Computing and Informatics, Vol. 32, 2013, 549–571

A MODEL TO OVERCOME INTEGRITY CHALLENGES OF AN UNTRUSTED DSMS SERVER

Majid Ghayoori, Mostafa S. Haghjoo

Computer Engineering Department
Iran University of Science and Technology (IUST)
Narmak, Tehran, Iran
e-mail: {ghayoori, haghjoom}@iust.ac.ir

Communicated by Isabel Campo Plasencia

Abstract. Despite the fact that using the services of outsourced data stream servers has been welcomed extremely, still the problem of obtaining assurance about the received results from these untrusted servers in unsecure environment is one of the basic challenges. In this paper, we present a probabilistic model for auditing received results from an outsourced data stream server through unsecure communication channels. In our architecture, the server is considered as a black box and the auditing process is fulfilled by cooperation between the data stream owner and users. Our method imposes an ignorable overhead on the user and needs no change in the structure of the server. The probabilistic modeling of the system proves algorithms convergence and the experimental evaluations show very acceptable results.

Keywords: Data stream security, data stream integrity, data stream management system, data stream management system outsourcing, query assurance

1 INTRODUCTION

Although some academic and commercial Data Stream Management Systems (DSMS) have been developed [1, 2, 3, 4, 5] and their usage is growing rapidly [6], many enterprises are not able to start up and pay the expenditure of such systems. That is why they are interested in receiving the service of data stream processing from a third party server. The idea of using outsourced processing services has already been used in Database Management Systems (DBMS), and the concept of Database As the Service (DAS) has been considered as its solution [7]. Figure 1 shows the architecture of an outsourced DSMS. In this architecture, a Data Stream Owner (DSO) sends the data records as a stream to the server and the server applies registered users' queries on the data stream and sends the results back.

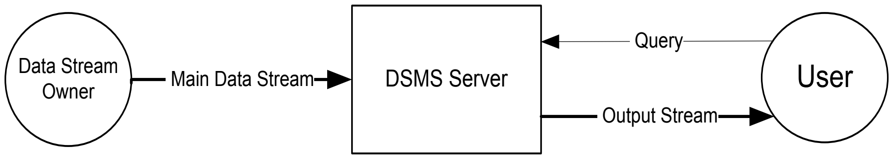


Fig. 1. System architecture of an outsourced DSMS

To exploit this idea, users must be assured about security of communication channels as well as about accuracy and honesty of the outsourced server. An attacker could arrange a Man-In-The-Middle (MITM) attack and inject some spurious records in results or prevent transmission of some result records to users. In addition, an external untrusted server could send inaccurate or incomplete results for different reasons, including software bugs, increasing its benefits, allocating less resources, misleading users, etc. To ensure integrity of results received from an untrusted server in an unsecure environment, users need mechanisms to audit correctness, freshness and completeness of results [8]. Correctness of results means keeping the main volume of data without making any changes in it. To keep the freshness we should make sure that received results are generated based on the latest stream records and not on previous ones. Completeness of results means that the user should receive the whole records of the results, not just a part of it, which is the most complicated task in the integrity auditing mechanisms.

In this paper, we present a model to audit integrity of results by users. In our model, integrity auditing is accomplished through cooperation between users and the data stream owner. The data stream owner injects some fake records to the real stream and sends it to the server. The users, who are aware of the fake records identification and generation procedure, apply their query on the fake data stream and compare the results with the received fake records. In our method, the accuracy of the system can be adjusted based on the user request.

The main contributions of this paper are as follows:

- Presenting an architecture for auditing integrity of outsourced DSMS server.
- Probabilistic modeling of the system.
- Formulating the accuracy of the auditing level based on received results.
- Implementation of the system and evaluation of the results.

This paper is organized as follows. Section 2 describes related works. Section 3 describes problem statement and preliminary assumptions. Section 4 presents our method. Section 5 focuses on the integrity auditing algorithms, and Section 6 experimentally evaluates our method. Section 7 proves the convergence and correctness of the algorithms, and finally, Section 8 concludes the paper.

2 RELATED WORKS

The integrity auditing methods can be categorized in two major categories, namely authenticated data structure (ADS) based method, and probabilistic methods.

So far, the most popular authenticated structure, used for integrity auditing, is Merkle Hash-Tree (MH-Tree) [9]. In [10] an extended model of MH-Tree, based on B+-Tree, is presented. In this model, record domains are divided into some ranges and records tree is built upon these ranges. This model is designed for executing continuous range queries and supports one-time queries as well. This model can audit correctness and completeness of results. In order to eliminate the tree refreshes, an algorithm to divide the data stream into some trees is presented in [8].

To exploit these two ADS-based methods, like every ADS-based method, the server must be restructured based on the selected ADS. This is not an acceptable precondition for many third party SDMS service providers because they serve their DSMS services for some customers at the same time. Thus, they could not restructure for any customer. In addition, all DSMS servers have their stream-processing engine, in which their staff are experienced and professional. They have designed their hardware infrastructure based on their stream processing engine. Obviously, replacing their engine by another one is in conflict with the spirit of outsourcing DSMS services. Our method considers the server as a black box and needs not any changes in it. Also, as we know, the server load shedding is one of the necessary techniques in DSMS engines when they encounter high input load [11]. Because these two methods strictly check the completeness of results, they alarm on absence of any results record. As a result, they could not tolerate server load shedding and take it as an attack. Our method can tolerate previously agreed load shedding.

Based on our studies, there is one research that uses probabilistic methods [12, 13]. This method is designed for analyzing the network data traffic, and is called Polynomial Identity Random Synopses (PIRS). In this method, the user continually calculates some synopses of the received data stream and compares them with expected ones for auditing the integrity. This method supports COUNT and SUM queries and has an acceptable computation and memory overhead. In PIRS, users must have access to data stream for calculating synopses. The users' access to main data stream is not a realistic presumption because it has a high overhead to communication channels when users have access to a limited low bandwidth communication channel. In our model, users need not have access to main data stream and they can audit received results by evaluating the results.

3 PROBLEM STATEMENT AND PRELIMINARY ASSUMPTIONS

In an outsourced DSMS, the users should be assured about integrity of the results received from the untrusted server in an unsecure environment. In order to confirm the integrity, the user should make sure about the points below:

Completeness: No record is deleted from the expected results.

Correctness: All the received records have originated from the data stream source and there are no spurious records in them.

Freshness: The received results from the server are prepared based on the latest received records from data streams.

In this research, we introduce an efficient method for auditing completeness in a setting consisting of the data stream owners, the server and the users. In this method, there is no need to make any modification in the structure and functionalities of the server.

