Separating Authentication from Query Execution in Outsourced Databases

Stavros Papadopoulos[#1], Dimitris Papadias[#2], Weiwei Cheng[*3], Kian-Lee Tan[*4]

[#] *Hong Kong University of Science and Technology*
[1] stavros@cse.ust.hk
[2] dimitris@cse.ust.hk

[*] *National University of Singapore*
[3] chengwei@comp.nus.edu.sg
[4] tankl@comp.nus.edu.sg

Abstract **— In the database outsourcing paradigm, a data owner (*DO*) delegates its DBMS administration to a specialized service provider (*SP*) that receives and processes queries from clients. The traditional outsourcing model (TOM) requires that the *DO* and the *SP* maintain authenticated data structures to enable authentication of query results. In this paper, we present SAE, a novel outsourcing model that separates authentication from query execution. Specifically, the *DO* does not perform any task except for maintaining its dataset (if there are updates). The *SP* only stores the *DO*'s dataset and computes the query results using a conventional DBMS. All security-related tasks are outsourced to a separate *trusted entity* (*TE*), which maintains limited authentication information about the original dataset. A client contacts the *TE* when it wishes to establish the correctness of a result returned by the *SP*. The *TE* efficiently generates a *verification token* of negligible size. The client can verify the token with minimal cost. SAE eliminates the participation of the *DO* and the *SP* in the authentication process, and outperforms TOM in every aspect, including processing cost for all parties involved, communication overhead, query response time and ease of implementation in practical applications.**

I. INTRODUCTION

Instead of administrating their data locally, several organizations *outsource* DBMS management to third-party service providers that receive and process queries from clients. The providers are not necessarily trustworthy and, thus, they should be able to prove that the results are *sound* (i.e., unaltered and containing no bogus data) and *complete* (i.e., all records satisfying the query are present). We refer to a sound and complete query result as *correct*. Figure 1 illustrates the traditional outsourcing model (TOM). The data owner (*DO*) builds an *authenticated data structure* (*ADS*) over its dataset. The *ADS* is a conventional index, augmented with hash values or signatures generated with a public-key cryptosystem (e.g., RSA). The *DO* transmits its dataset and signatures to the service provider (*SP*), which constructs the *ADS* locally and utilizes it to compute the result of each incoming query, as well as a *verification object* (*VO*). The *VO* contains authentication data (i.e., hashes/signatures) for proving the correctness of the query result. In case of updates, the *DO* generates new signatures, modifies its *ADS*, and notifies the *SP* that updates its *ADS* accordingly.

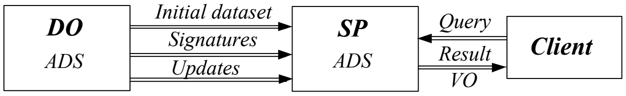


Fig. 1 The traditional database outsourcing model

Several *ADS* have been proposed for range query authentication [2, 3, 4, 5, 6, 7, 8, 10, 11]. The current state-of-the-art for disk-based datasets is the *MB-Tree* [5], which combines the concepts of the B^+-Tree and the *Merkle Hash Tree* [7]. A leaf node entry in the MB-tree is associated with a digest computed on the binary representation of the corresponding record, using a *one-way*, *collision-resistant hash function*. An intermediate node entry is associated with a digest computed on the concatenation of the digests in the page it points to. The *DO* signs the digest h_{root} associated with the root. The *DO* and the *SP* maintain identical copies of the MB-tree and the signature. Assume a 1D range query q corresponding to the result $RS = \{r_i, r_{i+1,\dots,} r_j\}$. In addition to *RS*, the *SP* returns to the client a *VO* that contains: (i) the two *boundary records* r_{i-1}, r_{j+1} that enclose *RS*, (ii) the digests of all left siblings of every visited entry in the path from the root to r_{i-1}, (iii) the digests of all right siblings of every visited entry in the path from the root to r_{j+1}, and (iv) the *DO*'s signature. The client can re-construct h_{root} from *RS* and the digests in the *VO*, and match it against the signature. Soundness is established by hash collision resistance. Completeness is guaranteed through the presence of the two boundary records.

