Authenticating Query Results in Data Publishing

Di Ma[1,3], Robert H. Deng[2], Hweehwa Pang[2], and Jianying Zhou[3]

[1]University of California, Irvine
{dma1}@uci.edu
[2]Singapore Management University, Singapore
{robertdeng, hhpang}@smu.edu.sg
[3]Institute for Infocomm Research, Singapore
{madi, jyzhou}@i2r.a-star.edu.sg

Abstract. We propose a communication-efficient authentication scheme to authenticate query results disseminated by untrusted data publishing servers. In our scheme, signatures of multiple tuples in the result set are aggregated into one and thus the communication overhead incurred by the signature keeps constant. Next attr-MHTs (tuple based Merkle Hash Tree) are built to further reduce the communication overhead incurred by auxiliary authentication information (AAI). Besides the property of communication-efficiency, our scheme also supports dynamic SET operations (UNION, INTERSECTION) and dynamic JOIN with immunity to reordering attack.

Keywords: data publishing, authentication, merkle hash tree, aggregated signature.

1 Introduction

1.1 The Third Party Publishing Problem

This paper studies techniques for authenticating query results in the third party publishing scenario shown in Figure 1, where database owners outsource their data management to a third party publisher to disseminate data to users on demand. The motivation of outsourcing is to achieve greater data survivability and higher distribution efficiency: Firstly, as external servers take care of the data management, organizations can concentrate on their core tasks and thus reduce substantial cost on software, hardware and hiring professionals to maintain the in-house system; Secondly, by adding data processing server(s) near user cluster, users can get faster response to their queries from the server; Last, secret keys which are used for protection of the database are kept on the corporate end and not online, so that much better security is achieved.

As valuable information is stored in the third party publishing servers, the servers as well as the data delivery networks are frequently the targets of malicious attacks. Furthermore, the server itself might be malicious. A malicious server may attempt to insert fake records into the database or modify existing records. As a result, the integrity and origin authenticity of query results coming from these servers must be verified before a querier can consume them. Especially when the querier is dependent on the results to make high-stake decisions, she needs strong guarantees of the integrity and accuracy of the data received.

S. Qing et al. (Eds.): ICICS 2005, LNCS 3783, pp. 376–388, 2005.

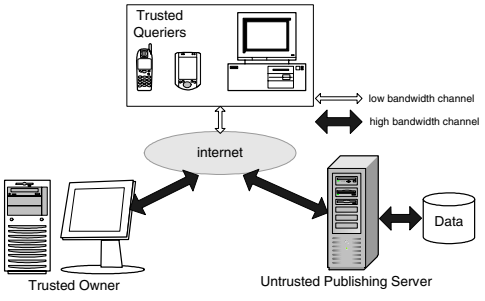

Fig. 1. System Set-Up

1.2 Related Work and Our Contributions

We propose a novel scheme to authenticate query results disseminated by an untrusted third party publishing server. Our scheme is based on the Merkle Hash Tree (MHT) and aggregated digital signatures. A couple of database authentication schemes employing MHT (hereafter refereed to as MHT-based schemes) are proposed in the literature [7,12]. However, the way we make use of MHT is unique. The MHT in previously proposed schemes is constructed over the entire relation, while our MHT (refereed to as attr-MHT) is constructed on individual tuples in order to reduce communication overhead incurred by auxiliary authentication information (AAI). In our view, the disadvantages of constructing a MHT over the entire relation are as follows: 1) it is very expensive to perform data update as any change in the relation will have impact on the whole MHT; 2) as a result of 1), it is not suitable to authenticate frequently changing data such as stock and sales information; 3) it is difficult to reduce the communication overhead incurred by AAI and imposes high processing cost on the querier (e. g., PROJECT has to be performed by the querier [7]); and 4) it does not support dynamic JOIN and SET operations.

Our research is motivated by the work done by Mykletun, et. al. [10] (hereafter we refer to the scheme in [10] as well as our scheme as the aggregated signature based schemes) which explores the use of aggregated signature schemes in database system authentication. Their main contribution is the reduction of communication overhead incurred by signatures, through an aggregated signature scheme. In this paper, we extend and improve their work by constructing attr-MHTs over individual tuples to reduce the communication overhead of AAIs that result mostly from PROJECT operations and applying to all the relational operations. By using MHT and aggregated signature scheme synergetically, our scheme is the first authentication scheme which reduces communication overhead incurred by AAI as well as supports dynamic JOIN and SET operations. The main contributions of our scheme are: 1) achieve constant communication overhead for SELECT and allow a querier to verify all the result tuples with just one digital signature verification operation; 2) minimize communication overhead incurred by AAI for PROJECT operation through optimized attr-MHT; 3) support dynamic SET operations (UNION, INTERSECTION); 4) support dynamic JOIN with immunity to reordering attack.