The initial assumptions in this system are as follows:

- The channels between the data stream owner, the server and the users are not secure and Man-In-The-Middle attacks are probable. Therefore, attackers may modify some result records, inject some spurious records in results and/or drop some records from them.
- The server is not secure and trustable. The problems of the server are of two types:
  - The server is lazy or works less, i.e. it does not answer the user's query completely or answers with delay.
  - The server is malicious, i.e. it intentionally deletes some records from the results or changes them.
- The users do not have a direct connection with the data stream owner and their connection is only possible trough the server.
- The users and the data stream owner can transfer limited amount of data at the beginning of the system start up.
- The processing power of the users is limited. We should not possibly impose any kind of extra overhead to them.
- The records are transferred between the data stream owner, the server, and the users as encrypted.
- A data stream received from a data stream owner p is an infinite series of indexed records as $S_p = \{r_0, r_1, \ldots\}$. Records arrive one at a time sequentially. A sliding window of size n contains n records as $(r_{\tau-n+1}, \ldots, r_\tau)$. This model is referred to as record-based sliding windows [14].

- The user k registers query Q on the data stream p (Q_p^k) as follows:
  *SELECT **
  FROM S_p
  WHERE Condition$(A_1, A_2, \ldots, A_m)$
  WINDOW SIZE n *SLIDE EVERY* t *($t >= n$).*

  In these queries, $Condition(A_1, A_2, \ldots, A_m)$, is any combination of conditional phrases defined on attributes of S_p which are combined with logical operators (*AND*, *OR*, *NOT*).
- The result of applying query Q_p^k on S_p is an infinite series of records (R_p^k):

$$R_p^k = \{r_i | (r_i \in S_p) \wedge (\forall k \in N, i \in [k\tau - n + 1, k\tau]) \wedge (r_i \textbf{satisfies } Q_p^k)\} \quad (1)$$

Based on the above assumption, the main subject in this study is developing a new method for auditing the integrity of received results from an untrusted DSMS server in an unsecure environment without making any change in the server.

4 OUR METHOD

In our method, the Data Stream Owner (DSO) injects some fake records into the real stream records. The users can also generate such records in parallel with the DSO. The DSO sends records to the server in encrypted form, so the server is unable to distinguish fake records from real ones. The server executes the user queries on all records (real and fake ones) and sends the results to the users. Because the users are aware of fake records, they are able to separate fake results from the real ones and judge the integrity of the results.

4.1 Our System Architecture

Our system architecture is shown in Figure 2. To keep records confidential, all records are encrypted before being sent to the server (the server executes queries on encrypted data). Some methods for executing queries on encrypted data are presented in [15, 16, 17]. The methods of encryption and of executing queries on encrypted data are irrelevant to our research. According to the figure, there is a common part in the data stream owner and the users for the fake data stream generation. The server and users in the format of Service Level Agreement (SLA) agree on the maximum Load Shedding Ratio (LSR). In addition, the data stream owner and users agree on the Fake Records Ratio (FRR).

Data stream owner adds a header to all data stream records as follows:

$$\begin{aligned} \forall r \textit{ Real Data Stream} &: \; H_r = \textit{Hash}(a_1 | a_2 | \ldots | a_n) \\ \forall r' \textit{ FakeData Stream} &: \; H_{r'} = \textit{Hash}(a_1 | a_2 | \ldots | a_n) + 1. \end{aligned} \quad (2)$$

The users use the above header to audit correctness of the results and to separate real records from fake ones [18].

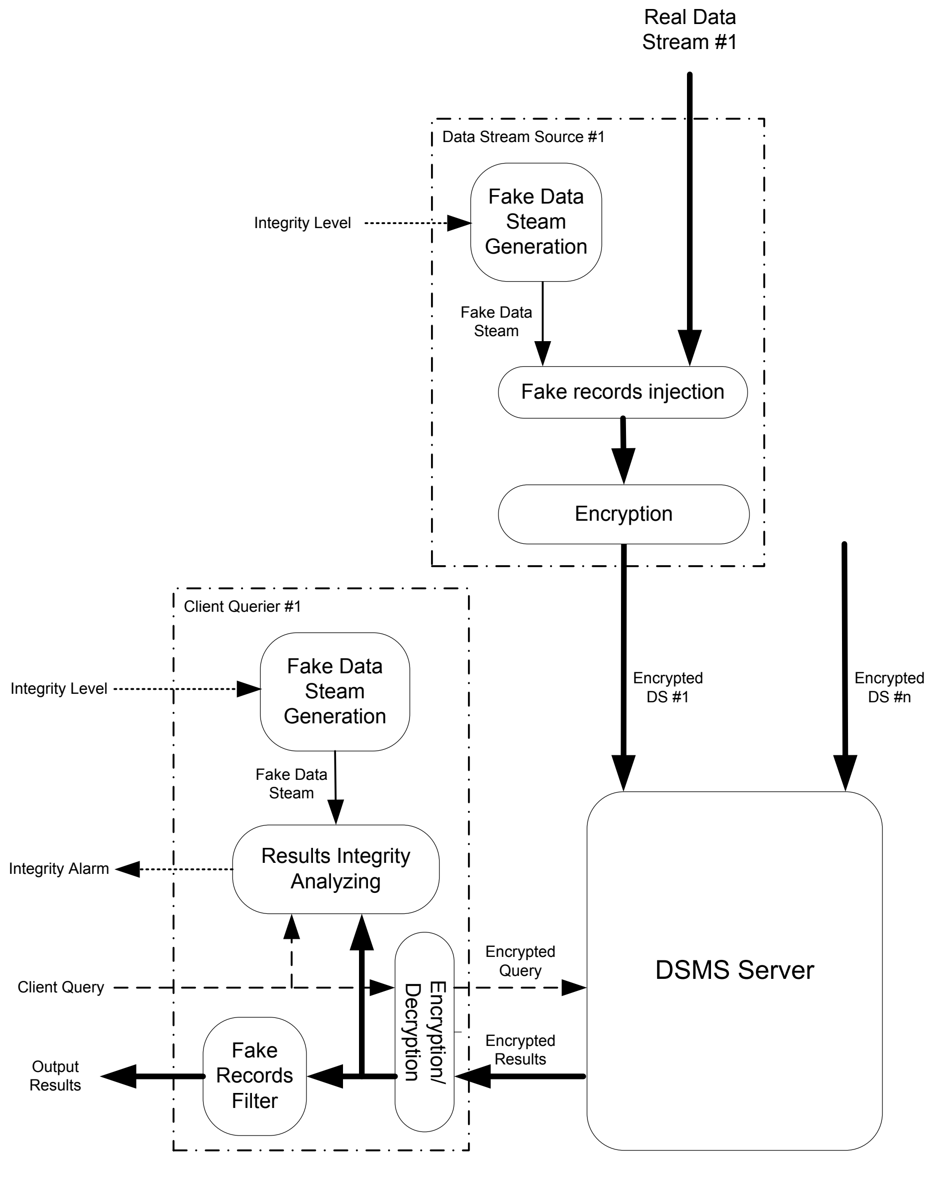


Fig. 2. System architecture

To implement the above architecture, we have to overcome the following challenges:

1. Synchronization of the data stream owner and users in the fake data stream generation.
2. Generating the fake records so that there should be some fake records in each user's query.
3. Synchronization of applying queries by the server and users.