TOM suffers from several drawbacks: (i) the *DO* is actively involved in the framework, since it has to build and maintain an *ADS* locally, thus, defeating the purpose of outsourcing; (ii) the *SP* must modify its DBMS in order to embed *ADS* functionality; (iii) the *VO* is typically large, imposing substantial communication overhead; (iv) the *ADS* exhibits inferior performance with respect to a conventional index. Motivated by these shortcomings, we propose SAE, a novel outsourcing model that offers a wide range of benefits. SAE separates authentication from query execution by exploiting trustworthy organizations with expertise on security issues. We henceforth refer to such an organization as a *trusted entity* (*TE*). Although a *TE* possesses up-to-date resources and know-how on security standards, cryptographic libraries, etc., it does not necessarily have the infrastructure to manage large databases and high query loads. Therefore, SAE assigns to the *TE* only the authentication process, which involves little computational effort compared to the actual query processing performed at the *SP*. Next, we present the main concepts of SAE, focusing on 1D range queries.

II. SAE

Figure 2 outlines the basic functionality of SAE. The *DO* transmits a relational table *R* (to be outsourced) to the *SP*, which stores *R* in a conventional DBMS. The *DO* also sends the dataset to the *TE*. For each record $r_i \in R$, the *TE* generates a tuple $t_i = <t_i.id, t_i.a, t_i.h>$, where: (i) $t_i.id$ is the unique identifier of r_i, (ii) $t_i.a$ is the value of the query attribute in r_i (i.e., $r_i.a$), and (iii) $t_i.h$ is computed by applying a (one-way, collision-resistant) hash function on the binary representation of r_i. For instance, suppose that the *DO* is a consumer electronics shop, and *R* is a relation of digital camera specifications that contains columns (*id*, *manufacturer*, *model*, *price*). Let *price* be the query attribute and r_m = (15, "Canon", "SD850 IS", 250) be an arbitrary record in *R*. The *SP* stores the entire r_m, whereas the *TE* keeps t_m = (15, 250, h_m), where h_m is computed on the binary representation of r_m, and discards the other attribute values (i.e., *manufacturer, model*). We denote the set of tuples *t* generated from the records in *R* as *T*.

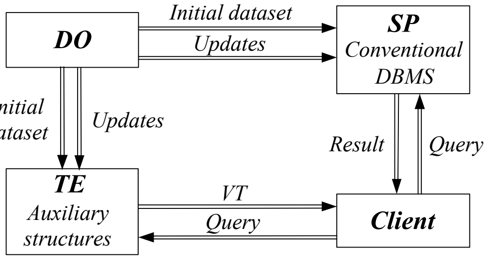


Fig. 2 The entities and their interaction in SAE

Clients issue queries directly to the *SP*, which sends back only the results. Let *q* be a range query on attribute *a*. $RS = \{r_i, r_{i+1}, \ldots, r_j\} \subseteq R$ signifies the actual result, and |*RS*| its cardinality. Continuing the above example, *q* could be *"select all cameras from R whose price is between 200 and 300 euros"*. Assume that after processing *q*, the *SP* returns a set RS^{SP} to the client, which may be different from the correct result *RS*, if the *SP* is malicious. In order to verify RS^{SP}, the client sends *q* to the *TE*[1]. Let $S_\oplus$ be the exclusive-OR (XOR) of the digests of tuples in a set *S*. Upon receiving *q*, the *TE* first determines set $TS = \{t_i, t_{i+1}, \ldots, t_j\} \subseteq T$, whose tuples correspond to the records of *RS*. It then produces a *verification token* $VT = TS_\oplus = t_i.h \oplus t_{i+1}.h \oplus \ldots \oplus t_j.h = RS_\oplus$, which captures authentication information about *RS*. *VT* consumes just a few bytes (i.e., the size of a digest) independently of the result size and, as we show in Section III, it can be computed very efficiently by utilizing auxiliary structures. The *TE* finally transmits the *VT* to the client.

The client computes $RS^{SP}_\oplus$ (i.e., the XOR of the digests of the records in RS^{SP}) locally, and matches it against the *VT*. Assume that a malicious *SP* returns to the client a corrupt result $RS^{SP} = (RS\text{-}DS) \cup IS$, where DS ($\subseteq RS$) is a subset of the actual results and *IS* is a set of fake tuples. By removing *DS* from *RS*, the *SP* attacks the result *completeness*, whereas by injecting *IS* it attacks *soundness*. Note that modifying a tuple $r \in RS$ to r' is equivalent to including *r* into *DS*, and r' into *IS*. The *SP* can escape detection, if and only if the *VT* produced for *RS* (i.e., $RS_\oplus$) is equal to $RS^{SP}_\oplus$. This happens if and only if:

$$RS_\oplus = ((RS\text{-}DS) \cup IS)_\oplus \Leftrightarrow RS_\oplus = RS_\oplus \oplus DS_\oplus \oplus IS_\oplus$$

$$\Leftrightarrow DS_\oplus = IS_\oplus$$

In [9], we prove that it is computationally infeasible for the *SP* to find sets of records *DS* and *IS* such that $DS_\oplus = IS_\oplus$; therefore, SAE is secure. Furthermore, compared to TOM, SAE has numerous advantages including the following:

- The *DO* has a minimal participation, as it simply transmits its dataset (and updates, if any) to the *SP* and the *TE*, without having to compute authentication information and maintain a sophisticated *ADS* locally.
- The *SP* does not need specialized infrastructure. Since there is no overhead of authentication information, query processing is as fast as in conventional database systems.
- The transmission overhead is very small, because the size of the *VT* is negligible compared to that of typical *VO*s. Furthermore, the client can only verify the queries of its choice, whereas in TOM the authentication overhead occurs for all queries.

III. XB-Tree

Recall that the *TE* maintains a set *T* of tuples $t_i = <t_i.id, t_i.a, t_i.h>$, each corresponding to a record $r_i \in R$. In order to compute the *VT* for a query *q*, the *TE* could perform a sequential scan of *T* and retrieve the digests of all records qualifying *q*. Although the size of t_i is usually much smaller than that of r_i, this scan can be expensive contradicting the goal of SAE, which requires that the effort of the *TE* should be minimal compared to that of the *SP*. Towards this goal, we propose the *XOR B-Tree* (*XB-Tree*), a disk-based index that organizes XOR values. Each entry *e* of an intermediate node has the form $<e.sk, e.L, e.X, e.c>$[2], where: (i) *e.sk* is a search key, (ii) *e.L* is a pointer to a disk page containing the ids and digests of the tuples in *T* with *a* values equal to *e.sk*, (iii) *e.X* is a bit string, and (iv) *e.c* is a pointer to the child node of *e*. Let e_{i-1}, e_i be two successive entries in the same intermediate node. The sub-tree rooted at node $e_i.c$ ($e_{i-1}.c$) contains entries whose search keys are larger (smaller) than $e_i.sk$. $e_i.X$ is the XOR of the digests of the tuples in $e_i.L$ and the *X* values of the entries in $e_i.c$. In other words, $e_i.X$ represents the XOR of the tuples

[1] Query processing at the *SP* and *VT* generation at the *TE* are independent; therefore, the client sends the query to both the *TE* and the *SP* simultaneously, in order to reduce the response time.

[2] Except for the first entry e_0 that has the form < $e_0.X$, $e_0.c$>. For leaf nodes, $e_0.X$ is 0 and $e_0.c$ is null.

with search keys larger than or equal to $e_i.sk$, and (strictly) less than $e_{i+1}.sk$. Leaf entries are similar, except that their child pointers are *null*. Figure 3 outlines the features of an XB-Tree indexing tuples t_1 - t_{14}, with search keys {1, 3, 3, 6, 6, 12, 13, 15, 18, 18, 20, 23, 23, 25}, respectively. For simplicity, we assume that the fanout is 3, but for typical disk page sizes, the number of entries per node is in the order of 100. For instance, in case of entry e_4: (i) $e_4.sk = 6$, (ii) $e_4.L$ points to the page that accommodates tuples {<$t_4.id$, $t_4.h$>, <$t_5.id$, $t_5.h$>}, (iii) $e_4.c$ points to node N_5, (iv) $e_4.X$ is equal to $e_4.L_\oplus \oplus e_{11}.X \oplus e_{12}.X = t_4.h \oplus t_5.h \oplus t_6.h$ ($e_4.L_\oplus$ denotes the XOR of the digests of the tuples in $e_4.L$). Finally, observe that for the first entry e_3 of node N_2: $e_3.X = e_8.X \oplus e_9.X \oplus e_{10}.X$ and $e_3.c = N_4$, but there are no $e_3.sk$ and $e_3.L$ attributes.