2 Preliminaries

2.1 The Merkle Hash Tree

We illustrate the construction and application of the MHT with a simple example. The reader is referred to [9] for detailed description. To authenticate data values $n_1, n_2, \ldots, n_w$, the data source constructs the MHT as depicted in Fig. 2 assuming that $w = 4$. The values of the four leaf nodes are the message digests, $H(n_i)$, $i = 1, 2, 3, 4$, respectively, of the data values under a one-way hash function $H(\cdot)$. The value of each internal node of the tree is derived from its child nodes. For example, the value of node A is $h_a = H(H(n_1)\|H(n_2))$ where "$\|$" denote concatenation. The value of the root node is $h_r = H(h_a\|h_b)$ which is used to authenticate any subset of the data values n_1, n_2, n_3, n_4, in conjunction with a small amount of AAI. For example, a user, who is assumed to have the authentic root value h_r, requests for n_3 and requires the authentication of the received n_3. Besides n_3, the source sends the auxiliary information h_a and $H(n_4)$ to the user. The user can then check the authenticity of the received n_3 as follows. The user first computes $H(n_3)$, $h_b = H(H(n_3)\|H(n_4))$ and $h_r = H(h_a\|h_b)$, and then checks if the latter is the same as the authentic root value h_r. Only if when this check is positive, the user accepts n_3. The concept of MHT has been used for certifying query answers over XML documents [6], proving the presence/absence of public key certificates on revocation lists [8,11], certifying data published by untrusted publishers [7] and certifying JPEG2000 sub-images [5].

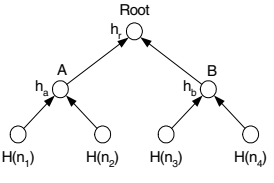

Fig. 2. An example Merkle Hash Tree

2.2 Aggregated Signature Scheme

A digital signature algorithm is a cryptographic tool for generating non-repudiation evidence,for authenticating the integrity of the signed message as well as its origin. An aggregated signature scheme is a digital signature scheme which allows aggregation of multiple individual signatures into one *unified* signature such that verification of the unified signature is *equivalent* to verifying individual component signatures. The concept of aggregated signature is first introduced by Boneh, et al. in [2] (hereafter referred to as BGLS scheme). According to the ability to aggregate signatures from different signers, there are single-signer scheme and multi-signer scheme. Single-signer scheme, like Condensed-RSA [10], aggregates only signatures from the same signer into one unified signature. Multi-signer scheme, like BGLS, can aggregate signatures from multiple signers.

3 System Overview

Our proposed authentication scheme is targeted for the third party publishing model [7] depicted in Fig. 1. The objective of our scheme is to provide adequate security measures to protect the stored data from both the malicious outsider attacks and the server itself.

3.1 Trust Model

The owner of data is fully trusted by the queriers. Only the owner knows the private key for signing individual tuples. Queriers are trusted database clients and a querier verifies query results by checking the data integrity and data origin based on the owner signatures. The data processing server is responsible for replication, backup and dissemination of the outsourced database that it hosts. However, the server is not trusted with the integrity of the data.

There are two kinds of attacks: server side attacks and communication channel attack. Server side attacks refers to attacks happening on the server side. The server is assumed to be unsecured, meaning it is possible for a hacker to tamper (insert, delete, modify) with the data there. Also, the server itself might be malicious and it may attempt to tamper with the data it stores or processes. Insertion attack and modification attack of the data on the sever side can be detected easily at the querier side. Deletion attack is a bit more complex. There are two kinds of deletion attacks happening on the server side: permanent deletion of tuples from the database and dynamic deletion of tuples from the result set (the database remains intact). Dynamic deletion cannot be detected by the querier. It is an unsolved problem in the literature [7,12,10] and stays unsolved in our scheme. However we have some methods to detect permanent deletion attack. One way to detect permanent deletion attack is for the DBMS to maintain a global information map (such as a MHT on the entire database), and to check the integrity of the database periodically. How to maintain the global information map for the entire database is beyond the scope of this paper. Our strategy is putting the responsibility of integrity check of the whole database on the owner side. Although the capability provided by permanent deletion detection is limited but still useful, in the sense that users can be sure that they are working with authentic data, although they cannot be sure they can receive all the relevant data.

