2012

Authenticated Top-K Aggregation in Distributed and Authenticated Top-K Aggregation in Distributed and

Sunoh Choi

Hyo-Sang Lim

Elisa Bertino
bertino@purdue.edu

2012 ASE/IEEE International Conference on Social Computing and 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust

Authenticated Top-*K* Aggregation in Distributed and Outsourced Databases

Sunoh Choi
ECE Department
Purdue University
West Lafayette IN USA
choi39@purdue.edu

Hyo-Sang Lim
CTE Division
Yonsei University
Wonju Kangwon Korea
hyosang@yonsei.ac.kr

Elisa Bertino
CS Department
Purdue University
West Lafayette IN USA
bertino@cs.purdue.edu

***Abstract*—Top-*k* queries have attracted interest in many different areas like network and system monitoring, information retrieval, sensor networks, and so on. Since today many applications issue top-*k* queries on distributed and outsourced databases, authentication of top-*k* query results becomes more important. This paper addresses the problem of authenticated top-*k* aggregation queries (e.g. "find the *k* objects with the highest aggregate values") in a distributed system. We propose a new algorithm, called Authenticated Three Phase Uniform Threshold (A-TPUT), which provides not only efficient top-*k* aggregation over distributed databases but also authentication on the top-*k* results. We also introduce several enhancements for A-TPUT to reduce both the computation cost and the communication cost. Finally, we confirm the efficiency of our solutions through an extensive experimental evaluation.**

Keywords-component; Top-k, Authentication, Distributed Databases, Outsourced Databases

I. INTRODUCTION

The results of a top-*k* query are *k* objects that have the highest overall scores. Top-*k* queries have attracted much interest in many different areas like network and system monitoring [2], information retrieval [5], sensor networks [12, 13] and so on. The main reason for such interest is that they reduce the overhead by pruning away uninteresting answers.

As network and ubiquitous environments are emerging, the objects which are subjects of top-*k* queries are distributed across nodes in the network. This means that the target of top-*k* queries is no longer a centralized database but distributed multiple databases. In such distributed environments, top-*k* aggregation first aggregates the scores for each object which resides in multiple distributed databases, and then, finds *k* objects whose aggregated score (mostly the sum of scores from distributed databases) is ranked within top-*k*. An example of top-*k* aggregation is the top-*k* query processing over a content distribution networks (CDN) for large enterprises. Large enterprises have branch offices located around the globe. The number of branch offices ranges from a few tens to a few thousands. Due to the diverse geographical locations of branch offices, the links between the offices may have low bandwidth and long round trip time. Successful operations of CDN rely on effective monitoring of the activities on the network which means that the central management station is often asked to answer "top-*k*" queries like "list the top-*k* most popular documents across the whole CDN". In this example, the score associated with a document is the number of downloads for the document.

The existing top-*k* aggregation algorithms mainly address efficiency issues such as how to find top-*k* objects with the least communication overhead. In the CDN example, if the number of documents runs to millions, a naïve method by which all data are transmitted to the central manager is inefficient. As we discuss in Section II, various algorithms have been developed to reduce the network communication costs. However, there is one more important issue in distributed top-*k* aggregation: the authentication issue.

In top-*k* aggregation, the multiple distributed databases can be autonomously managed and sometimes outsourced. Also, the data service provider (in short, DSP) which collects data from the databases and calculates top-*k* results can also be an independent party and can be outsourced for cost down. In such distributed and outsourced environment, the DSP or databases may be malicious or subverted by an adversary. Even if just one among the DSP and the databases is compromised, it could return tampered results, including: 1) incomplete results, 2) altered ranking, and 3) spurious results. If an attacker drops from the result some higher ranked objects, the user receives incomplete information. By tampering with the ranking order, the attacker can bias the results. In addition, the attacker may add fake information to the result.

In this paper we investigate algorithms that authenticate top-*k* aggregation results in distributed and outsourced databases. However, we address not only authentication but also efficiency. Our solution is based on a well-known top-*k* aggregation algorithm: the Three Phase Uniform Threshold (TPUT) algorithm [2]. We first develop an authenticated top-*k* aggregation algorithm based on TPUT. We call this algorithm A-TPUT. The main strength of A-TPUT is that 1) it provides the authentication capability which is not supported in the original TPUT algorithm and 2) it only requires a fixed number of communication rounds between the DSP and the databases regardless of the number of objects needed to find top-*k* results.

To develop an authenticated version of TPUT, we delicately integrate two authentication techniques. The first technique is the Merkle Hash Tree (MHT) which is a tree-based data structure for detecting tampering over a series of values. With MHT, we can guarantee the completeness and correctness of data communicated between trusted parties and untrusted parties. The second technique is the Condensed-RSA algorithm [14]. Condensed-RSA is a digital signature technique which is suitable for combining signatures generated by a single signer into a single condensed signature.

978-0-7695-4848-7/12 $26.00 © 2012 IEEE
DOI 10.1109/SocialCom-PASSAT.2012.103

We use this signature scheme to reduce the communication cost between trust parties and untrusted parties. By using Condensed-RSA, we can combine signatures of multiple objects and send only one digital signature (instead of multiple signatures) to reduce the communication cost.

Next, we develop an optimization technique for A-TPUT with regard to the communication between the databases and the DSP. Here, we reduce communication costs by increasing the threshold which is used to determine how many objects should be transmitted from the databases to the DSP. Higher threshold means less data transmission. We provide formal equations to find a higher threshold value, and then prove that A-TPUT always finds the genuine top-k aggregation results and correctly authenticates the results with the higher threshold value.

Through extensive experiments, we first show that our approach efficiently authenticates top-k aggregation. The results show that our approach significantly reduces not only communication costs but also response time compared to other algorithms.

Our contributions are summarized as follows:

- We develop authenticated top-k aggregation algorithms (A-TPUT and S-TPUT) using MHT and Condensed-RSA for distributed and outsourced databases. We also prove that A-TPUT and S-TPUT correctly authenticate the top-k results.
- We propose an optimization technique for A-TPUT and S-TPUT. This technique reduces communication costs between the databases and the DSP.
- With extensive experiments, we show the efficiency of our authentication algorithms and optimization technique.

The rest of the paper is organized as follows. Section II discusses related work on existing top-k aggregation algorithms and authentication for the top-k results. Section III presents our system model and attack model, and then briefly describes a basic top-k aggregation algorithm (i.e., TPUT) which is the basis for our approach. Section IV proposes our authenticated top-k aggregation algorithms (A-TPUT and S-TPUT). Section V presents an optimization technique to reduce the amount of data transmission for aggregating top-k results. Section VI reports the experimental results. Section VII summarizes and concludes the paper.

II. Related Work

Work related to our approach falls into two categories: (i) top-k aggregation algorithms and (ii) authentication for the top-k results.