In this section, we present our algorithms to the first two challenges and the last challenge is described in the next chapter.

4.2 Fake Data Stream Generation

4.2.1 Fake Records Position

To achieve synchronization between the data stream owner and the users, the fake record positions must be determined by a deterministic function. In this research, we design an algorithm based on a pseudo-random-number-generation function for determining the fake record positions. In our algorithm, the initial seed and the selected pseudo-random number generation function should be transferred as symmetric keys between the data stream owner and the users through a secure channel.

The data stream owner determines the fake record positions by the following key steps:

1. The data stream owner adds a timestamp (ts) to all records (real and fake ones). This field is a small integer and has no significant effect on the record size.
2. The data stream owner classifies the data stream into a sequence of rather large consecutive windows named the General Window (GW). The data stream owner sets the timestamp of the first record in any general window to zero.
3. The length of general windows (L_{GW}) is much larger than the users' Query Windows (QW) ($L_{GW} >> n$), so a single general window includes a considerable number of query windows. Due to the large length of the general windows, we assume that in each general window, at least some records are sent to the users as results of their queries.

4.2.2 Fake Records Generation

The fake records must be generated as a fake data stream by the data stream owner, and then merged with the real data stream. Additionally, the users have to produce the same fake data stream synchronized with the data stream owner. Therefore, to produce the fake data stream, we must address the following problems:

1. Coordination of the data stream owner and users in the fake data stream generation.

Method 1: A simple method to produce the same sequence of the fake records by the data stream owner and users is to pre-generate and save them. This method imposes a rather low computing overhead on the data stream owner and users, but the system suffers a considerable storage, and more importantly, data transmission overhead through the limited secure channel between the data stream owner and users.

Method 2: The data stream owner and users may use a deterministic function to generate the fake data stream. The initial value of this function is transferred between the data stream owner and the users through a secure channel. This method has less memory overhead, but it does have some computing overhead. This overhead strongly depends on the fake records ratio.

2. Maximum coverage of the fake records in the query results.

The more fake records in the users' query results, the more reliable the integrity auditing. To generate fake records, we use the simulation and production methods. These methods are based on the probability distribution function, which is obtained from statistical analysis of the real records.

In the literature, some distribution functions are proposed for different applications. For example, the Zipfian distribution function is used for simulating the size of cities, the length of English words, or the frequency of a word's repetition in a text; measurement errors are usually estimated by a Gaussian function, and the period of system failure is estimated by a Poisson or Negative Exponential function [19]; etc.

In [20], a method for record generation is proposed based on the domain values distribution histogram. In [21] and [22], which simulate web page access streams and telecommunication systems, respectively, this problem is well-studied, and a distribution function is presented. The records generated by these systems are simulated based on a self-similar model. Another record generator algorithm is presented in [19]. In this study, a multidimensional data stream, based on the probability distribution of domains, is proposed. A hierarchical (tree) structure is built and records are generated based on that structure.

In our system, we do not present a new method for fake records generation; rather, we use existing methods that are presented by other researchers. Our fake record generator depends on the specifications of the system and its environment as well as on the statistics of the real records. Each of the functions introduced above may be used in one or more application areas.

5 INTEGRITY AUDITING OF RESULTS

Users are aware of the fake records generation algorithm; thus, they can determine the fake records that are expected to appear in the results by performing their query on the generated fake records. Now, they are able to verify the integrity of the results by comparing the expected records with the records they actually received.

To implement the above process, the users must determine the exact position of server query window (Figure 3) and apply their queries to the fake data stream synchronously.

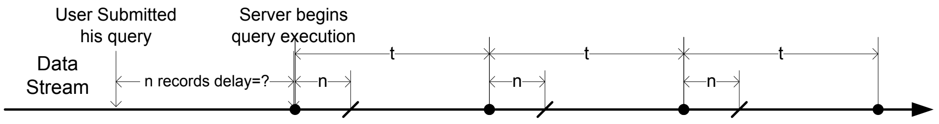


Fig. 3. Delay of server before starting query execution

In the rest of this section, first we present the algorithm of the estimation of server query window position and then the complete the algorithm for integrity control of results.

5.1 Estimating Position of Server Query Window by Users

Users must execute their queries on the fake data stream and compare the results with received records. In continuous queries, the data stream is divided into consecutive sliding query windows of t records, and the query is applied only to the n first records ($n \leq t$) of each query window (Figure 3). In order to compare the fake records with the expected ones, the users' query window must be synchronous with the server's query window. The users are not aware of the server's lag time in start of query execution, so, they have to estimate the position of the server's query window using the received records.

In this paper, we present a repetitive algorithm for the estimation of the server query window position. The abbreviations used in our algorithm are defined in Table 1.

Abbreviation	Description
GW_c	current general window
QW_i	i^{th} query window in GW_c
ts_{es}	minimum received ts in QW_0
ts_{ee}	maximum received ts in QW_0
ts_{cs}	calculated ts of QW_0 beginning
ts_{ce}	calculated timestamp of QW_0 end
ts_c	timestamp of current received record
ts_{c-1}	timestamp of previous received record
ts_L	last received ts mapped to QW_0
C_{er}	number of errors

Table 1. Abbreviations used in algorithm of server query window estimation

In our algorithm, the position of the first query window (QW_0) is estimated, and the positions of other windows are computed based on QW_0. QW_0 is the first query window that starts and ends in the current general window (Figure 4).