When the querier and the server are not communicating over a secure channel, e.g., SSL/TLS, the open communication channel is totally untrusted and the query result set is vulnerable to communication channel attacks such as deletion of records, insertion of spurious records, modification of valid records and reordering of query results from JOIN. Any communication channel attacks can be detected at the querier side.

3.2 Overview of System Operation

Using our scheme, the system in Fig. 1 operates as follows:

1. An owner prepares an authenticated tuple in the database by constructing a MHT on the tuple. As the leaves of our MHT are the digests of attribute values in the

tuple, we refer to such a MHT as attr-MHT to distinguish it from the MHT constructed on the entire table. The owner signs on the root value of the attr-MHT. The authenticated table, consisting of all the tuples and their corresponding signatures, are then uploaded to the publishing server(s) through a high bandwidth channel.
2. The server executes a client's query by selecting records that match the query predicate, generates a verification object $VO = \{\sigma_{1,t}, \Lambda\}$ and sends back the VO along with the result set. A verification object is an object which contains enough information for the client to verify the query result set. The VO consists of two parts. The first part $\sigma_{1,t}$ is a unified aggregated signature calculated from all the tuple signatures in the result set. The second part Λ represents authentication auxiliary information (AAI) which is necessary to authenticate the result set.
3. The client, or the querier, after receiving the result set and the VO, verifies the authenticity of the result set using the VO. As the queriers may have a diversified range of computation and communication capabilities, an important design objective of our authentication scheme is to minimize communication and computation overhead at the querier side.

4 The Scheme

First, we show how to construct attr-MHT and generate tuple signature over it. Next we illustrate how to aggregate tuple signatures over a query result set and discuss considerations in choosing one of the aggregated signature schemes. We then present details on how to construct VOs of result sets from various basic algebraic operations [3].

4.1 The attr-MHT and Calculating Tuple Signature

An attr-MHT is a MHT built on an individual tuple. We use an example to illustrate the construction of an attr-MHT. Suppose there is a table with 16 attributes, for each tuple in the table, we construct an attr-MHT. Fig. 3 shows a binary attr-MHT constructed from one tuple in the table. Though the attr-MHT in our example is a binary and balanced tree, in general it can be non-binary and unbalanced. In the tree of Fig. 3, a leaf node corresponds to an attribute in the tuple and is assigned a value which is the message digest of the attribute value of this tuple. The internal nodes are assigned with values derived from its two child nodes. This process continues until the root value h is computed. The owner generates a tuple digital signature σ on the root value h: $\sigma = SIGN(h)$. To verify the authenticity of this tuple, the querier reconstructs the attr-MHT to compute the root value and then verifies the digital signature using the owners's public key and the computed root value. The querier accepts the received tuple as authentic only if the verification is successful.

The purpose of using attr-MHT in our authentication scheme is to reduce the communication overhead incurred by AAI which is used to verify a record from a PROJECT operation. For example, without using attr-MHT, if attributes A_{10} to A_{16} are filtered out, to authenticate record $< A_1, A_2, \cdots, A_9 >$, the server needs to send seven hash values (the hash values of nodes A_{10} to A_{16}) to the querier. With the help of attr-MHT, only three hash values (the hash values of nodes A_{10}, N_{36} and N_{24}) need to be sent out to

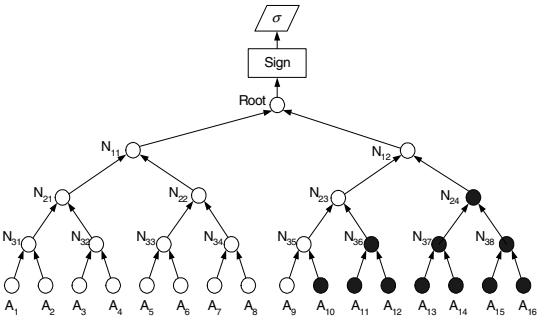

Fig. 3. An example attr-MHT

the querier. More details on how to optimize the attr-MHT to further reduce the size of AAI are discussed in Section 4.3.