Top-k aggregation algorithms: Efficient processing of top-k queries in distributed environments has received much attention [1,2,3,4,9]. A first important algorithm is Fagin's Algorithm (FA) [9] which models the general problem of answering top-k aggregation queries using distributed lists of data items sorted by their values. It first performs a sorted access to all lists and finds at least k data items retrieved in all lists. Then, it executes a random access to all the retrieved data items to guarantee that the values of the other items are less than the top-k values. The Threshold Algorithm (TA) [1] improves upon FA by using a threshold so as to stop at a higher position than FA. The Best Position Algorithm (BPA) [4] improves over TA by reducing the number of sorted accesses and random accesses. However a major drawback of BPA, as well as FA and TA, is that they require several communication rounds which incur very high communication costs especially when the number of databases is large.

Fagin et al. also suggested the No Random Access (NRA) algorithm [1]. It uses a threshold mechanism like TA, but unlike TA it does not require random accesses but only sorted accesses. Mamoulis et al. proposed the Lattice-based Rank Aggregation (LARA) in order to reduce the computational cost of NRA [3]. LARA has a computational cost of $O(2^m)$ where m is the number of databases as opposed to $O(n)$ required by NRA. When m is small, LARA is more efficient than NRA. But, when m is large, it is not more efficient than NRA.

The Three Phase Uniform Threshold (TPUT) algorithm [2] is an efficient algorithm to answer top-k queries in distributed systems. The algorithm reduces communication costs by pruning away ineligible data items and restricting the number of round-trip messages between the DSP and databases. However, TPUT does not provide any authentication mechanisms.

Yu et.al. proposed the Three Phase Adaptive Threshold algorithm (TPAT), the Three Phase Object Ranking based algorithm (TPOR), and the Hybrid-Threshold algorithm (HT) [16]. TPAT generalizes TPUT by utilizing summary statistics of the data. However, it could be very expensive to use summary statistics to accurately estimate data distributions. In TPOR, each database should send all data up to the last temporary top-k objects in phase 2. Therefore, TPOR performs worse than TPUT in the case when object rankings widely vary across all nodes. HT is proposed to combine the advantages of both TPUT and TPOR. Neumann et.al. suggested Adaptive Thresholds [17]. It is similar to TPAT, but does not require summary statistics. On the other hand, since the problem is NP-hard, it gives approximations based on a heuristic. In this paper, we give the exact solution and provide better optimization of TPUT than them.

Authentication for the top-k results: Verifying the completeness and authenticity of range and top-k queries results computed by untrusted third parties has also received much attention. Existing methods [6,7,8,13] for range query result verification fall in two categories. The first category includes methods that use the Merkle Hash Tree (MHT) [7,8]. The method by Li et al. [7] uses the more efficient Embedded Merkle B-tree (EMB-tree) structure. Such method indexes the data with a B^+-tree and then embeds an MHT into it. The second category includes methods based on signature-chaining schemes [6,13]. Assuming that a database is ordered according to an attribute the data owner hashes and signs every triple of consecutive data tuples. Li et al. [7] have shown however that signature-chaining approaches incur very high index construction cost, storage overhead, and user-side verification time.

Approaches for the authentication of top-k results have been proposed by Nath et al. [11] and Zhang et al. [12]. The former approach uses one-way chains and RSA for the authentication of the MAX aggregate functions and for top-k queries. The latter approach focuses on top-k query results authentication in two-tier sensor networks with the goal of reducing the communication overhead. However both these approaches only handles the situation where all the data items are unique. It means that every two nodes have disjoint sets of data items and there is no support for aggregation. By contrast, our A-TPUT allows that different databases have the same objects by considering the aggregate value for the object.

Pang and Mouratidis have proposed the Threshold with Random Access (TRA) algorithm and Threshold with No Random Access (TNRA) algorithm for authenticated top-k queries [5]. These algorithms are based on TA and NRA, respectively. However, since TA and NRA require an unpredictable number of rounds, TRA and TNRA are not suitable for distributed environments. By contrast, our A-TPUT algorithm only requires three rounds.

III. PRELIMINARIES

In this section, we first introduce our system model and the attack model. Then, we briefly discuss an unauthenticated top-k aggregation algorithm, TPUT, which is basis of our approaches.

A. System Model

We consider distributed environments for top-k query processing and authentication of the top-k results. In such distributed environments, our system model involves four parties: (i) the multiple data owners who provide data collection, (ii) the distributed databases which store a set of data, (iii) the data service provider (DSP) which processes top-k aggregation by communicating with the databases, and (iv) the users who issue top-k queries and receive the results from the DSP. In our system model, we assume that the databases and the DSP are not trusted since they can be outsourced. Figure 1 illustrates the four parties and data flows among them.

The data owners: Data owner DO_i manages a data collection D comprising n objects: $D = \{O_1,O_2,\ldots,O_n\}$, $n \geq 1$. For example, objects can be web pages in a web server, inventory items, or people. Each object O is bound to a value V which is the measure for deciding top-k results. For example, the value can be the number of accesses for each web page, the number of inventory items, or the salary of an individual. To compute a top-k aggregation, the data owner DO_i provides a sorted list L_i defined as $L_i = [<O_1,V_1>,<O_2,V_2>,\ldots,<O_n,V_n>]$ such that: (a) $\forall 1 \leq j \leq n$, $L_i.O_j$ is an object in D and $L_i.V_j$ indicates the value bound to O_j; and (b) $\forall 1 \leq j \leq l \leq n$, $L_i.V_j \geq L_i.V_l$. The data owner DO_i also manages authentication information which we will discuss in the next sections. For simplicity, we assume that all the data owners have the same data collection (but different values may be bound to the same object by different data owners).

The databases: Each data owner transfers its own list and authentication information to its associated database for query outsourcing. The databases are distributed in the network and autonomously managed by different authorities. The databases can be compromised or the data stored by database can be tampered with. Therefore, we assume that the databases cannot be trusted. The role of databases is to provide (parts of) the lists and authentication information to the DSP on behalf of the data owners.

The data service provider (DSP): The DSP accepts top-k queries from users and returns the results to users. A top-k query is forwarded to the databases associated with the different data owners and the DSP computes the result based on the data obtained from the databases. The query result for Q returned to the user, R, is an ordered list of k entries, $R = [<O_1,V_1>,<O_2,V_2>,\ldots,<O_k,V_k>]$, in which $\forall 1 \leq j \leq k$, $R.O_j \in D$ is a result data item and $R.V_j \in \mathbb{R}$ is its corresponding aggregate value. We assume that the DSP also can be compromised and the top-k results can be tampered with since the DSP can be outsourced.

The users: A user issues a query Q specifying a value for parameter k and receives the result R from the DSP. The user needs to verify the top-k result R is correct in cases in which the databases and the DSP cannot be trusted.

A correct query result R should relate to the query Q and the data collection D as follows: The aggregate value of an object O is $V(O)=\sum_{j=1}^{m} O.V_j$ where m is the number of databases in the distributed system. The query result R is correct if and only if it satisfies the following conditions:

- The result entries are ordered according to non-increasing aggregated values, i.e., $\forall 1 \leq j \leq l \leq k$, $R.V_j \geq R.V_l$.
- All the objects that are excluded from R have lower aggregate values than the last entry in R, i.e., for any non-result object $O \in D$, it holds that $V(O) \leq R.V_k$.