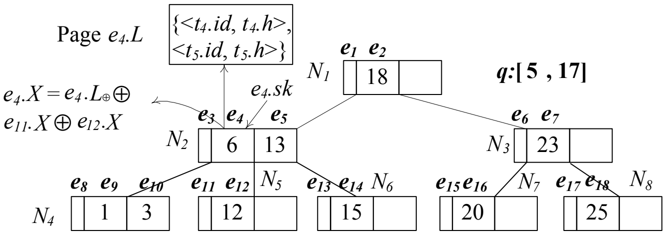


Fig. 3 XB-Tree example

To produce the *VT* for a query q:[ql, qu], where ql (qu) is q's lower (upper) bound, the *TE* executes algorithm *GenerateVT* of Figure 4. The procedure is recursive and takes as initial arguments: q, the root node N of the XB-Tree, and bitstring VT=0. Note that *VT* is passed by reference. Let f be the number of entries in N. For simplicity, in the pseudo-code we assume that $e_0.sk = -\infty$ (although the XB-Tree does not store $e_0.sk$), and that there is a fictitious entry e_f (also not existing in the tree) with $e_f.sk = +\infty$. The loop in lines 1-7 scans the f entries of N, and distinguishes two cases in which it updates *VT*: (i) If ql is less than or equal to $e_i.sk$, and qu is greater than or equal to $e_{i+1}.sk$, then all the tuples in $e_i.L$ as well as in the subtree rooted at $e_i.c$ are present in the result. Therefore, *GenerateVT* XORes $e_i.X$ to *VT* (lines 2-3). (ii) If $e_i.sk$ is enclosed in q (but not the subtree pointed by $e_i.c$), the algorithm sets *VT* to $VT \oplus e_i.L_\oplus$ (lines 4-5). Lines 6-8 simply ensure that the procedure recursively visits the nodes at deeper levels of the XB-Tree.

```
GenerateVT (RangeQuery q, XBNode N, &VT)
1.  For $i = 0, 1, \ldots, f-1$
2.      If $(q.ql \le e_i.sk)$ and $(q.qu \ge e_{i+1}.sk)$
3.          $VT = VT \oplus e_i.X$
4.      Else if $(q.ql \le e_i.sk)$ and $(q.qu \ge e_i.sk)$)
5.          $VT = VT \oplus e_i.L_\oplus$
6.      If ($(q.ql > e_i.sk)$ and $(q.ql < e_{i+1}.sk)$) or
7.          ($(q.qu > e_i.sk)$ and $(q.qu < e_{i+1}.sk)$)
8.          If ($e_i.c \ne null$), GenerateVT($q$, $e_i.c$, $VT$)
```

Fig. 4 Algorithms for generating the *VT*

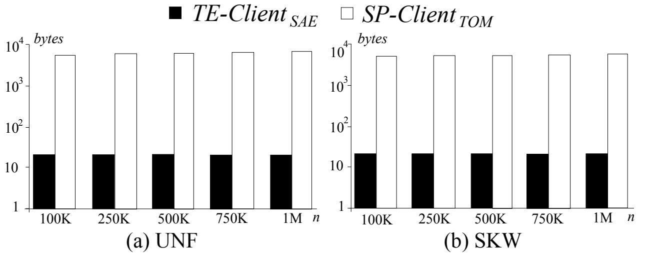
We demonstrate *GenerateVT* with the example of Figure 3, for query q:[5, 17]. Upon the first procedure call, $VT = 0$ and $N = N_1$. Since no search key in N_1 or subtree rooted at an N_1's entry is enclosed in q, the algorithm does not update *VT*, and recursively continues in N_2. Because both $e_4.sk = 6$ and $e_5.sk = 13$ lie in q, *GenerateVT* first computes $VT = VT \oplus e_4.X = 0 \oplus e_4.L_\oplus \oplus e_{11}.X \oplus e_{12}.X = t_4.h \oplus t_5.h \oplus t_6.h$ (lines 2-3 in Figure 4) since, except for the records in $e_4.L$, the tuples in the subtree rooted at $e_4.c$ (i.e., node N_5) are contained in q as well . The algorithm next calculates $VT = VT \oplus e_5.L_\oplus = (t_4.h \oplus t_5.h \oplus t_6.h) \oplus t_7.h$ (lines 4-5), in order to include the digests of the tuples with search key 13 in *VT*. Subsequently, the procedure visits N_4 and N_6. Since $ql > e_{10}.sk = 3$, *GenerateVT* does not perform any action in N_4. On the other hand, in N_6, it first updates *VT* to $VT \oplus e_{13}.X = VT \oplus 0 = t_4.h \oplus t_5.h \oplus t_6.h \oplus t_7.h$ (lines 2-3), and then evaluates $VT = VT \oplus e_{14}.L_\oplus = (t_4.h \oplus t_5.h \oplus t_6.h \oplus t_7.h) \oplus t_8.h$ (lines 4-5), because $e_{14}.sk = 15$ satisfies q. The algorithm then terminates. Observe that the final *VT*, i.e., $t_4.h \oplus t_5.h \oplus t_6.h \oplus t_7.h \oplus t_8.h$, is the desired one, as it represents the XOR of the digests of the tuples in q's actual result.

