Skip List Based Authenticated Data Structure in DAS Paradigm

Jieping Wang[1,2], Xiaoyong Du[1,2]
1. Key Laboratory of Data Engineering and Knowledge Engineering, Beijing 100872, China, MOE
2. School of Information, Renmin University of China
{wangjieping,duyong}@ruc.edu.cn

***Abstract*—In Database-as-a-Service (DAS) computing paradigm, data owners delegate database management tasks to service provider. Since the service provider is untrusted, query authentication becomes an essential issue before database outsourcing. In previous research, various Authentication Data Structures (ADS) have been proposed. However, most of them are either disk-based or static. Considering service provider usually owns large memory and powerful processors, in this paper we propose a dynamic main memory ADS based on skip list, which could provide efficient authentication for range query both in static and in dynamic scenarios. Experimental results confirm our claim through a systematic analysis of verification metrics.**

Keywords-DAS; query authentication; skip list; authenticated data structure

I. INTRODUCTION

DAS (a.k.a. Outsourced database, ODB) [1] is a promising database management paradigm, where data owners delegate their data to the third-party: service provider. Due to economy of scale, DAS could provide high quality of service at reduced cost. Since the service provider is untrusted or compromised, new security issues have been introduced, among which authentication of query results is a major concern before outsourcing database. Basically, query authentication focuses the following three dimensions of data: soundness, completeness and freshness [2]. Soundness means the result set does originate from data owner and has not been tampered with in any way. Completeness means no qualified tuples missing from result set. Freshness means the result set incorporates the latest version. To achieve query authentication, we introduce a three-tier architecture in Figure 1.

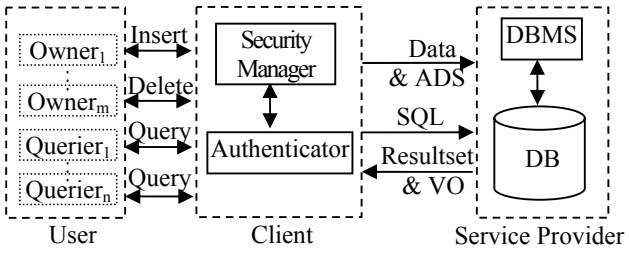


Figure 1. System overview

In Figure 1, there are three entities involved: trusted user, trusted client and untrusted service provider. User can be divided into two roles: owner, who is responsible for data generation and maintenance; querier, who accesses data. Client is responsible for security related tasks, such as data encryption and query authentication. The rest of database management task is carried out by service provider.

Basically, we partition database management tasks into three phases: initial phase, query phase and maintenance phase. In initial phase, owners generate data and upload them to client. Client creates ADS upon data and delegates them to service provider. In query phase, service provider executes query locally and returns result set and associated VO (*Verification Object*) to client, which does the final verification. In maintenance phase, client recalculates ADS and transfers it to service provider. Consequently, authentication cost includes three parts: .construction cost of ADS, query authentication cost and ADS maintenance cost.

In real applications, above three entities own different computer powers: user is usually power limited, e.g. mobile device; client typically possesses a work station, or even a PC computer, while service provider runs a server cluster. The goal of DAS is to push workload to service provider as much as possible and keep client overhead and communication overhead between client and service provider minimized.

In previous research, various ADSs have been proposed. However, most of them focus on disk-based structures such as VB-tree [3], MB-tree and EMB [2], where I/O cost and cryptographic operation cost are two dominating factors. Obviously, none of them considers the fact that service provider has large memory and powerful processors. The only main memory ADS proposed is based on MHT, a binary balanced tree. Since update operations will cause complete rebuilding, MHT is applicable only for static scenario. In this paper, we propose a dynamic main memory **ADS** based on **S**kip **L**ist: ADSSL, which has one dominating factor: cryptographic operation. As a probabilistic alternative to balanced tree, Skip list is easy to implement and efficient in data updates. Specifically, ADSSL has the following advantages:

- ADSSL is space efficient and suitable for main memory structure.