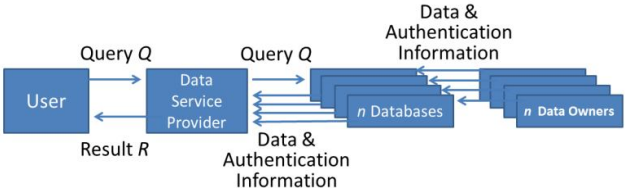


Figure 1. The System Model for Top-K Aggregation

B. Attack Model

As described in Section III.A, among the entities in our system model, the DSP and the databases are the potential adversaries as they could be subverted by attackers. The attacks can happen both in the databases and in the DSP as follows:

- In the databases, the adversaries may alter the lists. This means that values associated with objects may be altered or some objects and values may be omitted.
- In the DSP, the adversaries may execute the top-k aggregation query processing algorithm incorrectly or tamper with the results. This means that the order (i.e., ranking) of the top-k may be changed or some top-k results may be omitted.

Example 1: Assume that a top-3 query is given and assume that the correct result is $[<O_1,V_1>,<O_2,V_2>,<O_3,V_3>]$ where $V_1>V_2>V_3$. Now assume that a malicious DSP changes V_2 to V_2'

so that $V_1 > V_3 > V_2$'. In this case, even if the user still gets the correct set of top-k objects, the ordering of these objects in the result is not correct. In addition, a malicious DSP may drop the record $<O_3,V_3>$ from the result and add a record $<O_4,V_4>$ where $V_3 > V_4$. In this case, the user gets an incomplete result $[<O_1,V_1>,<O_2,V_2>,<O_4,V_4>]$.■

The goal of this paper is to protect top-k results against such attacks. To achieve this goal, we will allow the users (i) to verify the correctness of the query results and (ii) to check the completeness of the results.

C. The Three Phase Uniform Threshold (TPUT) Algorithm

The Three Phase Uniform Threshold (TPUT) algorithm [2] is an efficient top-k aggregation algorithm but it does not provide any authentication mechanism. We will use TPUT as the basis of our authenticated top-k aggregation algorithm since it is simple and has desirable features for distributed top-k aggregation such as a fixed number of communication rounds between the DSP and the databases.

The Three Phase Uniform Threshold (TPUT) algorithm [2] consists of three phases, each taking one round of communication:

- Phase 1: it establishes a lower bound on the true bottom. The DSP informs all databases that it would like to start computing a top-k query. Each database sends the top-k objects from its list. After receiving the data from all databases, the DSP calculates the partial sums of the values of the objects. Then, it looks at the k highest partial sums and takes the k-th one as the lower bound, denoted as t_1 and called "phase 1 bottom".
- Phase 2: it prunes away ineligible objects. The DSP sets a threshold $T = t_1/m$ and sends it to all databases. Each database sends back the list of objects whose values are greater or equal to T. The DSP then performs two tasks. First, it calculates partial sums for the objects. Let's call the k-th highest sum "phase 2 bottom" denoted by t_2. Clearly, $t_1 \leq t_2$. Then, it tries to prune away more objects by calculating the upper bounds of the objects. The objects whose upper bounds are less than t_2 are eliminated. The set of remaining objects is the candidate set S.
- Phase 3: it identifies the top-k objects. The DSP sends the set S to all databases and each database sends back the values of the objects in S. The DSP calculates the exact sum of the objects in S and selects the top-k objects.

Example 2: Consider the lists in Table I. A top-2 query is given. In phase 1, all databases send the data at positions 1 and 2 to the DSP. The DSP calculates the partial sums: $V(O_2) = 0.97$, $V(O_5) = 1.89$, $V(O_6) = 0.89$, and $V(O_0) = 1.59$. The two highest partial sums are 1.89 and 1.59 and the phase-1 bottom t_1 is 1.59. Then, the threshold T is set to 1.59/3 = 0.53.

In phase 2, database 1 does not send any data since it already sent the objects whose values are greater than T in phase 1 and databases 2 and 3 send data up to position 4. The DSP determines the phase-2 bottom as $t_2 = 1.59$ since $V(O_5) = 1.89$ and $V(O_0) = 1.59$ are the top-2 sums. Since the upper bounds of all objects which are retrieved are greater than 1.59, only two objects O_1 and O_4 which are not retrieved in phase 2 are eliminated and $S = \{O_2,O_5,O_6,O_0,O_3,O_7\}$.

In phase 3, the DSP sends S to all databases. The databases respond and the DSP concludes that the top-2 objects are O_5 and O_0 and the values are 2.33 and 1.95. Finally, they are returned to the user. ■

TABLE I. AN EXAMPLE DATA SET WITH THREE LISTS

Position	*List L_1*	*List L_2*	*List L_3*
1	$<O_2,0.97>$	$<O_5,0.97>$	$<O_5,0.92>$
2	$<O_6,0.89>$	$<O_0,0.80>$	$<O_0,0.79>$
3	$<O_7,0.45>$	$<O_3,0.70>$	$<O_3,0.72>$
4	$<O_5,0.44>$	$<O_7,0.65>$	$<O_7,0.64>$
5	$<O_0,0.36>$	$<O_2,0.52>$	$<O_6,0.29>$
6	$<O_1,0.28>$	$<O_4,0.22>$	$<O_2,0.28>$
7	$<O_3,0.19>$	$<O_1,0.12>$	$<O_1,0.24>$
8	$<O_4,0.13>$	$<O_6,0.01>$	$<O_4,0.01>$

IV. AUTHENTICATED TOP-K AGGREGATION

In this section, we introduce two mechanisms for supporting authentication in TPUT. The technical challenges for authentication are two folds: (1) to allow users to verify the completeness and the correctness of the top-k results and (2) to minimize data transmissions between the databases and the DSP. For addressing the former issue, we first introduce the Skewed Merkle Hash Tree (S-MHT), and then, for addressing the latter issue, we develop a mechanism to reduce data transmission that uses S-MHT and Condensed-RSA [14] together.

A. Authenticated TPUT (A-TPUT)

Our algorithm is based on TPUT extended by the use of the Skewed Merkle Hash Tree (S-MHT). Merkle Hash Tree is a data structure to prove completeness and correctness of a series of values by detecting tampering over the values. Therefore it is suitable for authenticating top-k query results. In the TPUT algorithm, we observe that the entries in the lists in the databases are sorted and accessed from the front. This means that to calculate top-k results with TPUT, we only need a partial list which begins from the first entry of the list. Based on this observation, we modify the original MHT structure to skew the tree from left to right (i.e., construct the tree structure from the first entries to the last entries) as shown in Figure 2.

