] S. Papadopoulos, A. Kiayias, D. Papadias, "Exact in-network aggregation with integrity and confidentiality," *IEEE Trans. Knowl. Data Eng. (TKDE)*, vol. 24, no. 10, pp. 1760-1773, 2012,

[20] M. Xie, H. Wang, J. Yin, and X. Meng,"Integrity Auditing of Outsourced Data", In: *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007)*, pp.782-793, 2007.

[21] H. Wang, J. Yin, C. Perng, and P. Yu, "Dual encryption for query integrity assurance", In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM 2008)*, pp. 863-872, 2008.

[22] Y. Shi, K. Zhang, Q. Li, "Meta-data driven data chunk based secure data storage for SaaS", *JDCTA: Int. J. Digit. Cont. Tech. and its Appl.*, vol. 5, no. 1, pp. 173-185, 2011.

[23] J. G. Apostolopoulos, "*Authentication of modified data*", U. S. Patent Application 20080115194. May 15, 2008.

[24] J. D. Koziol, A. R. Koziol, "*Data authenticator*", U. S. Patent Application 20100031048. Feb 04, 2010.

[25] M. Boesgaard, "*Authenticating a web page with embedded javascript*", U. S. Patent Application 20120222127, Aug 30, 2012.

[26] C. Weissman, D. Moellenhoff, S. Wong, P. Nakada "*Multi-Tenant Database System*", U. S. Patent 8280874, Oct 2, 2012.

[27] M. Sakai, "*SAAS management system, method for saas management, and SAAS management program*", J. P. Patent, 2013168044. Aug 29, 2013.

[28] E. Ilyayev, "*Cloud plarform for managing software as a service (SAAS) resources*" U. S. Patent Application 20110126168. May 26, 2011.

[29] C. W. Eidler, W. T. Fuller, M. Hanly, S. H. Berman, R. Joyner, B. Van Hooser, P. T. Conroy, "*Secured shared storage architecture*" U. S. Patent 7693970. Apr 06, 2010.

[30] A. Brown, J. S. Chase, "*Trusted platform-as-a-service: a foundation for trustworthy cloud-hosted applications*", *CCSW*, pp. 15-20, 2011.

[31] J. E. King, R. J. Jones, "*Enhancing trusted platform module performance*", U. S. Patent Application 20060005000. Jul 05, 2006.

[32] J. E. King, R. J. Jones, "*Trusted platform modules*," U. S. Patent 8429423. Apr 23, 2013.

[33] M. Scott-nash, A. Munoz, A. Altman, "*Virtual trusted platform module*", U. S. Patent Application 20140007087. Jul 02, 2014

Received: September 22, 2014 Revised: November 30, 2014 Accepted: December 02, 2014