This work is supported by two MOE projects under Grant No 708004 and 200800020001.

978-0-7695-3766-5/09 $25.00 © 2009 IEEE
DOI 10.1109/GCC.2009.33

- ADSSL is probability balanced and can supply flexible trade-off between storage and verification cost.
- ADSSL supports efficient authentication of range query.
- Maintenance cost of ADSSL is similar to that of query authentication cost.

Besides query authentication, there's another important class of security issue: data security, which mainly focuses on how to prevent sensitive data from unauthorized access. It's orthogonal to our issue and not covered in this paper.

The rest of the paper is organized as follows. We presented related work in Section II. In section III, we proposed our new ADS based on skip list and gave systematic evaluations on various authentication costs. Besides, a VO size optimized version of ADSSL is proposed. Experiments were made in section IV. Section V concludes the paper and points out future work.

II. Related Work

Existing research on query authentication can be divided into three categories: signature based, ADS based and probability based. In the following, we compare them from three aspects: authentication ability (the type of query supported), authentication cost and authentication ratio.

Although assigning one signature to each tuple could provide soundness authentication at tuple level, it has serious efficiency problem both in space and time cost. Furthermore, it could not provide completeness authentication. To decrease verification time and network workload, aggregate signature [6, 7, 9] was proposed, which transforms multiple signature verifications into multiple modular multiplications and one verification. Signature chain [7, 8] was proposed to support completeness authentication, where one signature is based on three consecutive tuples instead of one tuple. In a word, signature based approach is a simple approach with 100% authentication ratio. However, it has high authentication cost and limited authentication ability. Specifically, it can only support soundness and completeness authentication of range query.

To decrease signature cost at tuple level, authentication data structure (ADS) [2, 3, 10, 12] has been extensively studied which organises data into a tree so that only one signature for root is needed. Similar to the signature approach, ADS could achieve 100% authentication ratio. Among them Merkle Hash Tree (MHT) is a commonly used one. Basic structure of MHT is illustrated in Figure 2(a). Through rebuilding of MHT and verification on root node, any tampering will be detected. Due to the fast and space compact of hash, MHT can decrease authentication cost dramatically compared with signature approach. To enhance performance further, several variants of MHT [2, 3] are proposed, including VB-tree, MB-tree and EMB. The last two structures are illustrated in Figure 2(b) and 2(c) respectively. In a word, ADS can supply soundness and completeness authentication for one-dimensional range query efficiently. For multi-dimensional range query, multiple MHT must be built. Besides, it's only suitable for non-sensitive attribute and its maintenance cost is large.

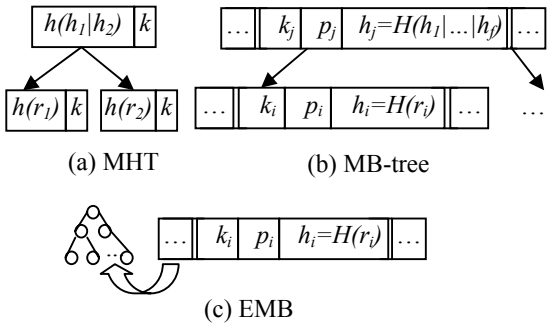


Figure 2. Three ADS based on MHT

Different from above two approaches, probability based approach can only provide authentication in part. On the other hand, it has strong authentication ability with less authentication cost. There have been two probability authentication approaches: fake tuple [13] and challenge token [11]. The basic idea of fake tuple is as follows: real tuples, coupled with certain percentage of fake tuples, are outsourced. Through authentication on fake tuples, authentication on real tuples is inferred with certain probability. However, fake tuple approach brings additional querying and storage cost at DSP. Furthermore, it can not support authentication of aggregate query. The basic idea of challenge token is that for a batch of queries, through authentication on sampled queries, authentication on all queries can be inferred. In [11], fake tokens are proposed to increase authentication ratio without increasing authentication cost. Challenge token can authenticate any type of query efficiently. However, it has two limitations. First, it's a server authentication approach, which means after providing correct execution proofs, DSP can still modify result set. Second, it has a strong assumption that the batch of queries is carried out on the same segment of data, which is difficult to ensure in real application. Compared with above two approaches, probability based approach can authenticate soundness, completeness and freshness of data.