Our S-MHT scheme works as follows. We compute a hash chain over the records in the list. We include the digest of each record in the digest computation of the record immediately ahead of it. Finally, the digest of the first record is signed by the private key of the data owner. This signature can be used to verify any j leading records of the list. The details are as follows.

Let n be the number of records in a list L_i.

$$Digest_{i,n} = h(O_{i,n} \mid V_{i,n})$$

$$Digest_{i,j} = h(O_{i,j} \mid V_{i,j} \mid Digest_{i,j+1}),\ 1 \leq j \leq n\text{-}1$$

$$Signature_i = Sign_{ski}(digest_{i,1})$$

These digests and the signature are computed by each trusted data owner and sent to the corresponding database. When a database i sends data up to position j, it sends the j-th digest and the signature of the data owner as well as the data. When

the DSP or the users receives them, they can verify that the data sent from the database i was not tampered.

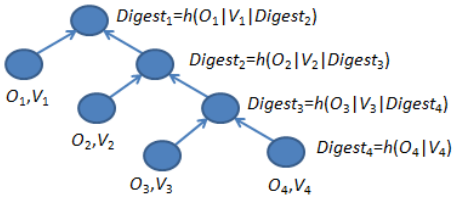


Figure 2. Skewed Merkle Hash Tree

We now introduce how to use S-MHT to develop A-TPUT. A-TPUT has three phases like TPUT. Only the phase 3 is modified for authentication.

- Phase 1: All databases send the top-k objects to the DSP and the DSP compute the phase-1 bottom t_1 (same to the original TPUT).
- Phase 2: The DSP sends the threshold $T = t_1/m$ to all databases and the databases send the objects having values greater or equal than T. Then, the DSP computes the phase-2 bottom t_2 and prunes away objects whose upper bounds are less than t_2. The remaining objects are included in set S (same to the original TPUT).
- Phase 3: The DSP sends S to all databases and each database sends the *sequence* containing the objects corresponding to set S. In addition, for authentication, each database sends its signature and a digest corresponding to the last object which is located in the lowest position in the sequence.

Figure 3 formally illustrates the algorithm for A-TPUT with S-MHT. Phase 1 is step 1, phase 2 is from step 2 to 5, and phase 3 is from step 6 to 7.

Algorithm A-TPUT with S-MHT
1. Request the local top-k objects to all databases;
2. Compute a threshold $T=t_1/m$ where t_1 is phase-1 bottom;
3. Request objects whose values $\geq T$ to all databases;
4. Compute phase-2 bottom t_2;
5. Prune objects whose upper bounds are less than t_2;
6. Request each *sequence* containing remaining objects, a digest, and a signature from each database;
7. Report each *sequence*, each digest, and each signature for each database to the user;

Figure 3. A-TPUT (with S-MHT) algorithm

Example 3: To illustrate the algorithm, consider the lists in Table 1. The first two phases of A-TPUT are the same as in TPUT. However, in phase 3, since $S = \{O_2,O_5,O_6,O_0,O_3,O_7\}$, the database 1 should send a *sequence* containing objects up to position 7, $Digest_{1,7}$, and the signature. The database 2 has to send a *sequence* containing objects up to position 8, $Digest_{2,8}$, and the signature. Finally the database 3 should transmit a *sequence* containing objects up to position 6, $Digest_{3,6}$, and the signature. When the DSP receives the sequences from the databases, it forwards them to the user that can then verify the top-k result.■

The following theorem establishes the correctness of algorithm A-TPUT with S-MHT (i.e., the algorithm authenticates top-k results.)

Theorem 1. *A-TPUT correctly authenticates the top-k objects.*

Proof (sketch). We prove the theorem for all integer $x\in[1,n]$: x is determined as the threshold and the remaining objects in phase 3. If the user accepts a sequence from the database i in A-TPUT, then conditioned upon the top-(x-1) values being correct, V_x must be the value of the object O_x with the x-th largest value. Trivially top-1 value is proved to be correct by $Digest_{i,1}$ and $Signature_i$. Obviously, the theorem can be trivially proved via an induction on x.

Let the object with the x-th largest value be object O_x and let its value be V_x. Let O_x' and V_x' be the corresponding answers returned by the DSP. They may be forged. We prove by contradiction. Assume that $V_x' \neq V_x$. The user must successfully verify $Digest_x$' for V_x'. But, when the user checks whether $Verify_{pk}(Digest_1', Signature) = 1$, since $Digest_1' = h(\ldots h(O_x'|V_x'|\ Digest_{x+1}))$ is different from $Digest_1$, the verification fails.

On the other hand, an adversary may drop an object at the bottom of the sequence. The smallest value in the sequence should be less or equal to the local top-k value and the threshold by the definition of A-TPUT. In addition, the sequence should contain the top-k objects and the upper bounds of all remaining objects should be greater or equal to the smallest top-k value. When the adversary drops an object, these conditions may not be satisfied. If these conditions are not satisfied, it means that the result was forged or an object was dropped.■

Compared to the existing authenticated top-k aggregation algorithms in outsourced databases, TRA and TNRA [5], in our algorithm, the databases may send more data than TNRA. However the response time of A-TPUT is much less than TRA and TNRA since our algorithm has the strength on the fixed number of data communication rounds in distributed environments. TRA and TNRA are based on TA and NRA [1] and the latency of TRA and TNRA is unpredictable because the number of rounds varies by data input. The response time consists of several round trip times. Each round trip time contains transmission time, propagation delay, and computation time at the DSP. In distributed environments, the propagation delay is usually much longer than the transmission time. For example, when the distance is 1000km, the bandwidth is 100Mbps, and we send a packet of size 100Bytes, then, the propagation delay is about 4ms and the transmission time is 0.008ms. Even if databases send k records every round as TPUT, the propagation delay is much longer than the transmission time. Moreover, TA and NRA has much more rounds than TPUT. As the number of rounds increases, the response time increases. So, for distributed databases, TRA and TNRA are not desirable. We will show the advantage of our algorithm in the experimental section.

B. Signature-based TPUT (S-TPUT)

One weak point of A-TPUT is the number of data entries which have to be transmitted from databases to DSP. This number depends on the threshold T in phase 2 and the set of remaining data S in phase 3. Here, we focus on the set S of phase 3. We will focus on T in Section V.

We note that in A-TPUT the amount of data transmission does not depend on the number of objects in S but depends on

the lowest rank in S. In our basic S-MHT based algorithm, even though the number of remaining objects in S is small, if the rank of an object in S is low, the databases should send a lot of data to the DSP. This is because we should send the partial list which begins from the first entry to the entry which has the lowest rank in S to authenticate the results (especially, completeness). This means that we cannot omit any entry between the first entry and the least ranked entry.

So, in this subsection, we exploit a signature-based technique to address this problem (i.e., allowing us to omit useless entries in the list). In our approach, data owners additionally sign each tuple using Condensed-RSA [14]. The Condensed-RSA scheme is a simple extension of the standard RSA scheme. One of the well-known features of RSA is its multiplicative homomorphic property. This property makes RSA suitable for combining signatures generated on each data item in a set by a single signer into a single condensed signature. Having successfully verified a condensed signature, a user can be assured that each data covered by the condensed signature was signed by the data owner.