When there exist multiple relations in the database, to identify a tuple in a specified relation, the owner generates a digital signature for this tuple with the relation's name. The signature generated with relation's name is represented as $\tilde{\sigma} = SIGN(\tilde{h})$, where $\tilde{h} = H(h \| RELATION\_NAME)$ (h is the root value of the tuple attr-MHT). For example, for a tuple in a relation named "R", its signature with relation's name is calculated as: $\tilde{\sigma} = SIGN(H(h \| \text{“}R\text{”}))$. After generating all the tuple signatures in the database, the owner uploads the authentic database (the original database and all the tuple signatures) to the server.

4.2 Aggregating Tuple Signatures

A query result set usually includes tens, hundreds or even thousands of records. To verify these records, a straightforward solution is that the server sends one signature per record to the querier and the querier verifies these records one by one. This direct solution is not very attractive on two counts. First it is not efficient because it is expensive to verify these records one by one. We thus favor a solution that enables the querier to do batch-verification: verify once and verify all the records. Furthermore, as the size of a database record can be small (relative to the signature and digest) in many cases, the communication overhead incurred by tuple signatures is not negligible. Aggregated signature schemes provide a near perfect solution. Mykletun, et. al. [10] is the first to use aggregate signature schemes to authenticate query results from outsourced databases under three system models, i.e., Unified-Client model, Multi-Querier model, and Multi-Owner model, to reduce communication overhead. Suppose there are n tuples in the result set, the server aggregates these tuple signatures. The aggregation process is depicted in Fig. 4.2 where σ_i denotes the signature of tuple i in the result set and $\sigma_{1,n}$ denotes the aggregated signature of the result set. The aggregated signature $\sigma_{1,n}$, instead of the n individual tuple signatures, is sent to the querier as part of VO for query result verification.

There are some concerns in choosing one of the two aggregated signature schemes to be used in an authentication scheme. First, bandwidth cost concern. The Condensed-

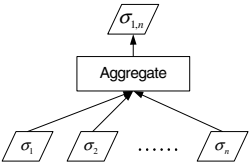

Fig. 4. Aggregating tuple signatures

RSA scheme achieves constant bandwidth in both the Unified-Client and Multi-Querier models but not in the Multi-Owner model while the BGLS scheme achieves constant bandwidth in all the three models.

Second, verification cost and system model concern. The verification costs of both schemes are linear to the number of signatures. The Condensed-RSA scheme is not suitable for the Multi-Owner model, that is, tuples in a database are signed by different signers who create them. For example, a McDonalds database would have sales information from each franchise and each franchise could sign its own sales data. The BGLS signature scheme can aggregate signatures by distinct users into one short signature, however the computational complexity is unfortunately quite high Compared with the Condensed-RSA scheme.

In a Multi-Owner scenario, the BGLS scheme is more favored since it can aggregate any two or more signatures from different signers. Although Condensed-RSA cannot aggregate signatures from different signers, to achieve more computation efficiency, it still can be used in our authentication scheme with a slight modification. If we regard the whole database as a property of an organization, every tuple is owned by the organization and should be organization-signed. In a multi-signer application, if we choose to use organization-signed tuple signatures instead of individual creator signed tuple signatures, Condensed-RSA can be used in the same way as the BGLS is used. In the rest of the paper, we use BGLS as the default signature scheme to illustrate our authentication scheme.

4.3 *VO*s for Relational Algebraic Operations

In the relational data model, queries against a database are formulated in SQL [4]. Such queries are then typically translated by the DBMS query processing engine into expressions of the relational algebra for the purpose of query optimization and execution. We mainly concerns about providing VO for query results where the queries are formulated as expressions of relational algebra. We will illustrate in this Section how to construct VOs for various relational algebraic operations: SELECT, PROJECT, SET operations (UNION, INTERSECT), and JOIN. They are the basic algebraic operations in a relational database system. Although we present a solution for each operation separately, the solutions for individual operations can be combined together to provide a solution for complicated queries, e.g. a SELECT-PROJECT-JOIN query.

***VO* for SELECT.** SELECT is defined as: $\sigma_C(R) := \{t | t \in R \textbf{ and } C(t)\}$ where R is a relation, C is a condition of the form $A_i \Theta c$ where A_i is an attribute of R, $c \in D_i$, D_i

is the domain on which attribute A_i is defined, and $\Theta \in \{=, \neq, <, >, \leq, \geq\}$. It extracts specified tuples from a target relation.