In a conclusion, signature approaches and ADSs can provide 100% authentication ratio, but they have limited authentication ability and relative large authentication cost. The probability based approach is much more efficient and effective, but its authentication ratio is less than 100%. To decrease authentication cost of current ADS, we propose a main memory ADS, which only cryptographic operation costs are involved.

III. Authentication Data Structure Based on Skip List

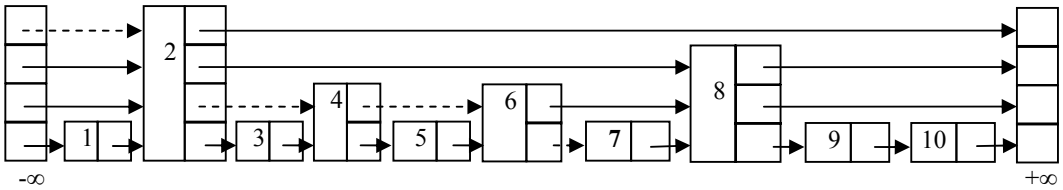
In this section, we first introduce our main memory ADS based on skip list, and then give a detailed analysis on various authentication costs. Finally, we give a VO size efficient version.

A. Structure

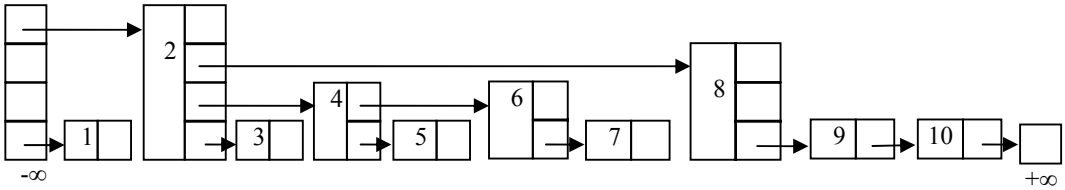
Skip list was first proposed by William [4] in 1990 as a probabilistic alternative to balanced tree, e.g. B+-tree. Figure 3(a) is an example, where pointers with same level are linked by sorted order. The nodes at low level jump higher with uniform probability *p*. For brevity, a node that has k forward pointers is called a *level k node*. According to nodes jumping higher randomly or not, there're two kinds of skip lists: randomized skip lists, e.g. Figure 3(a); deterministic skip lists, e.g. each *1/p* node jumps one higher. Among them, randomized skip lists are suitable for dynamic scenario and adopted in this paper. Compared with balanced trees, skip lists have the following characteristics:

- Skip lists are easy to implement and practically efficient in search, especially update time (without node split or merge). Interested readers can refer to [4] for concrete comparisons.
- Skip lists are space compact, where space is allocated when needed, while empty space (decided by filling factor) is preserved in balanced tree.
- Skip lists are main memory index, while balanced tree are disk-based index.

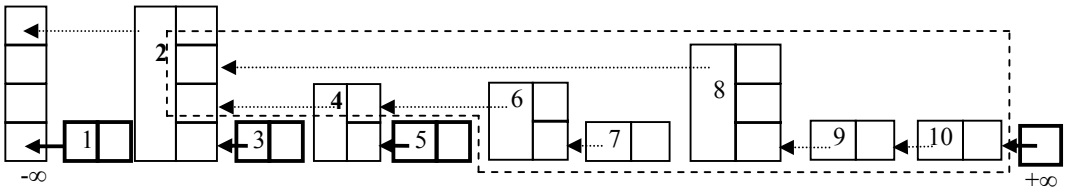
In balanced tree, there's only one search direction: move down from root node to leaf node. The number of I/O is decided by the height of tree. However, in skip lists, search includes two directions: *hop forward* and *drop down*. For example, search path of value 7 is plotted by dotted arrows in Figure 3(a). It's unsuitable to partition skip lists either horizontally or vertically. In conclusion, skip lists are dynamic and efficient main memory index, especially suitable for the context of service provider.