Standard-RSA: A data owner has a public key $pk = (n', e)$ and a secret key $sk = (d)$, where n' is a k'-bit modulus computed as the product of two random k'/2-bit primes p and q. The respective public and secret exponents $e, d \in Z_{n'}^*$ satisfy $ed \equiv 1 \bmod \emptyset(n')$ where $\emptyset(n') = (p-1)(q-1)$. An RSA signature is computed over the hash of the input message.

Let $h()$ denote a suitable cryptographic hash function such as MD5 or SHA-1 which produces a fixed-length output $h(m^*)$ upon a variable-length input m'. A standard RSA signature on message m^* is computed as: $\sigma = h(m^*)^d \pmod{n'}$. RSA signature verification involves checking that $\sigma^e \equiv h(m^*) \bmod n'$.

Condensed-RSA: Given j input messages $\{m_1,\ldots,m_j\}$ and their corresponding signatures $\{s_1,\ldots,s_j\}$, a Condensed-RSA signature is given by the product of the individual signatures:

$$s_{1,j} = \prod_{l=1}^{j} s_l \pmod{n'}$$

The resulting signature $s_{1,j}$ has the same size as a standard RSA signature. When verifying a condensed signature, the verifier needs to multiply the hashes of all input data and check that:

$$(s_{1,j})^e \equiv \prod_{l=1}^{j} h(m_l) \pmod{n'}$$

Now we will explain S-TPUT algorithm. In phase 3, when the DSP requests data corresponding to remaining objects, each database computes a Condensed-RSA signature from the signatures of data corresponding to the remaining objects. Then, each database sends the data, the digest corresponding to the last object in phase 2, the signature of S-MHT and the Condensed-RSA signature to the DSP.

So, when a user receives the data, the digest, the signature of S-MHT and the Condensed-RSA signature for each database, it knows which objects are remaining objects whose upper bounds are greater than the smallest top-k value. Then, by using the Condensed-RSA signature the user can verify whether the data corresponding to remaining objects are forged or dropped.

Figure 4 shows the S-TPUT algorithm. Compared to algorithm in Figure 3, Steps 1 to 5 are same but the remaining parts include the use of Condensed-RSA. In step 6, we can see that it requests a set of only *data* corresponding to remaining objects instead of a *sequence* containing remaining objects.

By using S-TPUT, we can reduce the communication overhead between databases and DSP. But, since each record need a signature, the databases have more storage overhead. However, we assume that since there are big storages nowadays, the storage overhead is not a significant problem. In addition, since S-TPUT exploits the Condensed-RSA signature, the databases have more computation overhead than A-TPUT. But, the computation times overlap with disk I/O time at the databases, and S-TPUT needs only a small number of signatures of records when compared to all records to be sent. So, S-TPUT has much less computation time than ASB-tree which needs the signatures for all records [7]. In [7], one I/O operation time for random access is 15ms and the cost of one modular multiplication with 128Byte modulus for the Condensed-RSA signature is 100 μ s.

Algorithm S-TPUT
1. Request the local top-k objects to all databases;
2. Compute a threshold $T=t_1/m$ where t_1 is phase 1 bottom;
3. Request objects whose values $\geq T$ to all databases;
4. Compute phase 2 bottom t_2;
5. Prune objects whose upper bounds are less than t_2;
6. Request *data* corresponding to the remaining objects;
7. Each database compute its Condensed-RSA signature from the signatures corresponding to the remaining objects
8. Report the *data*, each digest, each signature of S-MHT, and each Condensed-RSA signature for each database to the user;

Figure 4. S-TPUT algorithm

Example 4: In phase 3 of A-TPUT, since $S = \{O_2,O_5,O_6,O_0,O_3,O_7\}$, the database 1 should send data up to position 7, $Digest_{1,7}$, and the signature of S-MHT. The database 2 has to send data up to position 8, $Digest_{2,8}$, and the signature of S-MHT. Finally the database 3 should transmit data up to position 6, $Digest_{3,6}$, and the signature of S-MHT.

However, in phase 3 of S-TPUT, the database 1 does not need to send $<O_1,0.28>$ since O_1 is not in S. Instead, it computes a Condensed-RSA signature $CS_1 = S_{O7} \cdot S_{O5} \cdot S_{O0} \cdot S_{O3}$ and sends O_7, O_5, O_0, and O_3 with the aggregate signature CS_1, the digest, and the signature of S-MHT. The database 2 does not need to send O_4 and O_1. It sends O_2 and O_6 with its Condensed-RSA signature CS_2. The database 3 sends O_6 and O_2 with its Condensed-RSA signature CS_3. So, S-TPUT sends 3 records less than A-TPUT in this example. Finally, when the user receives the data and the Condensed-RSA signatures, it multiplies the hashes of the data from each database corresponding to the remaining objects and it checks whether the product is equal to each Condensed-RSA signature.■

Theorem 2. *S-TPUT correctly authenticates the top-k objects.* Proof (sketch). By Theorem 1, a sequence in phase 2 satisfies correctness and the completeness since each database should send data whose values are greater or equal to the threshold. Suppose that there are x remaining objects in phase 3. An adversary succeeds in breaking Condensed-RSA if it produces a valid aggregated signature for the remaining objects which

passes verification. There are two cases. First, the adversary can forge the value of an object. Second, it can drop an object.

First, suppose that the adversary changes the value V_x to V_x' for the object O_x. However, since it does not know the data owner's private key, it cannot generate valid individual signature S_x' for the forged value V_x'. Hence, it cannot generate valid Condensed-RSA signature to pass the verification. Thus, $(s_{1,x})^e \neq \prod_{l=1}^{x} h(O_l|V_l') \pmod n$. Second, the adversary may drop an object. In that case, by Theorem 1, the user knows which objects are remaining objects in phase 3. So, if the adversary drops an object, it is detected by the user. Therefore, S-TPUT correctly authenticates the top-k objects. ■

V. OPTIMIZATION

In this section, we present an optimization technique for A-TPUT and S-TPUT. Optimization can be done between the databases and the DSP. We focus on minimizing the amount of data transmission.

A. Optimization between Databases and DSP

Even though A-TPUT and S-TPUT are efficient algorithms in that it reduces the communication cost by pruning away ineligible data items, it can be inefficient especially in the case where the threshold T is too small. In A-TPUT and S-TPUT, the threshold is set to $T = t_1/m$ where t_1 is the phase 1 bottom and m is the number of databases. If T is small, the databases should send large parts of their data to the DSP. This results in a large amount of data transmissions between databases and DSP which makes A-TPUT and S-TPUT inefficient.