With the definition of VO given in Section 3, the use of aggregated signature to construct a VO is straightforward. Suppose there are t tuples in the result set, the VO for this result set is: $VO = \{\sigma_{1,t}, \phi\}$, where $\sigma_{1,t}$ is the aggregated signature calculated from signatures of the t tuples in the result set and ϕ denotes null. To verify the result set, the querier reconstructs the attr-MHT for each tuple in the result set and calculate the root value of the attr-MHT: h_i, $i = 1, 2, \cdots, t$. Next the querier verifies the query result with $\sigma_{1,t}$ and the calculated h_i.

***VO* for PROJECT.** PROJECT is defined as: $\pi_{A_k,\cdots,A_l(R)} := \{< t.A_k, \cdots, t.A_l > | t \in R\}$. It extracts specified attributes from a target relation. For queries involving PROJECT operations, some attributes of the tuple are filtered out. In this case, additional information need to be sent to the querier to reconstruct the attr-MHT. Suppose there are t records in the result set, the VO for this result set is: $VO = \{\sigma_{1,t}, \Lambda\}$ and $\Lambda = \{\Lambda_i | i = 1, 2, \cdots, t\}$. $\sigma_{1,t}$ is the aggregated signature calculated from the corresponding tuple signatures. Λ_i is the AAI to authenticate record i in the result set. It consists of all the values of the sibling nodes of those nodes on the path from the selected leaf attribute nodes to the root.

For filtered attributes, we have a definition of **Highest Common Ancestor (HCA)**: Several attributes are said to have a HCA if there exist a subtree in the attr-MHT which has all these attributes and only these attributes as leaf nodes and the root of this subtree is not an ancestor of any other attribute. The root of the subtree is called the HCA of these attributes. In a special case, when a filtered attribute has no common ancestor with all the other filtered attributes, it is said the HCA of this attribute is itself. To illustrate, in Fig. 3 A_{10} is the HCA of itself; N_{36} is the HCA of A_{11} and A_{12}; N_{24} is the HCA of A_{13}, A_{14}, A_{15} and A_{16}. Apparently, HCAs are the AAI data needed to be sent to the querier.

To minimize the size of AAI or the number of HCAs, the attr-MHT can be optimized by sorting the attributes according to the attribute query frequency (AQF): the frequency at which an attribute is queried, which can be obtained by query statistics. We give an example to show how to sort the attributes here. Let the most frequently selected attributes come first in the attr-MHT, and the less frequently requested attributes come later. The purpose of this sorting is to produce an ordering of the filtered attributes like A_{10} to A_{16} in Fig. 3, to increase the probability that more attributes will have a HCA and thus reduce the total number of HCAs as far as possible.

***VO* for SET Operations.** We consider SET operations of UNION and INTERSECT. They are both binary operations which build a new logical relation from two specified relations. A tuple in the logical relation can be mapped into a tuple in either of the two specified relations. The construction of VO for these operations over two relations uses tuple signatures with relation's name: $\tilde{\sigma}$.

UNION is defined as: $R \cup S := \{t | t \in R \textbf{ or } t \in S\}$. It builds a logical relation consisting of all tuples appearing in either or both of two specified relations: R and S. For a tuple which belongs to both relations, we make the following rule in choosing which signature for aggregation and verification: if a tuple appears in both relations,

its signature generated by a signer in R (or S) is chosen for signature aggregation; the public key of the signer from R (or S) who signs this tuple is chosen accordingly for signature verification.

Suppose the query result set from a UNIONed operation contains totally t tuples, m (inclusive tuples appearing in both relations) from relation R, n (exclusive tuples appearing in both relations) from relation S, and $m+n=t$. Let $\tilde{\sigma_i^R}$ $(i=1,2,\cdots,m)$ denote the signature of tuple i from R, $\tilde{\sigma_j^S}$ $(j=1,2,\cdots,n)$ the signature of tuple j from S. The VO for this result set is: $VO=\{\sigma_{1,t},\phi\}$ where $\sigma_{1,t}$ is calculated as $\sigma_{1,t}=\prod_{i=1}^{m}\tilde{\sigma_i^R}\times\prod_{j=1}^{n}\tilde{\sigma_j^S}$. The querier verifies the result set according to the equation $e(\sigma_{1,t},g_2)=\prod_{i=1}^{m}e(\tilde{h_i^R},v_i^R)\times\prod_{j=1}^{n}e(\tilde{h_j^S},v_j^S)$ where $\tilde{h_i^R}=H(h_i^R\|\text{“}R\text{”})$ and $\tilde{h_j^S}=H(h_j^S\|\text{“}S\text{”})$). The querier is able to verify the result set containing tuples from either or both of the two relations.