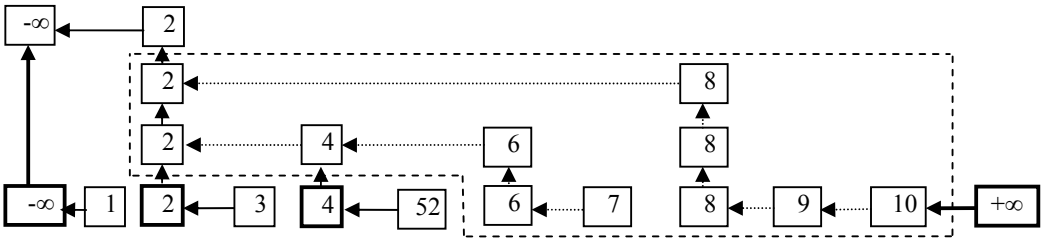
(a) Search path of value 7 in skip list by dotted arrows



(b) Authentication data structure based on skip list



(c) Verification process of result set {6,7,8,9,10}



(d) VO size optimized authentication structure based on skip list

Figure 3. ADSSL and BADSSL based on skip list

Although Goodrich etc. proposed authentication dictionary based on skip list and commutative hashing [5], it could only support membership query, such as "is element s in set S". For range query, a new authentication structure is required. Inspired by MHT, we transform skip lists into our new authentication structure, which is illustrated in Figure 3(b). Unlike linked list in Figure 3(a), Figure 3(b) can be regarded as a tree with root node -∞, where at most one path exists between any two nodes.

The search path in Figure 3(b) is similar to that in Figure 3(a), except that the number of comparisons is reduced. The construction algorithm of ADSSL is presented in Figure 4.

Algorithm: Construction of ADS Based on Skip Lists
Input: data set: ds, probability: p; maximal level： l_{max}
Output: ADS Based on Skip List

```
1.  for each data in ds{
2.      level = random(l_max);
3.      createNode(data, level);
4.  }
5.  createNode(–∞, l_max);
6.  createNode(+∞, 1);
7.  sort nodes on search key;
8.  for each pointer of each node {
9.      if exists(consecutive node with equal or less level)
10.     link them;
11. }
12. scan nodes by reversed sorted order{
13.     calculate verification value for each node;
14. }
```

Figure 4. Construction algorithm of ADS based on skip list

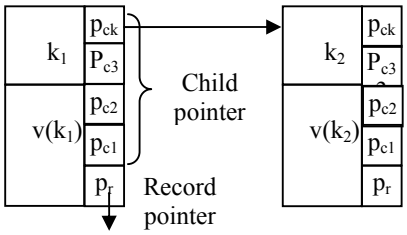


Figure 5. Node structure

To support authentication of range queries, each node is assigned with a verification value and the node structure is illustrated in Figure 5, where a node is composed of three components: key (e.g. k_1, k_2), verification value (e.g. $v(k_1)$, $v(k_2)$) and pointers (e.g. p_r, p_{c1}). The calculation of verification value ensures that it includes information of itself and all of its children. Here we divide pointers into two classes: child pointers and record pointers. Record pointers, denoted as p_r, refer to the pointer to record. Child pointers, denoted as p_c, refer to remaining pointers. To calculate verification value uniquely, child pointers are sorted by their levels. As for node $n(k, v, p_r, p_{c1},\ldots p_{ck})$, according to whether p_c is null or not, the calculation of v is as follows:

Case 1: p_c=null

$v(k) = h(r)$

Case 2: $p_c \neq$null

$v(k) = h(h(r)||v(k->p_{c1})||v(k->p_{c2})||\ldots||\ v(k->p_{ck}))$

For example, in Figure 3(b), node (7) and node (2) correspond to above two cases respectively. Their verification values are calculated as follows:

$v(7) = h(r_7)$

$v(2) = h(h(r_2)||v(3)\ ||v(4)\ ||v(8))$

After initial construction of authentication structure, we calculate verification value for each node from right to left. The calculation of verification value ensures that each node contributes one and only one time to the root node, which is consistent with MHT.