In this section, we introduce an approach, called Improved TPUT (I-TPUT), to decrease the communication overhead of A-TPUT and S-TPUT by increasing the threshold T. We observe that data about an object are not sent from all databases in phase 1. This means that the local top-k objects are usually not exactly same in all databases. We can use this observation to replace t_1 by t_1' which is greater than t_1. Consequently, we can use $T' = t_1'/m$ instead of T. This increases the threshold in phase 1 since $T' > T$ and then decreases communication cost between databases and DSPs.

We calculate T' as follows. When the DSP receives the top-k objects from the databases, it computes the global objects and their aggregate values. In addition, for the object having the k-th largest value, it counts how many databases have sent the object. The counter for object O_k is denoted by C_k. For instance, when C_k is equal to j ($1 \leq j \leq m$), it means that the object O_k was received from j databases and $(m - j)$ databases did not send the object O_k in phase 1.

First, we assume that the values of the object O_k in the unreported databases are greater than or equal to T and make a new threshold as follows according to the assumption:

$$T' = (t_1 + (m - C_k) * T)/m$$

If the data values among databases are correlated [1], the assumption is true with high probability. For the case where the assumption is not true, we will recalculate T' later in I-TPUT algorithm.

Next, the DSP requests the databases to send data whose values are greater than or equal to T'. When the DSP receives data whose values are greater than or equal to T' in phase 2, it should check that the k-th largest value is greater than or equal to $m*T'$ to see whether the assumption we used to calculate T' is true or not:

1) If the k-th largest value is greater than or equal to $m*T'$, it means that the assumption is true (i.e., the objects which are not reported so far do not have values greater or equal to $m*T'$). Therefore, we can safely use T' ($< T$) as the threshold and do not need to receive additional data from databases.

2) On the other hand, if the k-th largest value is less than $m*T'$, it means that the assumption is false and we need to set a new threshold value $T^* = t_2/m$. Since t_2 is the k-th largest value in phase 2, it is greater than or equal to t_1. Therefore, $T \leq T^* \leq T'$ and this means that the threshold is still greater than or equal to the original TPUT. Now, the DSP requests the databases to send additional data whose values are greater than T^*.

Figure 5 shows the I-PUT algorithm in detail.

Algorithm I-TPUT
1. Request the local top-k objects to all databases;
2. Compute a threshold $T'=(t_1+(m-C_k)*T)/m$ where t_1 is phase 1 bottom;
3. Request objects whose values $\geq T'$ to all databases;
4. Compute phase 2 bottom t_2;
5. Check whether the smallest top-k value is greater than or equal to $m*T'$;
6. If so, go to the step 9;
7. Otherwise, Request objects whose values $\geq T^*$ ($=t_2/m$) to all databases;
8. Compute phase 2 bottom t_2;
9. Prune objects whose upper bounds are less than t_2;
10. Request *data* corresponding to the remaining objects;

Figure 5. I-TPUT algorithm

Example 6: Suppose that there are three databases and top-1 query is given. If (O_1, 0.6), (O_1, 0.6), (O_2, 0.7) are received from three databases in phase 1, then, $T = (0.6+0.6)/3 = 0.4$. But, in I-TPUT, $T' = (1.2 + (3-2)*0.4)/3 = 0.53$. So, by using T' instead of T, we can reduce the communication overhead. In phase 2, when the DSP receives data whose values are greater than or equal to 0.53, it should check whether the smallest top-k value is greater or equal to 1.59(=0.53*3). If so, the algorithm can terminate. Otherwise, the DSP should receive data whose values are greater or equal to T^* like the original TPUT. ■

Performance Analysis of I-TPUT. By using a threshold T' greater than T, I-TPUT reduces the communication overhead corresponding to part A in Figure 6. Since I-TPUT has higher threshold than A-TPUT, it sends less data than A-TPUT. The data which is not needed to be sent is shown part A. However, since I-TPUT may have more remaining objects than A-TPUT in phase 3, it may result in a higher communication overhead corresponding part B. If I-TPUT has more remaining objects than A-TPUT, its lowest position of the remaining objects is lower than A-TPUT. But, usually part A is much greater than part B. So, I-TPUT has much less communication overhead than A-TPUT.

The communication overhead of A-TPUT is $m * n * (1.0 - T) + \sum_{i=1}^{m} n * p_i * (T - LO_i)$ where LO_i is the smallest

[1] This kind of data correlation is common in real-world applications[10].

value for the remaining objects in list L_i and p_i is the probability that LO_i is less than T in list L_i. The communication overhead of I-TPUT is $m * n * (1.0 - T') + \sum_{i=1}^{m} n * p_i' * (T' - LO_i')$. Thus, the difference between A-TPUT and I-TPUT is $m * n * (T' - T) + \sum_{i=1}^{m} n * p_i * (T - LO_i) - \sum_{i=1}^{m} n * p_i' * (T' - LO_i')$. In correlated databases which have similar sets of top-k objects, usually p and p' are equal to 0. Thus, the difference is $m * n * (T' - T)$ where $T'>T$. Therefore, I-TPUT has much less communication overhead than the original A-TPUT.

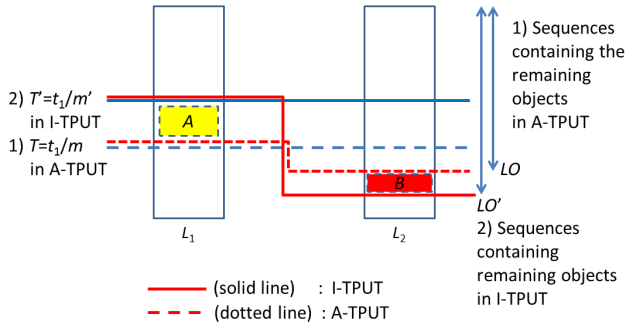


Figure 6. Performance Comparison of A-TPUT and I-TPUT

We note that I-TPUT can be used in S-TPUT as well as A-TPUT since I-TPUT is only involved in phase 1.

VI. EXPERIMENTS

A. Setup

We implemented the following algorithms in : A-TPUT, S-TPUT, AI-TPUT, and SI-TPUT. To better assess our algorithms we also implemented two existing algorithms:

- Naïve: The databases send all data to the DSP and the DSP forwards to the user all data received from the databases.
- TNRA: NRA based authenticated top-k aggregation algorithm proposed in [5]. TRA also provides authentication for top-k aggregation, but we compare our algorithms to only TNRA since [5] shows that TRA has worse performance than TNRA.

We tested them over correlated synthetic data sets.