INTERSECT is defined as: $R\cap S:=\{t|t\in R\ \textbf{and}\ t\in S\}$. The INTERSECT operation builds a logical relation consisting of all tuples appearing in both of two specified relations. The querier needs to verify that every tuple in the result set appears in both relations.

Suppose the query result set from a INTERSECTed operation contains totally t tuples. Let h_i $(i=1,2,\cdots,t)$ denote the root value of the attr-MHT for tuple i. Let $\tilde{\sigma_i^R}$ denote its signature with R's name in R, $\tilde{\sigma_i^S}$ its signature with S's name in S. The VO for this result set is: $VO=\{\sigma_{1,t},\phi\}$ where $\sigma_{1,t}$ is calculated as $\sigma_{1,t}=\prod_{i=1}^{t}(\tilde{\sigma_i^R}\times\tilde{\sigma_i^S})$. The querier verifies the result set according to the equation $e(\sigma_{1,t},g_2)=\prod_{i=1}^{t}(e(\tilde{h_i^R},v_i^R)\times e(\tilde{h_i^S},v_i^S))$. The querier is able to verify the result set containing tuples from both of the two relations.

***VO* for JOIN.** JOIN is defined as: $R\bowtie_C S:=\{tq|t\in R\ \textbf{and}\ q\in S\ \textbf{and}\ C(t,q)\}$ where tq is a tuple pair, C is condition of the form $A_j\Theta A_k$, A_j and A_k are attributes of relations R and S respectively, and $\Theta\in\{=,\neq,<,>,\leq,\geq\}$. It builds a relation from two specified relations consisting of all possible concatenated pairs of tuples, one from each of the two specified relations, such that in each pair the two tuples satisfy the specified condition. Without loss of generality, let T denote the relation resulted from a JOIN $R\bowtie_C S$, that is: $T=R\bowtie_C S=\{p_i|i=1,2,\cdots,n\}=\{(t_iq_i)|i=1,2,\cdots,n.\ t_i\in R,q_i\in S,\ \textbf{and}\ t_i.A_j\Theta q_i.A_k=TRUE\}$, where p_i is a tuple of relation T and it is in the form of a fixed tuple pair (t_iq_i).

The VO for a JOIN is in the form of $\{\sigma_{1,n},\phi\}$. $\sigma_{1,n}$ is the aggregated signature by aggregating the signatures of all the n tuples in relation T and is calculated as $\sigma_{1,n}=\prod_{i=1}^{n}\sigma_i^R\times\prod_{i=1}^{n}\sigma_i^S$. The querier uses the equation $e(\sigma_{1,n},g_2)=\prod_{i=1}^{n}e(h_i^R,v_i^R)\times\prod_{i=1}^{n}e(h_i^S,v_i^S)$ to verify the result set from a JOIN. In case that all the tuples are signed by the same entity, $v_i^R=v_i^S$.

However this single step cryptographic signature verification is not enough to authenticate the result set as it cannot detect the reordering attack which is illustrated in Fig. 5. An attacker switches the q element in the two tuples $p_1=(t_1q_1)$ and $p_2=(t_2q_2)$. After the switch, p_1 becomes $p_1'=(t_1q_2)$ and p_2 becomes $p_2'=(t_2q_1)$ and the conditions $C(t_1,q_2)$ and $C(t_2,q_1)$ may no longer hold. Because p_1 and p_1', p_2 and p_2' are different in values, they may convey wrong results to the querier and thus en-

danger the querier's decisions. The reordering attack can occur both on the server side when the server behaves maliciously or is compromised by an outside attacker or during the transmission process when the querier and the server are not communicating over a secure channel, e.g., SSL/TLS. When the server is compromised, the attacker controls the server and reorders the result set before the server sends it out to the querier. When the querier and the server are not communicating over a secure channel, the attacker is able to eavesdrop on the conversation and reorders the result set without destroying it. This reordering attack cannot be detected using only cryptographic verification because the verification of an aggregated signature is order independent and does not check the validity of JOIN condition.