B. Query Verification

In this subsection, we discuss how to achieve authentication for range query based on node verification values calculated above. Generally, one range query can be transformed into two (boundary) point queries. For example, assume ADSSL in Figure 3(c) is constructed on attribute X. To authenticate range query "7<=x<=9", two point queries "x=7" and "x=9" are issued respectively. Query process starts from the root node (here is -∞) and recursively hops forward and drops down until the node with search key is reached. Here the final result set is {7, 8, 9}. To provide boundary completeness, two additional nodes are included: {6, 10}. So the result set is {6, 7, 8, 9, 10}. The following problem is how to construct VOs to prove that the result set is sound and complete.

Generally, VO generation is a reverse process of search, which constructs path from the result set to root. Any node which is required in construction but can not be inferred by the result set (co-path) is included in VO. For example, in Figure 3(c), to calculate verification value of node (4), two VOs are included: $h(r_4)$ and v(5). Two point queries form a subtree, which is illustrated by dotted line polygon in Figure 3(c). The VO is composed of those nodes intersecting with subtree, which are illustrated by bold arrows in Figure 3(c). VO is illustrated by bold box. The formal algorithm of verifiable result set generation is illustrated in Figure 6.

Consider, for instance, above query "7<=x<=9", the verifiable result set generated is:

[h(-∞),v(1),[h(r2),v(3),[h(r4),v5],[r6,r7]],[r8,[r9,[r10,v(+∞)]]]]. The time complexity of VO generation is similar to that of query process. After receiving the result set and root signature from service provider, client would verify if it is sound. Specifically, it scans result set to compute root hash and checks if it agrees with unsigned root signature. Completeness verification can be achieved by consecutive validation of values in the final result set. There must not be any VO among values in the result set. For example, if one value, such as r_8, is omitted from the result set, then the result set is not consecutive. For example, $\{r_6, r_7, VO, r_9, r_{10}\}$. Authentication of ADSSL is ensured by the following theorem.

Theorem 1. ADSSL could provide soundness and completeness verification of query results.

Proof: Basically, authentication ability of ADSSL is based on the following three factors: first, hash function used to calculate node verification value is one-way and collision-free; second, each node contributes one and only one time to the computation of root hash; third, the order in which each of value is involved in root hash calculation is fixed and unique. If a attacker modifies, insert or deletes any value in the result set, the possibility of re-computing root hash correctly is equal to

the possibility of collision in hash, which is negligible in practice.

Algorithm: Verifiable result set generation
Input: skip list root, rang query[low, high];
Output: a verifiable result set

```
1.  Init stack s; //note: s is a set
2.  search low and push search path in stack;
3.  search high and push search path in stack;
4.  Init rs;
5.  while (!empty(s)) {
6.      n = pop(s);
7.      push n's pointer values in rs in order;
8.  }
9.  return rs;
```

Figure 6. Verifiable result set generation algorithm

C. Maintenance of ADSSL

As we pointed out previously, skip lists are efficient especially in dynamic scenario. The maintenance of ADSSL is similar to that of skip lists, which is composed of two steps: query and insert (delete). Here update is replaced by a pair of insert and delete. So maintenance cost is the sum of query cost and insert cost. Since ADSSL is a main memory index, query cost is negligible compared with that of insert. Insert cost is dominated by recalculation of node verification value from inserted (deleted) node to root, which is linear to the search path. Since ADSSL can be maintained efficiently, it can authenticate data freshness efficiently.

D. Cost Analysis

Since authentication cost is involved in three phases, we will discuss them respectively. In each phase, authentication cost includes three parts: space cost, time cost and network cost. Costs of MHT are presented for comparison.