Correlated sets are data sets in which the values of the data in the lists are correlated. In real-world applications, such correlations are common [10]. In our experiments, we generate two sets of correlated data. Inspired from [10,4], we use a correlation parameter α ($0 \le \alpha \le 1$). We use two kinds of synthetics as follows:

- Zipf law (CZ-Data): The first set of correlated databases was generated as follows. For the first list, we randomly select the position of data items. Let p_1 be the position of a data item in the first list, then for each list L_i ($2 \le i \le m$) we generate a random number r in the interval [1 .. $n*\alpha$] where n is the number of data items, and we insert the data item in the list at a position p such that its distance from p_1 is r. If p is occupied previously by another data item, we insert the data item at the free position closest to p. After setting the positions of all data items in all lists, we generate the values of the data items in each list in such a way that they follow the Zipf law. The Zipf law states that the value of an item in a ranked list is inversely proportional to its rank (position). Such distribution is commonly observed in many kinds of phenomena, e.g. the frequency of words in a corpus of natural language utterances.
- Uniform distribution (CU-Data): The second set of correlated databases was generated as follows. For the first list, we randomly generate a number for an object O_j. The values follow the uniform distribution. Let $p_{1,j}$ be the number. Then for each list L_i ($2 \le i \le m$) we generate a random number r in interval [$-\alpha$,… ,α] and we set $p_{i,j}$ to $p_{1,j} + r$. By controlling the value of α, we create databases with stronger or weaker correlations.

Our default settings for different experimental parameters are shown in Table II. In our tests, the number of databases, i,e, m, is a varying parameter. The default number of databases is 128. The default number of data items in each database is 10,000. Typically, users are interested in a small number of top answers, thus unless specified we set k=100. Like many previous approaches to top-k query processing [2], we use a scoring function that computes the sum of the local values. In addition, the default number of correlation parameter α is 0.01.

TABLE II. EXPERIMENTAL PARAMETERS

Parameter	Values
# of databases (m)	32,64,**128**,256,512
# of records (n)	2500,5000,**10000**,20000,40000
k in top-k	25,50,**100**,150,200
Correlation parameter α	0.001,0.005,**0.01**,0.05,0.1

To evaluate the performance of the algorithms, we measure the following metrics: communication overhead between databases and DSP, and response time taken to get the authenticated top-k results from the databases. Concerning the communication overhead it is important to notice that even though the number of remaining objects at phase 3 is small, if the position of the last top-k object is low, in A-TPUT databases would typically send a lot of data to the DSP. By measuring the number of records transmitted by A-TPUT and S-TPUT, we can verify that S-TPUT is more efficient than A-TPUT.

On the other hand, when the threshold is very small in A-TPUT or S-TPUT, each database should send a lot of data to the DSP. By measuring the communication overhead of AI-TPUT and SI-TPUT, we can verify that AI-TPUT and SI-TPUT are more efficient than A-TPUT and S-TPUT. AI-TPUT and SI-TPUT uses I-TPUT technique of section V.A.

Response time is a time that an algorithm executes for finding the top-k data items. TNRA requires several rounds. By contrast our algorithms only require three rounds. In distributed environments, a round trip time is much longer than a transmission time. So, TNRA has much longer response time than ours. We compare our algorithm to only TNRA since TRA has much larger communication overhead than TNRA [5].

For the experiments in Sections VI.B – VI.D, we use the synthetic data sets for the experiments since we can fine tune

characteristics of data. We omit the experiment for real data due to space constraint.

B. Communication Cost of S-TPUT

In this experiment, we compare the communication overhead of our algorithms with that of naïve approach to show the effect of the S-TPUT and the I-TPUT optimization (i.e., higher threshold values). In this subsection, we evaluate the efficiency of S-TPUT, whereas in next subsection we evaluate that of I-TPUT. The communication cost metric is the number of records transmitted from the databases to the DSP. The result with CZ-Data is shown in Figure 7 and the result with CU-Data is shown in Figure 8. From the results, we can see that the efficiency of our proposed algorithm is largely depends on the data distribution but our algorithm overwhelms the existing algorithms in most cases.

In Figure 7 we can see that with CZ-data, S-TPUT incurs about 100 times less communication overhead than A-TPUT. This is due to that, in phase-3, S-TPUT receives only the data corresponding to the set S of the remaining objects instead of the sequences containing all of the objects in S as with A-TPUT. When the number of remaining objects S in phase 3 becomes small, the advantage of S-TPUT becomes large. In addition, when the position of the last objects becomes low, the communication cost of A-TPUT becomes large.

The experiments in Figure 8 show that, unlike with CZ-Data, S-TPUT has a similar communication overhead with CU-Data compared to A-TPUT. With CZ-Data, S-TPUT has a small number of remaining objects in phase 3 compared to A-TPUT. But, with CU-Data, S-TPUT has a small threshold and there are a lot of data, whose values are greater than the threshold, to be sent in phase 2. On the other hand, with CZ-Data, even if S-TPUT has a small threshold, there are no a lot of data whose values are greater than the threshold in phase 2 since the values follow the Zipf law.

For example, let's assume that the threshold is 0.1. Since the values are between 0 and 1, with CZ-Data, a database sends only 10 records in phase 2 to DSP since the values follow the Zipf law. But, with CU-Data, the database should send about 90% of records since the values follow uniform distribution. Therefore, S-TPUT is efficient with CZ-Data, but it is not with CU-Data.

C. Communication Cost of I-TPUT

From the experiments in Figure 7, we can see that I-TPUT is not much efficient for CZ-data. This is due to that, in the Zipf law, the value of an item in a ranked list is inversely proportional to its rank. Since the value is between 0 and 1, the value in the first rank is 1, the value in the second rank is 0.5, and the value in the j-th rank is $1/j$. So, for example, when k=100, a database sends records whose values are greater than 0.01 in phase 1 by the Zipf law since it sends local top-100 records in phase 1. If the threshold of S-TPUT is greater than 0.01, SI-TPUT is not much efficient since even if SI-TPUT has a higher threshold than S-TPUT, it already sent records whose values are greater than 0.01 in phase 1.

In contrast, in Figure 8, with CU-Data, we can see that I-TPUT is much efficient compared to A-TPUT and S-TPUT. AI-TPUT and SI-TPUT incur about 40% less communication overhead than A-TPUT and S-TPUT. This is due to the fact that when the threshold is small in A-TPUT or S-TPUT, the databases have to send a lot of data to the DSP since the values are uniformly distributed. But, in AI-TPUT or SI-TPUT, by increasing the threshold using our I-TPUT algorithm, we can reduce the communication overhead compared to A-TPUT and S-TPUT.

The results in Figure 8(c) show that, when k is 25, A-TPUT and S-TPUT have more communication overhead than the other cases in which k is greater than 25. When the values are uniformly distributed, if k is too small, we cannot find the proper threshold. As k increases, we can get the better threshold.