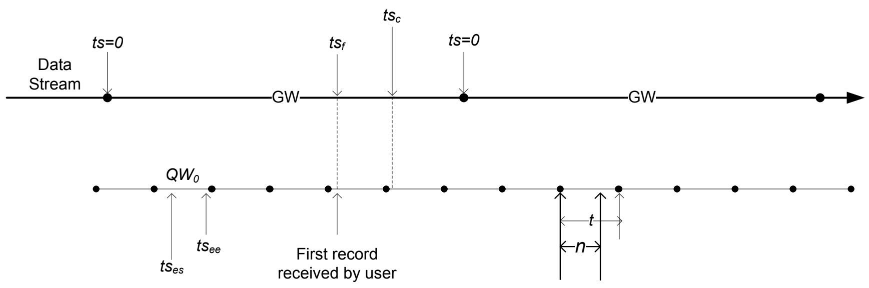


Fig. 4. Position of query windows in general windows

We know that all received records are in the range of query windows; therefore, after mapping their timestamp to the first query window, we examine the maximum and the minimum timestamps as start and end of the first query window, respectively. If the estimated query, window size is smaller than the real one, then we expand it from both sides.

Here is the algorithm:

```
Algorithm EstimateServerQueryWindow(Intput Current_ts){
  $ts_L$ = (Current_ts mod t) + t  //Map Current_ts to first QW
  If (Current_ts < Previous_ts)   // general window is changed
     $ts_{es}$ = $ts_{es}$ - ($L_{GW}$ mod t); $ts_{ee}$ = $ts_{ee}$ - ($L_{GW}$ mod t);
     If ($ts_{es}$ < t)  $ts_{es}$ += t; $ts_{ee}$ += t;
  Else
     If (($ts_{ee}$-$ts_L$>n | $ts_L$-$ts_{es}$>n)  & !($ts_{es}$==0 & $ts_{ee}$==0))
         // some records are deleted or freshness error
         $C_{er}$++;
         If ($C_{er}$ > ErrorThreshold)
            Alarm("Freshness Error");
            $ts_{ee}$ = $ts_L$; $ts_{es}$ = $ts_L$; $C_{er}$ = 0;
         End If
     Else
         $C_{er}$ = 0;
         $ts_{es}$ = Min($ts_L$,$ts_{es}$); $ts_{ee}$ = Max($ts_L$,$ts_{ee}$);
     End If
   End If
   $ts_{cs}$ = $ts_{es}$-((n-($ts_{ee}$-$ts_{es}$))/2); $ts_{ce}$ = $ts_{cs}$+n-1;
   Previous_ts = Current_ts;
End of Algorithm EstimateServerQueryWindow
```

Note 1. Based on our assumptions, we choose general windows large enough so that some result records appear in them for all the users' queries. As a result, the distance between two result records is certainly less than L_{GW}. Therefore, as shown in the algorithm, if $ts_c < ts_{c-1}$, a new general window is started and we should determine the position of QW_0 in the new general window.

Note 2. The above algorithm alarms successively if events such as temporary disconnection of the network occurs. In this case, some general windows pass without sending results, and the users must restart their program manually. To skip the manual restarting, we add an automatic restart mechanism to our algorithm based on an error threshold. This mechanism also guarantees freshness of the results because if the server sends old records, it will change the query windows sequence, and the integrity violation would be captured.

5.2 Integrity-Auditing Algorithm

Figure 5 shows an overall view of the user integrity-auditing algorithm. As seen in the figure, by receiving each result record, the position of server query window is estimated. In addition, the user's query is applied on the user generated fake records and the results are sent to Check Integrity. Simultaneously, the fake records in results are selected, and are sent to Check Integrity. To compare the received fake records with the expected ones, we must execute the EXCEPT operation on the generated and received fake records. To remove this overhead, it is proved in [20] that the COUNT of the two sets can be compared instead of comparing them record by record. The comparison is done based on the requested Integrity Level (*IL*). Here, a trigger is generated by Generate Control Trigger to start integrity evaluation of the current received record set.

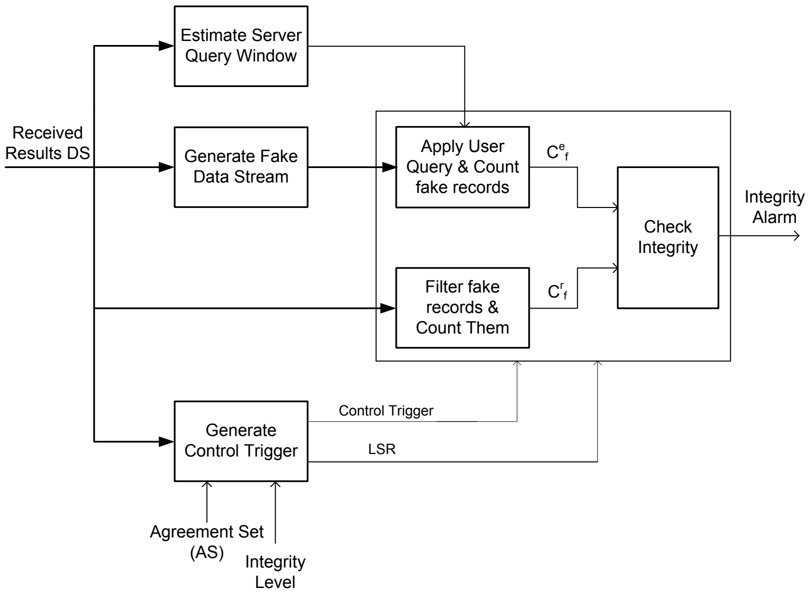


Fig. 5. User integrity audit algorithm schema

The integrity audit algorithm is shown below:

```
Algorithm EstimateServerQueryWindow(Intput Current_ts){
  $ts_L$ = (Current_ts mod t) + t  //Map Current_ts to first QW
  If (Current_ts < Previous_ts)   // general window is changed
     $ts_{es}$ = $ts_{es}$ - ($L_{GW}$ mod t); $ts_{ee}$ = $ts_{ee}$ - ($L_{GW}$ mod t);
     If ($ts_{es}$ < t)  $ts_{es}$ += t; $ts_{ee}$ += t;
  Else
     If (($ts_{ee}$-$ts_L$>n | $ts_L$-$ts_{es}$>n)  & !($ts_{es}$==0 & $ts_{ee}$==0))
         // some records are deleted or freshness error
         $C_{er}$++;
         If ($C_{er}$ > ErrorThreshold)
            Alarm("Freshness Error");
            $ts_{ee}$ = $ts_L$; $ts_{es}$ = $ts_L$; $C_{er}$ = 0;
         End If
     Else
         $C_{er}$ = 0;
         $ts_{es}$ = Min($ts_L$,$ts_{es}$); $ts_{ee}$ = Max($ts_L$,$ts_{ee}$);
     End If
   End If
   $ts_{cs}$ = $ts_{es}$-((n-($ts_{ee}$-$ts_{es}$))/2); $ts_{ce}$ = $ts_{cs}$+n-1;
   Previous_ts = Current_ts;
End of Algorithm EstimateServerQueryWindow
```

As shown in the algorithm, an endless loop compares the number of received fake records with the number of expected ones and detects attacks. Also, in this loop, we repeatedly estimate the server query window position and audit freshness of results.