TABLE I. NOTATIONS AND DESCRIPTION

Symbol	Description	Default Value
C_h	Time of hash	20 μs
C_s	Time of signature	9.53 ms
C_v	Time of verification	0.14 ms
|s|	Size of signature	128 bytes
|h|	Size of hash	16 bytes
|k|	Size of search key	4 bytes
|pointer|	Size of pointer	4 bytes
|nMHT|	Size of MHT node	28 bytes
| nSL|	Size of skip list node	28 bytes
N	Cardinality of table	2^{20}
P	Probability of skip list	
lmax	Max level of skip list	Log1/pN
Cx	Construction time of x	
Sx	Space cost of x	
Vx	Verification time of x	
VOx	VO size of x	
Mx	Maintenance cost of x	

In initial phase, ADS is constructed and delivered to the service provider. Since the root node can be signed only by the client, the client must construct the whole authentication tree locally. Fortunately, the client needn't shift authentication tree to the service provider. Instead, it can be reconstructed at the service provider. Thus a large amount of network cost is saved and the only additional network cost is one signature. So in initial phase, cost focuses on two aspects: construction time and space time. Specifically, construction time is dominated by hash computation. Symbols used in this paper are summarized in table I. Formally, initial costs of MHT and ADSSL are calculated as follows:

$$\mathbf{C_{MHT} = (2N-1)*\ C_h + C_s}$$
$$\mathbf{C_{ADSSL} = N*\ C_h + C_s}$$
$$\mathbf{S_{MHT} = (2N-1)*|n_{MHT}|+|sign|}$$
$$\mathbf{S_{ADSSL} = N*(|n_{SL}|+4p/(1-p))+|sign|}$$

Obviously, the number of hash in ADSSL construction is about half of MHT. Although the space cost of ADSSL varies with probability p, it's always much less than that of MHT. The calculation of SADSSL is as follows:

$$\sum_{i=0}^{l_{\max}-1} N*(p^i - p^{i+1})*(|n_{SL}|+|pointer|*i)$$

$$=N*(|n_{SL}|*\left(1-p^{l_{\max}}\right)+\frac{4*p*(1-p^{l_{\max}-1})}{1-p}-4*(l_{\max}-1)*p^{l_{\max}})$$

$$= N*(|n_{SL}|+\frac{4*p}{1-p}) \quad (p^{l_{\max}} \approx 0)$$

In query phase, authentication cost is dominated by the number of hash to recompute root hash. Specifically, verification cost and VO size are calculated as follows:

$$\boldsymbol{V_{MHT} = logN*}\ C_h + C_v$$
$$\boldsymbol{V_{ADSSL}} = \frac{\log_{1/p} N*(p+1)}{2p} C_h + C_v$$
$$\boldsymbol{VO_{MHT} = logN*|h|+|sign|}$$
$$\boldsymbol{VO_{ADSSL}} = \frac{\log_{1/p} N*(\log_{1/p} N-1)*(p+1)}{4p}|h|+|sign|$$

Assume result set has one value. Extension to result set with more values is straightforward. As for MHT, the height of tree is logN. So do the number of hash and VO. The analysis of ADSSL is much more complex. For jumping probability p, the height of ADSSL is $\log_{1/p}N$. On average, for 1/p nodes with the same level, only one of them jumps higher. Besides, the probability of any node jumping higher is the same. So the average search path at 1 level is $\sum_{i=1}^{1/p} i*p = \frac{1+p}{2p}$. The number of VO with k level node is (k-1). So the total number of VO is $\frac{1+p}{2p}\sum_{i=0}^{\log_{1/p}N-1} i$.