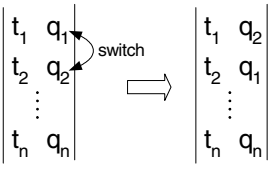

Fig. 5. Reordering attack

To detect reordering attacks, besides the cryptographic signature verification, the querier must also check if the condition $C(t_i, q_i)$ still holds for each p_i in the received result set. Thus the authentication process for a JOIN consists of two steps: condition check and signature verification. Before constructing the two attr-MHTs for each tuple p_i in the result set, the querier checks the condition $C(t_i, q_i)$ in p_i first. If the condition $C(t_i, q_i)$ does not hold the authentication of the result set fails. If the condition check for all the tuples in the result set are successful, the querier further cryptographically verifies the aggregated signature with all the calculated attr-MHT root values. Only when the condition check and the signature verification are both successful can the result set be regarded as authenticated.

4.4 Update Operations

As our attr-MTH is built on individual tuple and each tuple in the table is signed separately and the signature aggregate function is accumulative and communicative, update operations (INSERT, DELETE and UPDATE) can be processed in a direct way: DELETE can be processed by the server with the owner modifying the database's global information map accordingly; INSERT and UPDATE have to be channelled back to the owner as only the owner possesses the private key for generating new signatures. The owner signs on the new (updated) tuple and sends the new (updated) tuple with the newly generated signature to the server. There is no need to lock the whole table during the updating process. However, to ensure data consistency, the update operations should be done under a concurrency control mechanism like basic 2PL [1].

5 Analysis and Evaluation

We analyze our scheme in terms of the following five overheads as defined in [10]: querier communication, querier computation, server computation, server storage and owner computation. The parameters used in the analysis are summarized in Tab. 1:

Table 1. Notations

Notation	Meaning	Default
$\|S\|$	Length of a signature (Bytes)	128
$\|D\|$	Length of a hash value (Bytes)	16
T_R	Number of tuples/rows in table (million)	1
T_A	Number of attributes/columns in table	10
Q_R	Number of tuples in the result set	-
Q_A	Number of filtered attributes	-
$\|A_i\|$	Size of attribute A_i (Bytes)	-

5.1 Querier Communication Overhead

The querier communication overhead consists of two parts: overhead incurred by the aggregated signature and overhead incurred by AAIs. We analyze these two kinds of overheads separately.

Overhead Incurred by Signature. The communication overhead incurred by signature remains constant as $|S|$, the same as in [10]. It is independent on the number of records in the result set. In contrast, the communication overhead incurred by signature in the MHT-based schemes is linear to Q_R.

Overhead Incurred by Authentication Auxiliary Information. Let N_A denote the number of digests as AAI per tuple in the result. Suppose the Q_A filtered attributes are in a continuous sequence in the end of the sorted attribute list. N_A can be represented as:

$$N_A = \begin{cases} 1, \text{ if } \log_2 Q_A \text{ is an integer;} \\ [2,\, 2^{\lfloor \log_2 Q_A \rfloor} + 1], \text{ if } \log_2 Q_A \text{ is not an integer.} \end{cases}$$

When $\log_2 Q_A$ is an integer, that is $Q_A = 1, 2, 4, \cdots$, only one digest per record needs to be sent out. When $\log_2 Q_A$ is not an integer, N_A is in the range of $[2,\, 2^{\lfloor \log_2 Q_A \rfloor} + 1]$. Its maximum value $2^{\lfloor \log_2 Q_A \rfloor} + 1$ is obtained when $\log_2(Q_A + 1)$ is an integer. Thus in the worst case when $\log_2(Q_A + 1) = n$ (n is an integer), the ratio of N_A/Q_A is:

$$\frac{N_A}{Q_A} = \frac{2^{\lfloor \log_2 Q_A \rfloor} + 1}{Q_A} = \frac{2^{\lfloor \log_2 (2^n - 1) \rfloor} + 1}{2^n - 1} = \frac{2^{n-1} - 1}{2^n - 1} < \frac{1}{2} \tag{1}$$

From Eq. 1, we conclude that by employing attr-MHT our scheme reduces the communication overhead incurred by AAI at least by half compared with all the existing authentication schemes.

5.2 Querier Computation Cost

The querier computation cost consists of two parts: hashing cost spent on the reconstruction of the attr-MHTs and verifying cost on the aggregated signature.

Hashing cost on the reconstruction of the attr-MHTs is not significant compared to the cost on the aggregated signature verification which is a very expensive operation. Normally, hash functions are about 100 times faster than RSA signature verification [13], more than 1000 times faster than BGLS signature verification. The cost of hashing is dependent on the total length of input bits and is represented as: $Comp_{hashing} = Q_R * (2 * (T_A - 1) * |D| + \sum_{i=1}^{T_A} |A_i|)$.