5.3 Considering Load Shedding

Under our preliminary assumptions, the data stream owner, the users and the server have an agreement on the server's maximum load shedding schedule. As mentioned before, to audit the correctness of the results, the users just need to compare the number of received fake records with the number of expected ones. For considering load shedding; let C_{rf} be the number of received fake records and C_{ef} be the number of expected fake records, then, if $C_{ef} > (1-LSR)\times C_{rf}$, some of the expected records are not received from the server.

5.4 Defense Against Man-in-the-Middle Attacks

Network connections can be attacked in various ways. A general type of these attacks is "Man-In–The-Middle Attack". The idea behind this attack is to get in between the sender and the recipient, in order to access the traffic, modify it and then forward it to the recipient. Some different variants of this kind of attack exist, but a general definition of a MITM attack may be described as a "Computer security breach in which a malicious user intercepts – and possibly alters – data traveling along a network" [23].

Although the server is not trusted in outsourcing DSMS services, the integrity attacks could be forced in a form of MITM attacks. Here, the attacker may intercept the stream results, alter some result records, insert some spurious records into the results stream, and drop some records from results. Our system can detect all of these attacks as described below.

The attacker could not intercept the stream results because user's query and data stream are encrypted using an encryption key. The user could catch the alteration of records by record header. Because record header (H_r) is generated based on the one-way hash of all records attributes including timestamp, and the selected one-way hash function is hidden from attackers, any change in attributes causes the change of H_r and thus could be caught by the user. In the same way, the attacker could not inject a spurious record in stream results because he/she could not generate the H_r of it. In addition, the attacker could not reply previous results because all records have timestamps. We discussed the detection of replay attacks that affects the results freshness in section 5.1. Finally, the user could catch the result records dropping by detection of fake records absence in the results that the data stream owner previously injected them into real data stream.

5.5 Integrity Level

In our model, we define the integrity level based on the accuracy of the system in detecting the integrity attacks. For this purpose, we first present a probabilistic model to estimate the integrity auditing accuracy of our method and then define an algorithm to achieve the requested accuracy.

5.5.1 A Probabilistic Model for Completeness Control

In this section, we formulate the escape probability of the server or attackers who plan to delete some result records, i.e. the probability that our system cannot catch completeness attacks.

Let the average count of result records in a predefined time range be denoted by N and fake records ratio be FRR. Because the fake records are distributed uniformly in the data stream, the average count of the real records is $N \times (1 - FRR)$. The probability of the server or an adversary deleting only real records from the results (not discoverable by the user) is:

$$p = \prod_{i=0}^{N\times(1-FRR)-1} \frac{N \times (1 - FRR) - i}{N - i}. \tag{3}$$

In the above product, all terms are less than 1, and the largest term is equal to $\frac{N\times(1-FRR)}{N}$. Therefore,

$$p < (1 - FRR)^{N\times(1-FRR)}. \tag{4}$$

Figure 6 depicts the changes of p based on N for some $FRRs$. As seen in the figure, for all $FRRs$, after receiving 1 500 records, the probability of successful attacks against correctness is approximately zero.

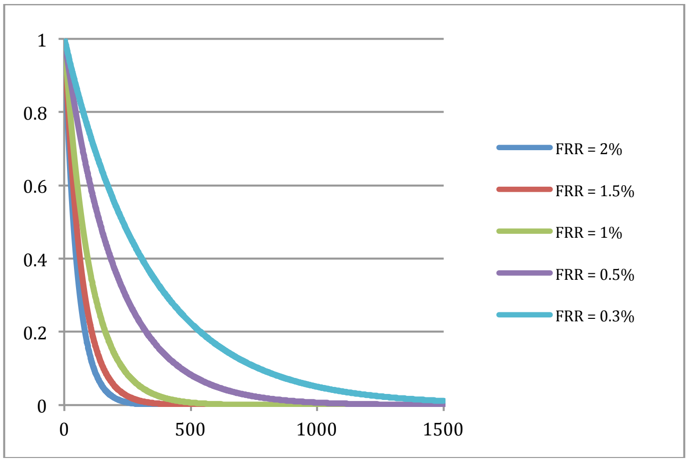


Fig. 6. Probability of successful attacks based on the result records count

Note 3. For load shedding, it is enough to replace *FRR* with *FRR-LSR* in the above formula. Then,

$$p < (1 - (FRR - LSR))^{N\times(1-(FRR-LSR))}. \tag{5}$$

5.5.2 Determination of Integrity Level

In our model, the integrity level is estimated by the accuracy of detecting the integrity attacks. Let α be the expected accuracy. Therefore, we have:

$$1 - a < (1 - FRR)^{N\times(1-FRR)}. \tag{6}$$

Because the *FRR* is agreed on by the data stream owner and users, to achieve the accuracy of α, it is enough to select N so that the above formula could be observed. Then, we have

$$N > \frac{Ln(1-a)}{(1-FRR)\times Ln(1-FRR)}. \tag{7}$$

For example, to achieve the accuracy of 0.99 with $FRR = 2\,\%$, it is enough to check the results for any 233 received records. We named every N result records as a checking window. The "Generate Audit Trigger" section of our algorithm shown in Figure 5 computes the checking window size, that is the minimum number of result records (N) required to achieve the expected accuracy based on the fake records ratio (FRR) and expected integrity level (α) and produces a "Control Trigger" after complete receipt of every checking window.

6 EXPERIMENTAL RESULTS

In order to simulate the data stream owner, the server and the user, we used three computers with a Pentium IV 2.0 GHz processor (2 Gigabytes main memories). These three machines were connected through a local network (100 MBps). We used typical query processor as the kernel of the server. We implemented all the algorithms in this experiment in Java language using JDK/JRE1, 6.

We used a data stream generated based on monitoring web page visits, related to the World Cup 1998 in the days 40 and 41 which approximately contains 20 000 000 records [24].

6.1 The Attack Model

6.1.1 The Completeness Attacks

In the completeness attacks, the server and/or the adversary does not send some results to the users. These attacks can be categorized into two classes, the deletion of some results (deletion attacks) and excessive load shedding.