Compared with MHT, ADSSL has higher verification cost, especially in VO size. The detailed comparison is plotted in Figure 8. Since VO will influence network performance negatively, we propose an improved version, denoted as BADSSL, which is presented in Figure 3(d). Obviously, each node with k child pointers in ADSSL is divided into k nodes with one child pointer. Consequently, The number of VO with k level node decreases from (k-1) to 1. The initial and verification of BADSSL is as follows:

$$C_{BADSSL} = \frac{N}{1-p} C_h + C_s$$

$$S_{BADSSL} = \frac{N}{1-p} |n_{SL}| + |\text{sign}|$$

$$V_{BADSSL} = \frac{\log_{1/p} N*(p+1)}{2p} C_h + C_v$$

$$VO_{BADSSL} = \frac{\log_{1/p} N*(p+1)}{2p} |h| + |sign|$$

In maintenance phase, MHT is static and maintenance cost is almost equal to rebuilding cost. While maintenance cost of ADSSL and BADSSL is dominated by search path. Formal analysis is as follows:

$$M_{MHT} = (2N-1)*h() + sign()$$

$$M_{ADSSL} = (\frac{\log_{1/p} N*(p+1)}{2p} + 1) C_h + C_v$$

$$M_{BADSSL} = (\frac{\log_{1/p} N*(p+1)}{2p} + 1) C_h + C_v$$

Here we assume before insertion, the level of current ADSSL is equal with max level. If the level of inserted element is larger than current max level, maintenance cost will increase a little.

IV. EXPERIMENTAL EVALUATION

A. Experimental Setup

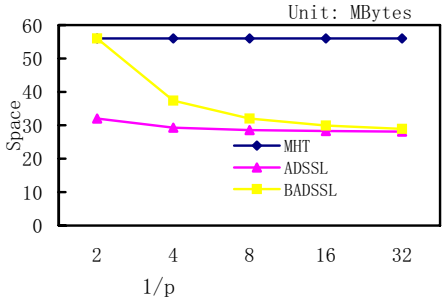
We carried out experiments on Microsoft Windows XP with a 1.7GHz Intel processor and 512M RAM. Cryptographic signature and hash were performed using the standard Java implementation of RSA and MD5 algorithm. Our ADSSL and BADSSL are implemented in JAVA/JRE 1.6 development kit and Runtime Library.

B. Results Analysis

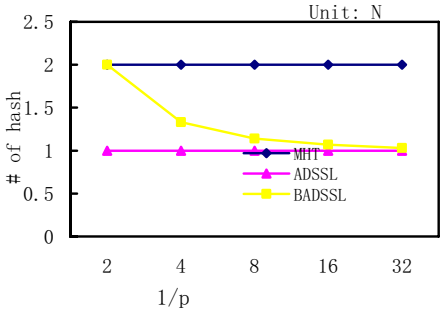
We compare initial construction cost, query authentication cost and maintenance cost of three ADSs: MHT, ADSSL and BADSSL.

Three sets of experiments were performed in total. In the first set of experiments, we study the space and time cost of initial construction. The results are illustrated in Figure 7. Both space cost and time cost of our approaches are less than those of MHT. Besides, MHT is independent of probability p, while our two approaches negatively depend on *1/p*. Although space and time cost of ADSSL are always less than those of BADSSL, with the increasing of *1/p*, the difference decreases quickly. Specifically, when *1/p* reaches 32, the costs of them are similar.

In the second set of experiments, we study query authentication cost and VO size. The results are illustrated in Figure 8(a) and 8(b). In Figure 8(a), verification cost of our approaches is always larger than that of MHT. However, for an appropriate *p*, such as 1/4, the difference between them is very little. So the selection of *p* is critical in deciding verification cost. In Figure 8(b), VO size of ADSSL is much larger than two others, which confirms BADSSL is VO size efficient.