With signature aggregation the querier only verifies one signature to authenticate multiple tuples which is very efficient compared with a naive solution which verifies tuple signatures one by one. To verify a BGLS aggregated signature generated from these $k \times t$ component signatures, it costs $k + 1$ bilinear mapping plus $k \times t - 1$ multiplication operations, that is: $Comp_Q^{aggregated} = Multi^{k*t-1}(p) + BM(k + 1)$. To authenticate a query result set from a JOIN, there are additional computation cost incurred by condition check for the querier. However, condition check incurs only a little computation overhead as the comparison operation is very efficiently implemented in modern computers.

5.3 Server Computation Cost

Upon a query, besides preparing the result set, the server needs to construct a VO for the result set by aggregating multiple signatures. The server computation cost is calculated as: $Comp_S = Multi^{Q_R-1}(p) = 0.12 \times (Q_R - 1)$. The multiplication operation cost involved are not expensive given that $Mult^1(1) = 0.12ms$ [10], and can be easily mitigated with a powerful server machine. Furthermore the resulting savings in communication overhead and processing cost at the querier side more than justify our proposed scheme.

5.4 Server Storage Cost

From the system management angle, although the disk units get cheaper, storage management costs are not getting cheaper and thus the server storage cost is a concern especially in outsourcing models. The attr-MHT incurs no server storage cost. After the signature is calculated, the tree structure which stores all the hash values of the intermediate nodes in the attr-MHT is discarded and only the signature is stored with the physical database. Thus the total cost of server storage of our scheme is calculated as: $Storage_S = T_R * |S|$.

5.5 Owner Computation Cost

To construct an authentic database, the owner needs to construct an attr-MHT and calculate a signature based on the root value of the attr-MHT for each tuple in the database. The signature generation process can be done off-line. As stated in Section 4.4, because the signature aggregate function is accumulative and communicative, update operations can be processed in a simple and direct way and there is no need to lock the whole table.

6 Conclusion

In this paper, we proposed a communication-efficient scheme based on aggregated signature schemes and MHT to authenticate query results disseminated by an untrusted data processing server. To our knowledge, this is the first work that addresses the issue of reducing communication overhead incurred by AAI and the first authentication scheme which supports dynamic JOIN and SET operations.

References

1. P. Bernstein and N. Goodman. Concurrency control in districuted database systems. In *ACM Computing Surveys*, Volume 13(2), pages 185-221, June 1981.
2. D.Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. in *Advances in Cryptology - EUROCRYPT'2003* (E. Biham, ed.), Lecture Notes in Computer Science, International Association for Cryptologic Research, Springer-verlag. Berlin, Germany, 2003
3. C.J. Date. An introduction to database systems (4th Edition). Addison-Wesley, 1985.
4. C.J. Date and H. Darwen. A guide to the SQL Standard (4th Edition). Addison-Wesley, 1997,
5. R. H. Deng, Y. Wu, D. Ma. Securing JPEG2000 Code-Streams. *International Workshop on Advanced Developments in Software and Systems Security*, Dec. 2003
6. P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls and G. Stubblebine. Flexible authentication of XML documents. *Proc. of the 8th ACM conference on Computer and Communication Security*, pp. 136-145, 2001.
7. P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. Authentic data publication over the internet. In *14th UFIP 11.3 Working Conference in Database Security,* Pages 102-112,2002.
8. M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing. *Proc. of DISCEX II'01*, Vol. 2, pp. 1068-1083, 2001.
9. R. C. Merkle. A certified digital signature. *Proc. of Advances in Cryptology-Crypto '89*, Lecture Notes on Computer Science, Vol. 0435, pp. 218-238, Spriner-Verlag, 1989.
10. E. Mykletun, M. Narasimha, and G. Tsudik, Authentication and integrity in outsourced databases. *NDSS 2004*, Feb. 2004.
11. M. Naor and K. Nissim. Certificate Revocation and Certificate Update. *Proc. of the 7th USENIX Security Symposium*, pp. 217-230, 1999.
12. H.H. Pang, K.L. Tan. Autehticating query results in edge computing. *ICDE 2004*, Mar. 2004.
13. R. Rivest and A. Shamir. PayWord and MicroMint–Two Simple Micropayment Schemes in *Proceedings of 1996 International Workshop on Security Protocols*, (ed. Mark Lomas), Lecture Notes in Computer Science No. 1189, pages 69–87. Springer, 1997.