In our experiments, the server or adversary deletes β per cent of the results. If the server deletes result records with the same probability, the deleted records are uniformly distributed in the results. We define $f(x)$ as the deletion decision function:

$$\forall r \in \mathit{Results\ Strem}, x = \mathit{Random}(r), \quad 0 \le x \le 1, \quad f(x) = \begin{cases} \mathit{True}, & x < \frac{\beta}{100} \\ \mathit{False}, & x \ge \frac{\beta}{100} \end{cases}. \tag{8}$$

Load shedding is simulated similarly. Here, γ is the LSR in per cent and $g(x)$ is the deletion decision function:

$$\forall r' \in \mathit{Input\ Stream}, x = \mathit{Random}(r'), \quad 0 \le x \le 1, \quad g(x) = \begin{cases} \mathit{True}, & x < \frac{\gamma}{100} \\ \mathit{False}, & x \ge \frac{\gamma}{100} \end{cases}. \tag{9}$$

6.1.2 Freshness Attacks

In freshness attacks, the server and/or adversary sends old results (instead of fresh ones) to the users. To model these attacks, we define a local data set S that saves κ records. The server or adversary uses this data set to save and resend old results to the users.

6.1.3 Deletion Detection

Users audit the results in consecutive sliding windows. When the server or attackers delete records, the system produces an alarm for the corresponding window. However, the system can only detect the deletion of fake records. Therefore, deletions are not detectable until a window with a deleted fake record arrives. This means that we always detect the records deletion, but there may be a delay. Our experimental results below show that the delay does not have a significant impact.

To evaluate our method, we defined two measures, the "Detection Delay" and the "Attack Detection Ratio", as follows:

Definition 1. The average number of records between record deletion and its detection is called the Detection Delay (DD). Formally,

$$\begin{aligned} r_k &: \textit{Deleted record} \\ r_l &: \textit{First deleted fake record after } r_k \\ n &= \textit{Number of deleted fake records} \\ DD &= \frac{1}{n}\sum(l-k). \end{aligned} \tag{10}$$

Obviously, a lower DD means the system detects the deletion attacks faster.

Definition 2. The ratio of "number of alarms" over "number of expected alarms" in per cent is called the Attack Detection Ratio (ADR). This measure shows the success of the system in attack detection. Formally,

$$ADR = \frac{Count(alarms)}{Count(expected\ alarm)} * 100 \tag{11}$$

Obviously, any $ADR > 0$ means that the attack is detected, and a higher ADR means that the system has better performance in the attack detection process.

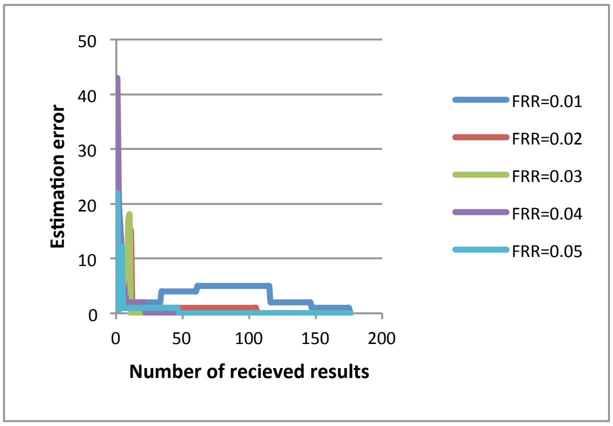
6.2 Query Window Location

The users must know the server query window position to apply their queries synchronously to the fake data stream. They estimate it by the EstimateServerQueryWindow algorithm. This estimation is a key step in detecting completeness attacks. Our experiments show that estimation error approaches zero after receiving a finite number of result records. We measured this error as the number of records between the estimated and the real position of the query window. Figure 7 depicts the effect of fake records ratio as well as the deletion of results (a completeness attack) on the error of query window position estimation.

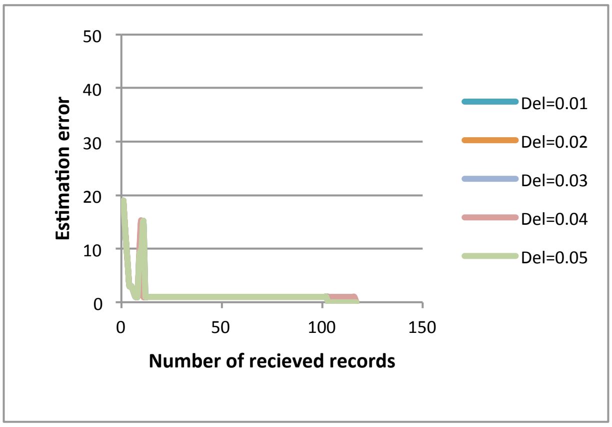
The results show that the two factors, result deletion and fake records ratio, have no significant effect on the error of server query window estimation, and the estimation error approaches zero after receiving a small number of records (60–100 records).

6.3 Completeness Attacks Detection

Figure 8 shows the detection delay based on fake records ratio. As shown in the figure, our method detects attacks quickly. The maximum delay of detection is not



a) No deletion attack



b) Results deletion

Fig. 7. Server query window estimation error

high and, as fake records ratio increases, detection delay decreases. For example, for $FRR = 3\,\%$, the detection delay lower than 1 000 means the attacks are detected at least after receiving 1000 records which is a small number of records in data stream processing.

Figure 9 shows the attack detection ratio based on deletion attack for different FRRs. As shown in Figure 9 a), all curves show a positive value for values between 0 and 2 of the deletion attack ratio (even for $FRR = 1.0\,\%$). This means that all attacks are detected by the system even for small deletion attack ratios (higher FRR results in more attack detections). Additionally, for all FRRs, the attack detection ratio approaches 100, which means deletion attacks are detected in time. Figure 9 b)

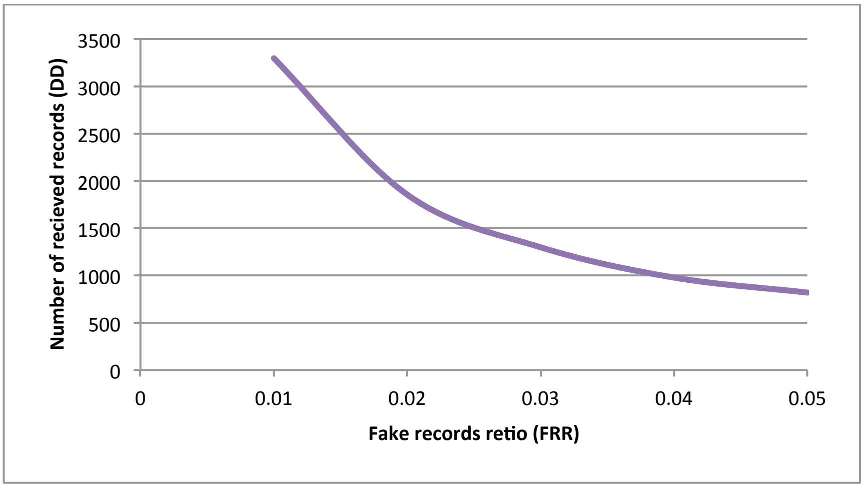


Fig. 8. Detection delay based on deletion attacks for different FRRs

shows that the behavior of the system under excessive load shedding is similar to that of under deletion attacks. The excessive load shedding detection has a better initial point on 0 and 2.