GenerateVT visits $O(\log_{f_{XB}} K)$ nodes independently of the query result size, where K is the number of tree nodes and f_{XB} is the maximum fanout of the XB-Tree (some traversals may not reach the leaf level). Furthermore, the XB-Tree supports fast insertion and deletion operations in $O(\log_{f_{XB}} K)$ time, utilizing the standard insertion and deletion algorithms of the traditional B-Tree, subject to some modifications concerning the proper updating of the X values. Since the *TE* maintains only two attributes (search key, id) and a digest for each record, the size of T is small compared to the original dataset R, especially if R contains numerous or large (e.g., text, images) attributes.

IV. Experimental Results

We compare SAE with TOM using an Intel Core Duo 2GHz, with 2GB of RAM. All cryptographic components are implemented with the Crypto++ library [1]. A digest consumes 20 bytes for both SAE and TOM. Each record contains a search key (i.e., a value on the range query attribute) and additional attributes. The search keys are integers (4 bytes) in the domain [0, 10^7]. The total record size is set to 500 bytes. We use two datasets: (i) UNF, where the search keys follow a uniform distribution, and (ii) SKW, where the keys are generated using ZIPF, with the skewness parameter set to 0.8 (i.e., so that 77% of the search keys are concentrated in 20% of the domain). We evaluate the performance of SAE and TOM for different dataset cardinalities (n). For each experiment, we perform 100 uniform queries with extent 0.5% of the entire domain, and present the average cost over all measurements. In TOM, the *SP* indexes records with an MB-Tree, while in SAE with a B^+-Tree. The *TE* (in SAE) uses an XB-Tree. All indexes are disk-based using pages of 4096 bytes. When measuring processing cost, we charge 10 milli-seconds for each node access.

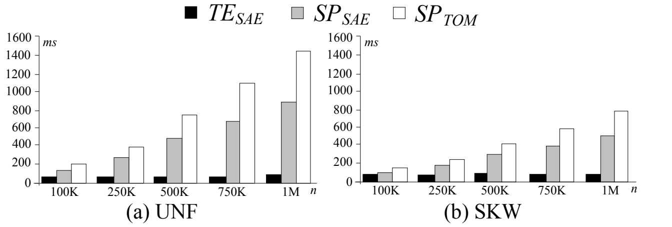
Figure 5 illustrates the communication overhead between the pair (*TE*, client) in SAE and (*SP*, client) in TOM. This overhead refers only to the authentication information and excludes the cost of transmitting the result. In SAE, the only authentication information exchanged is the *VT*, which is 20 bytes irrespectively of the cardinality of the result set. This overhead is negligible compared to the *VO* size in TOM, which is 2-3 orders of magnitude higher.



(a) UNF (b) SKW

Fig. 5 Communication overhead vs. *n*

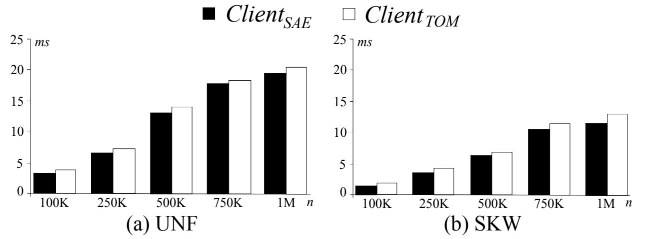
Figure 6 depicts the query processing overhead (in milliseconds) versus *n*. For SAE, we include the costs at the *SP* (for computing the result) and the *TE* (for producing the *VT*). TOM involves a higher cost at the *SP* because of the lower fanout of the MB-Tree compared to the B^+-Tree in SAE. In particular, SAE reduces the burden at the *SP* by 30%-39% in UNF, and 24%-37% in SKW. The computational effort at the *TE* is negligible because *VT* generation always involves two tree traversals. On the other hand, both the B^+-Tree and the MB-Tree entail two additional scans; (i) at the leaf level of the index, and (ii) in the dataset file for retrieving the results.