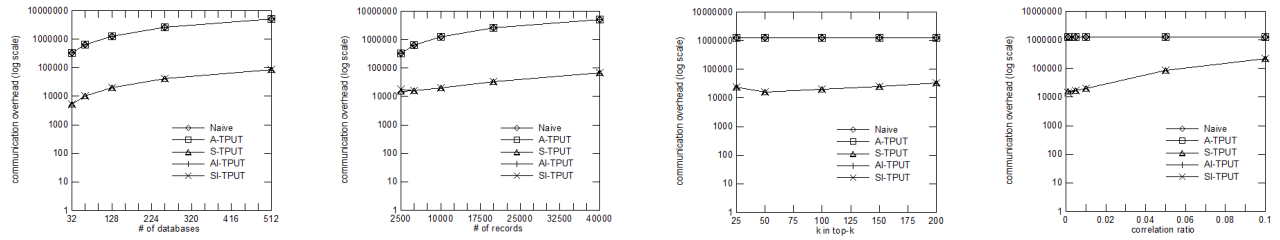
D. Comparing S-TPUT with TNRA

In this experiment, we compare our S-TPUT with the existing authenticated top-k aggregation algorithm, TNRA [5]. We measure the response time for the DSP to receive all data for top-k from databases. As we described in Section V, counting the number of transmitted records is not feasible to compare the response time since TNRA has unpredictable number of round trips and the round trip rime is much longer than the transmission time. But our approach only requires a fixed number of rounds. This feature significantly reduced actual response time since in distributed environments the round trip time is much higher than the packet transmission time. The round trip time is proportional to the distance between a database and a DSP. We assume that the round trip time is 10ms and the processing time is trivial. For TNRA, the database sends k (=100) records every round.

In Figures 9, we can see that, in all experimental instances, S-TPUT has a constant response time, whereas TNRA has a response time much greater than S-TPUT. As the parameters like the number of databases, k in top-k, the number of records, and correlation ratio α increase in TNRA, the response time increases. The results show that S-TPUT is the most suitable algorithm for a distributed environment. Thus, S-TPUT has much lower response time than TNRA.

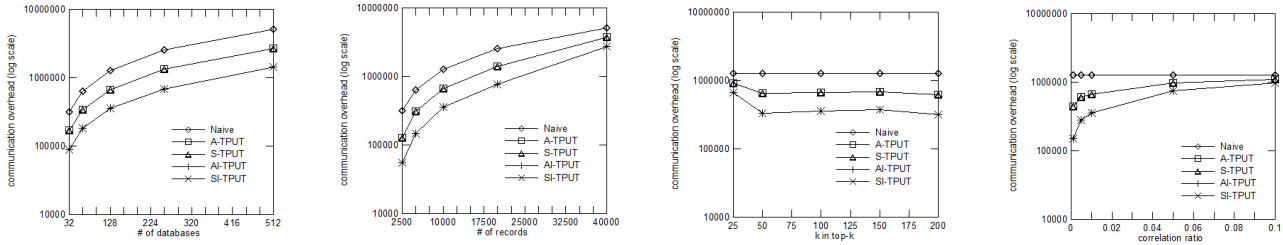
VII. Conclusions

In this paper, we present the work for authenticated top-k aggregation in distributed and outsourced databases. Our aim is to enable the users to detect whether the top-k results contain the correct results and to give efficient algorithms to compute the top-k results with less communication overhead and lower response time. Our algorithms are based on the Three Phase Uniform Threshold (TPUT) algorithm which gives top-k but does not have an authentication mechanism. To the best of our knowledge, ours is the first authentication mechanism based on TPUT which is efficient in distributed environments. First, we propose A-TPUT having authentication mechanism based on TPUT. Second, we suggest S-TPUT and I-TPUT to improve A-TPUT. In addition, we experimentally evaluate our techniques and demonstrate their robustness and practicality.



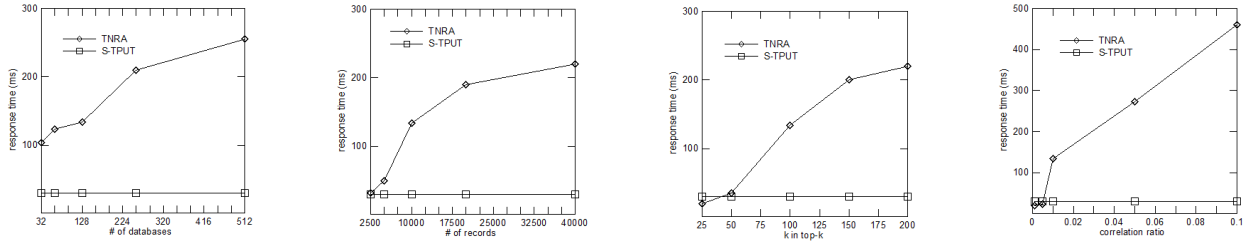
(a) communication cost by m (b) communication cost by n (c) communication cost by k (d) communication cost by α

Figure 7. Correlated Data following the Zipf law



(a) communication cost by m (b) communication cost by n (c) communication cost by k (d) communication cost by α

Figure 8. Correlated Data following Uniform distribution



(a) Response time by m (b) Response time by n (c) Response time by k (d) Response time by α

Figure 9. TNRA vs S-TPUT

ACKNOWLEDGMENT

The work reported in this paper has been partially supported by NSF under grants CNS-0964294 and CNS-1111512.

REFERENCES

[1] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal Aggregation Algorithms for Middleware. In ACM PODS, 2001

[2] Pei Cao and Zhe Wang. Efficient Top-K Query Calculation in Distributed Networks. In ACM PODC, 2004

[3] Nikos Mamoulis et al. Efficient Aggregation of Ranked Inputs. In IEEE ICDE, 2006

[4] Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. Best Position Algorithms for Top-k Queries. In VLDB, 2007

[5] HweeHwa Pang and Kyriakos Mouratidis. Authenticating the Query Results of Text Search Engines. In VLDB, 2008

[6] HweeHwa Pang et al. Verifying Completeness of Relational Query Results in Data Publishing. In ACM SIGMOD, 2005

[7] Feifei Li et al. Dynamic Authenticated Index Structures for Outsourced Databases. In ACM SIGMOD, 2006

[8] HweeHwa Pang and Kian-Lee Tan. Authenticating Query Results in Edge Computing. In IEEE ICDE, 2004

[9] Ronald Fagin. Combining Fuzzy Information from Multiple Systems. Journal of Computer and System Sciences, 1999

[10] Sebastial Michel, Peter Triantafillou, and Gerhard Weikum. KLEE: A Framework for Distributed Top-k Query Algorithms. In VLDB, 2005

[11] Suman Nath, Haifeng Yu, and Haowen Chan. Secure Outsourced Aggregation via One-way Chains. In ACM SIGMOD, 2009

[12] Rui Zhang et al. Verifiable Fine-Grained Top-k Queries in Tiered Sensor Networks. In IEEE INFOCOM, 2010

[13] Fei Chen and Alex X. Liu. SafeQ: Secure and Efficient Query Processing in Sensor Networks. In IEEE INFOCOM, 2010

[14] Einar Mykletun et al. Authentication and Integrity in Outsourced Databases. In NDSS, 2004

[15] http://ita.ee.lbl.gov/html/contrib/UCB.home-IP-HTTP.html

[16] Hailing Yu et.al. Efficient Processing of Distributed Top-k Queries. In DEXA, 2005

[17] Thomas Nuemann et.al. Distributed top-k aggregation queries at large, Distributed and Parrallel Databases, 2009