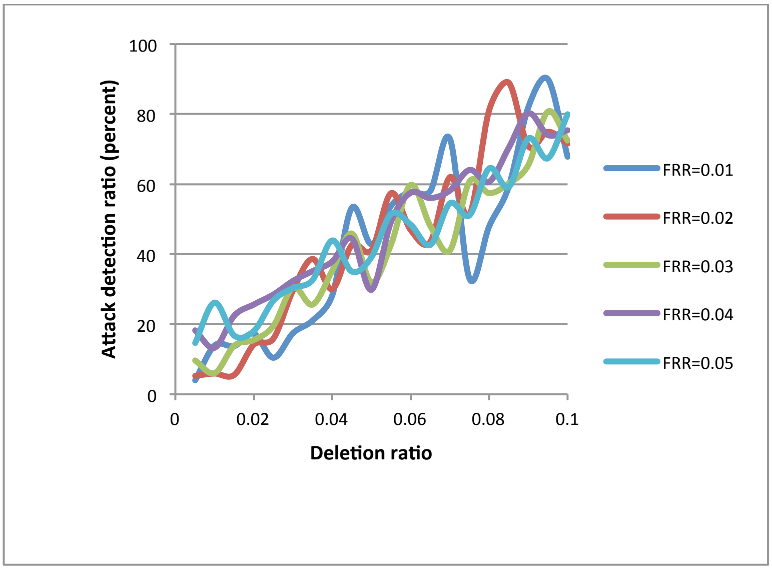
Figure 10 shows the attack detection ratio based on the number of received records (checking window size) for different fake records ratios. As shown in the figure, by increasing the checking window size, the attack detection ratio increases and approaches to 100. More fake records cause higher attack detection ratio for a constant checking window size. The results show a good behavior of our system in deletion attacks detection. Our system detects all deletion attacks even for small checking window sizes and the accuracy of detection is decreased by increasing the checking window size.

6.4 Freshness Attacks Detection

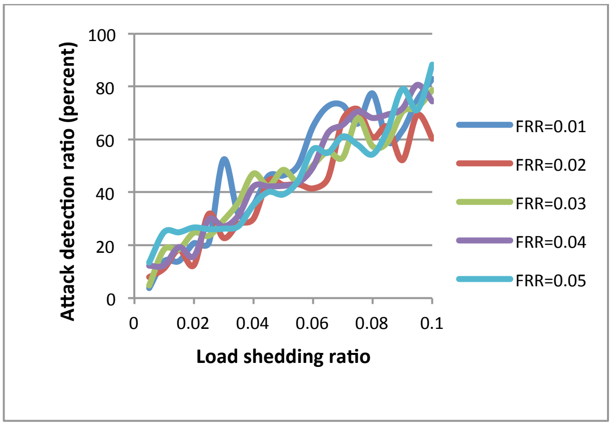
Our experiments on freshness attack detection showed that the system detects all freshness attacks with any κ. It is not an unexpected result because EstimateServer-QueryWindow algorithm uses record timestamps to check records sequence and any record with wrong timestamp can be detected by this algorithm.

7 SECURITY STATEMENT

In this section, we prove that our method for integrity auditing of outsourced DSMSs provides acceptable integrity assurance. Recalling that, the integrity assurance includes the correctness, completeness, and freshness.



a) Results deletion



b) Excessive load shedding

Fig. 9. Attack detection ratio based on deletion attack for different FRRs

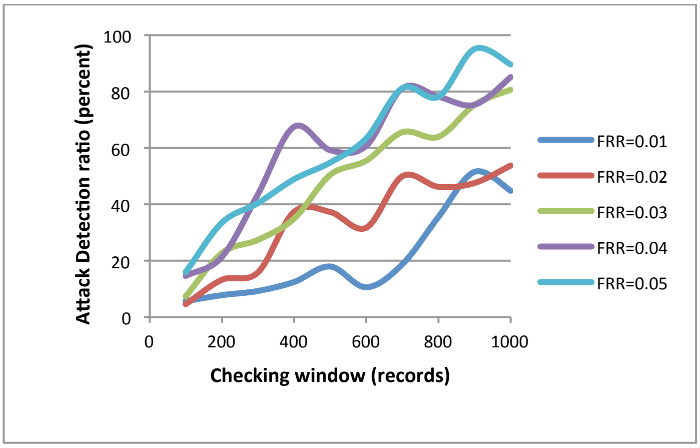


Fig. 10. Attack detection ratio based on checking window size for different FRRs

Correctness of results is evaluated by a header that is created by the data stream owner by applying a hash function to the contents of the records. Records are transferred between the data stream owner, the server and the users in encrypted forms. To add a spurious record, an attacker must guess the record header and the encryption key. Thus, the security of correctness audit algorithm depends on the security of the hash function and of the encryption method. This topic is out of the scope of this paper.

Below, we prove the correctness of completeness:

Theorem 1. The error of EstimateServerQueryWindow in estimating position of server query window approaches zero as the number of result records approaches infinity.

Proof. The positions of the query results are independent from the position of the server query window in the data stream. Let the query window size be n. The probability of a record appearing in any position of the query window is $1/n$. After receiving N result records, the appearance probability of at least one record at position k of any query window is:

$$p = \sum_{i=1}^{N} \left(\frac{n-1}{n}\right)^{i} \times \frac{1}{n} = \frac{1}{n} \sum_{i=1}^{N} \left(\frac{n-1}{n}\right)^{i}. \tag{12}$$

As $\frac{n-1}{n} < 1$, we have:

$$\lim_{N\to\infty} \sum_{i=1}^{N} \left(\frac{n-1}{n}\right)^{i} = \frac{1}{1-\frac{n-1}{n}} = n. \tag{13}$$

Therefore,

$$\lim_{N\to\infty} p = \frac{1}{n} \times n = 1. \tag{14}$$

Thus, as the number of result records increases, the probability of locating at least one record at any position of query window approaches 1. We have shown already that the EstimateServerQueryWindow algorithm estimates the position of the server query window accurately after receiving at least one result record at the beginning and end of the query window. As a result, as the number of result records increases, the probability of error in EstimateServerQueryWindow approaches zero. □

Theorem 2. The error probability of the completeness auditing algorithm approaches zero as the number of result records approaches infinity.