(a) UNF (b) SKW

Fig. 6 Query processing time vs. *n*

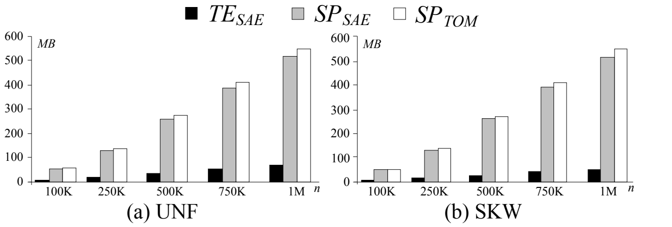
Figure 7 shows the verification cost as a function of the dataset cardinality. As *n* increases, more records satisfy the query, which has fixed extent. The verification cost is linearly dependent on the result size for both methods. In SAE the client has to compute a digest for each record received by the *SP*. In TOM, the client must also re-construct the root digest of the MB-Tree and verify it against the *DO*'s signature. The verification times are lower for SKW, because the average result cardinality is smaller and, thus, the client computes fewer digests.



(a) UNF (b) SKW

Fig. 7 Verification time vs. *n*

Figure 8 evaluates SAE and TOM with respect to storage cost. Both methods require similar total space at the *SP*, because the consumption is dominated by the outsourced dataset (i.e., the additional authentication information is small compared to the actual records). Furthermore, recall that the *TE* maintains only the digest of each record. Therefore, its storage requirements are minor compared to that of the *SP*, implying that the *TE* can maintain a main memory index (instead of the disk-based XB-Tree).



(a) UNF (b) SKW

Fig. 8 Storage cost vs. *n*

To conclude, in addition to its other benefits, SAE also outperforms TOM in every performance metric. The most significant savings refer to (i) the network overhead, where the *VT* is up to 3 orders of magnitude smaller than the average *VO*, and (ii) the computational burden at the *SP*, where SAE achieves reductions up to 39% of the total processing time. The above enable the client to experience a lower response time (i.e., interval between query transmission and result verification). Finally, our experiments show that the *TE* in SAE consumes negligible resources in comparison to the *SP*.

V. Conclusion

We propose SAE, a novel outsourcing model that separates authentication from query execution. SAE minimizes the participation of the *DO* in the authentication process, and eliminates the need of specialized infrastructure on behalf of the *SP*. Furthermore, it outperforms the conventional model on all metrics, achieving significant gains for the communication cost and processing effort at the *SP*, which constitute the most important factors.

Acknowledgments

This work was supported by grant HKUST 6181/08 from Hong Kong RGC.

References

[1] Crypto++ Library. *www.eskimo.com/~weidai/benchmark.html*

[2] Cheng, W., Pang, H., Tan, K.-L. Authenticating Multi-Dimensional Query Results in Data Publishing. *DBSec*, 2006.

[3] Devanbu, P., Gertz, M., Martel, C., Stubblebine, S. Authentic Data Publication Over the Internet. *Journal of Computer Security* 11(3): 291-314, 2003.

[4] Goodrich M., Tamassia R., Triandopoulos N., Cohen R. *Authenticated Data Structures for Graph and Geometric Searching*. CT-RSA, 2003.

[5] Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L. Dynamic Authenticated Index Structures for Outsourced Databases. *SIGMOD*, 2006.

[6] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., Stubblebine, S. A General Model for Authenticated Data Structures. *Algorithmica*, 39(1): 21-41, 2004.

[7] Merkle, R. A Certified Digital Signature. *CRYPTO*, 1989.

[8] Pang, H., Tan, K.-L. Authenticating Query Results in Edge Computing. *ICDE*, 2004.

[9] Papadopoulos, S, Papadias, D., Cheng, W., Tan, K-L. Separating Authentication from Query Execution in Outsourced Databases. *Technical Report, HKUST-CS08-06*, 2008.

[10] Tamassia, R., Triandopoulos, N. Efficient Content Authentication over Distributed Hash Tables. *International Conference on Applied Cryptography and Network Security*, 2007.

[11] Yang, Y., Papadopoulos, S., Papadias, D., Kollios, G. Spatial Outsourcing for Location-based Services. *ICDE*, 2008.