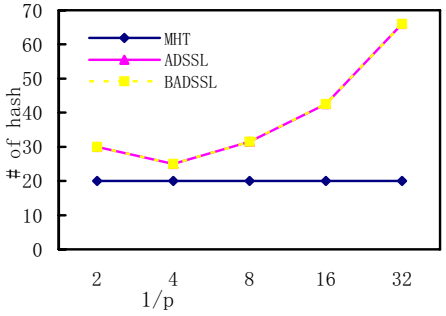


(a) Space cost

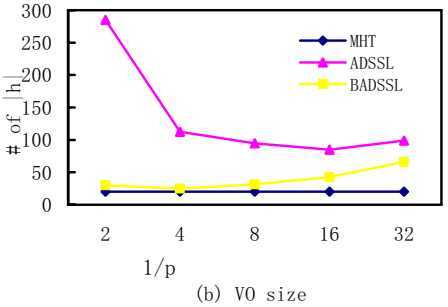


(b) Time cost

Figure 7. Initial construction cost of three ADSs



(a) Verification cost



(b) VO size

Figure 8. Query authentication cost and VO size comparison

The third set of experiments is to compare maintenance cost of three approaches. The results are illustrated in Figure 9. Since maintenance cost of MHT is almost equal to that of rebuilding, it's much larger than the others and omitted here. For ADSSL and BADSSL, maintenance costs are the same and little larger than that of verification time.

In conclusion, BADSSL is comprehensively efficient in construction cost, query authentication cost and maintenance cost.

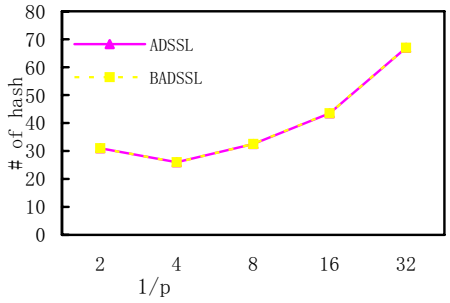


Figure 9. Maintenance time cost

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a dynamic main memory authentication data structure: ADSSL, which is not only space compact, but also efficient in query authentication cost and maintenance cost. To decrease query VO size of ADSSL, we propose an improved version: BADSSL, which supports a flexible trade-off between query verification cost and construction cost.

For now, ADSSL only supports authentication of range queries. How to support more types of queries, such as aggregate query, join queries is one of our future work. Moreover, here we assume the workload is uniform. When query distribution is non-uniform, or even skewed, how to adapt our authentication structure dynamically is one interesting and useful research topic.

REFERENCES

[1] H. Hacigumus, B. Iyer and S. Mehrotra, "Providing database as a service," in Proc. of 18th ICDE, San Jose, CA, USA, February 2002.

[2] F. Li, M. Hadjieleftheriou, G. Kollios, L. Reyzin, "Dynamic authenticated index structures for outsourced databases". SIGMOD, 2006.

[3] H. Pang, KL.Tan Authenticating query results in edge computing. In Proc. of International Conference on Data Engineering (ICDE), pages 560-571, 2004.

[4] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. Commun. ACM, 33(6):668–676, 1990.

[5] MT. Goodrich, R. Tamassia. "Efficient authenticated dictionaries with skip lists and commutative hashing". Technical report, Johns Hopkins Information Security Institute, 2000.

[6] E. Mykletun, M. Narasimha, G. Tsudik. "Authentication and Integrity in Outsourced Databases". In Network and Distributed Systems Security, 2004.

[7] M. Narasimha, G. Tsudik. "DSAC: Integrity of Outsourced Databases with Signature Aggregation and Chaining". In Proc. of Conference on Information and Knowledge Management (CIKM), pages 235-245, 2005.

[8] H. Pang, A. Jain, K. Ramamritham, KL Tan. "Verifying completeness of relational query results in data publishing". In SIGMOD Conference, pages 407–418. ACM, 2005.

[9] E. Mykletun, M. Narasimha, G. Tsudik. "Signature bouquets: Immutability for aggregated/condensed signatures". In European Symposium on Research in Computer Security (ESORICS), pages 160-176, 2004.

[10] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, SG. Stubblebine. "A general model for authenticated data structures". Technical report, CSE-2001-9 UC Davis Department of Computer Science December 6 2001.

[11] R. Sion. "Query execution assurance for outsourced databases". In VLDB, pages 601–612. ACM, 2005.

[12] P. Devanbu, M. Gertz, C. Martel, SG. Stubblebine. "Authentic third-party data publication". In IFIP Workshop on Database Security, pages 101-112, 2000.

[13] M. Xie, HJ Wang, J. Yin, XM Meng. "Integrity auditing of outsourced data". In Proceedings of the 33rd international conference on VLDB 2007, Pages 782-793