Proof. Let N be the number of the result records. If the server deletes only real records from the results, users cannot discover this deletion. Based on escape probabilistic model in Section 5.5.1, the probability of this situation is

$$p = \prod_{i=0}^{N\times(1-FRR)-1} \frac{N\times(1-FRR)-i}{N-i}. \tag{15}$$

In this product, all terms are less than 1 and the largest term is $\frac{\mathrm{N\times(1-FRR)}}{\mathrm{N}}$, so

$$p < (1-FRR)^{N\times(1-FRR)}. \tag{16}$$

Because $(1-FRR) < 1$,

$$\lim_{N\to\infty} p = 0. \tag{17}$$

Therefore, the error probability of the completeness auditing algorithm approaches zero as the number of result records approaches infinity. □

8 CONCLUSIONS

In this paper, we presented a solution for the integrity auditing of query results received from an outsourced DSMS server. First, we presented an architecture based on our idea, and then we presented detailed algorithms for each part of the architecture. We implemented our model and evaluated the results, which showed very good results. Finally, we proved the correctness of our algorithms.

This paper concentrates on applying queries on one stream. In future, we plan to consider multiple streams and investigate algorithms for the integrity auditing of join queries as well.

REFERENCES

[1] Arasu, A. et al.: STREAM: The Stanford Stream Data Manager. In: International Conference on Management of Data 2003. San Diego, California – Proceedings of the 2003 ACM SIGMOD.

[2] Chandrasekaran, S. et al.: TelegraphCQ: Continuous Dataow Processing for an Uncertain World. in CIDR 2003.

[3] Schmidt, S. et al.: Robust Real-Time Query Processing with QStream. In Proceedings of the 31st International Conference on Very Large Data Bases (VLDB). 2005, Trondheim, Norway 2005.

[4] Abadi, D. et al.: The Design of the Borealis Stream Processing Engine. In 2nd Biennial Conference on Innovative Data Systems Research (CIDR '05) 2005. pp. 277–289.

[5] Abadi, D. J. et al.: A new Model and Architecture for Data Stream Management. The VLDB Journal, Vol. 12, 2003, No. 2, pp. 120–139.

[6] Chandramouli, B. et al.: Data Stream Management Systems for Computational Finance. Computer, Vol. 43, 2010, No. 12, pp. 45–52.

[7] Hacigms, H.—Mehrotra, S.—Iyer, B.: Providing Database as a Service. In Proceedings of the 18th International Conference on Data Engineering 2002, p. 29.

[8] Li, F. et al.: Enabling Authentication of Sliding Window Queries On Streams. In VLDB Endowment, Vienna, Austria 2007.

[9] Merkle, R. C.: A Certified Digital Signature. In Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology 1990.

[10] Papadopoulos, S.—Yang, Y.—Papadias, D.: Continuous Authentication on Relational Streams. The VLDB Journal, Vol. 19, 2010, No. 2, pp. 161–180.

[11] Tatbul, N. et al.: Load Shedding in a Data Stream Manager. In 29th International Conference on Very large Data Bases 2003, Volume 29, pp. 309–320.

[12] Yi, K. et al.: Randomized Synopses for Query Assurance on Data Streams. In Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, pp. 416–425.

[13] Yi, K. et al.: Small Synopses for Group-by Query Verification on Outsourced Data Streams. ACM Trans. Database Systems, Vol. 34, 2009, No. 3, pp. 1–42.

[14] Babcock, B. et al.: Models and Issues in Data Stream Systems. In Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems 2002, pp. 1–16.

[15] Dong, C.—Russello, G.—Dulay, N.: Shared and Searchable Encrypted Data for Untrusted Servers. In Proceeedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security 2008, pp. 127–143.

[16] Brinkman, R.: Searching in Encrypted Data. In Department of Information and Computer Science, University of Twente 2007, p. 119.

[17] Song, D. X.—Wagner, D.—Perrig, A.: Practical Techniques for Searches on Encrypted Data. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, p. 44.

[18] Xie, M. et al.: Integrity auditing of outsourced data. In Proceedings of the 33rd International Conference on Very Large Data Bases 2007, pp. 782–793.

[19] Wang, X.—Liu, H.—Er, D.: HIDS: A Multifunctional Generator of Hierarchical Data Streams. SIGMIS Database, Vol. 40, 2009, No. 2, pp. 29–36.

[20] Xie, M. et al.: Integrity Auditing of Outsourced Data. In VLDB Endowment 2007.

[21] Dill, S. et al.: Self-Similarity in the Web. ACM Trans. Internet Technology, Vol. 2, 2002, No. 3, pp. 205–223.

[22] Fiorini, P. M.: Modeling Telecommunication Systems with Self-Similar Data Traffic. In Department of Computer Science. 1998, The University of Connecticut, p. 56.

[23] McFedries, P.: Definition of Man-in-the-Middle. 2002 [cited 2012 2012/02/17]; Available from: `http://www.wordspy.com/words/maninthemiddleattack.asp`.

[24] Bellare, M. et al.: A Concrete Security Treatment of Symmetric Encryption. In Proceedings of the 38th Annual Symposium on Foundations of Computer Science, 1997, p. 394.

Majid Ghayoori graduated from Iran University of Science and Technology (IUST) in computer engineering in 1991. He received his M. Sc. in computer engineering (artificial intelligence) from Amirkabir Industrial University, Tehran, Iran in 1995. After his graduation, he started to work in some research projects. He accepted as a Ph. D. student in computer engineering department of IUST under guidance of Dr. Haghjoo (2006), and he received his Ph. D. from Iran University of Science and Technology (IUST) in 2012. He currently works in some research projects in database and data stream fileds.

Mostafa S. Haghjoo graduated from Shiraz University (1978) in applied mathematics. He changed to computer science and received his M. Sc. and Applied Scientist degrees from the George-Washington University, Washington, D. C. in 1980 and 1982, respectively. Finally, he received his Ph.Ḋ. from Australian National University in 1995. He is teaching and perusing research in IUST, Iran for the last two decades.