RDAS: A Symmetric Key Scheme for Authenticated Query Processing in Outsourced Databases

Lil María Rodríguez-Henríquez and Debrup Chakraborty

Departamento de Computación, CINVESTAV-IPN
Av. IPN No. 2508, Col. San Pedro Zacatenco, México, D.F. 07360, México
lrodriguez@computacion.cs.cinvestav.mx, debrup@cs.cinvestav.mx

Abstract. Security of outsourced databases is an important problem of current practical interest. In this paper we address the problem of authenticated query processing in outsourced databases. We describe the syntax of a generic scheme for authenticated query processing called RDAS, and provide security definitions for RDAS in line with concrete provable security. Then, we propose a new scheme called RDAS1 which enables a client to ensure both correctness and completeness of the query results obtained from a server. Our solution involves use of bitmap indices and message authentication codes in a novel manner. We prove that RDAS1 is secure relative to our security definition. Finally, we discuss a concrete improvement over RDAS1 (which we call RDAS2) and provide performance data for both RDAS1 and RDAS2 on a real data base.

Keywords: Database security, query authentication, bitmap index, MACs.

1 Introduction

Cloud computing holds the promise of revolutionizing the manner in which enterprises manage, distribute, and share information. The data owner (client) can out-source almost all its information processing tasks to a "cloud". The cloud can be seen as a collection of servers (we shall sometimes refer to it as the server) which caters the data storage, processing and maintenance needs of the client. Needless to say this new concept of computing has already brought significant savings in terms of costs for the data owner.

Among others, an important service provided by a cloud is *Database as a Service (DAS)*. In this service the client delegates the duty of storage and maintenance of his/her data to a third party (an un-trusted server). This model has gained lot of popularity in the recent times. The DAS model allows the client to perform operations like create, modify and retrieve from databases in a remote location [6]. These operations are performed by the server on behalf of the client. However, delegating the duty of storage and maintenance of data to a third party brings in some new security challenges.

R. Accorsi and S. Ranise (Eds.): STM 2013, LNCS 8203, pp. 115–130, 2013.

The two main security goals of cryptography are privacy and authentication. These security issues are relevant to the outsourced data also. The client who keeps the data with an untrusted server has two main concerns. The first one being that the data may be sensitive and the client may not want to reveal the data to the server and the second one is the data whose storage and maintenance has been delegated to the server would be used by the client. The typical usage of the data would be that the client should be able to query the database and the answers to the client's queries would be provided by the server. It is natural for the client to be concerned about a malicious server who does not provide correct answers to the client queries. In this work we are interested in this problem. We aim to devise a scheme in which the client would be able to verify whether the server is responding correctly to its queries.

We consider the scenario where a client delegates a relational data base to an un-trusted server. When the client queries its outsourced data, it expects in return a set of records (query reply) satisfying the query's predicates. As the server is not trusted, so it must be capable of proving the correctness of its responses. We describe the intricacies of the problem with the help of an example. Consider the relational database of employees data shown in Table 1.

Table 1. Relation $R1$ (This relation would serve as a running example)

EmpId	Name	Gender	Level
TRW	Tom	M	L_2
MST	Mary	F	L_1
JOH	John	M	L_2
LCT	Lucy	F	L_1
ASY	Anne	F	L_1
RZT	Rosy	F	L_2

We consider that this relation has been delegated by a client to a server, and the client poses the following query

$$\texttt{SELECT * FROM } R1 \texttt{ WHERE } Gender = \texttt{'}M\texttt{' OR } Level = \texttt{'}L_2\texttt{'}.$$

The correct response to this query is the set Res consisting of three tuples

$$\mathsf{Res} = \{(\text{TRW, Tom, M}, L_2), (\text{JOH, John, M}, L_2), (\text{RZT, Rosy, F}, L_2)\}.$$

In answering the query the server can act maliciously in various ways. In the context of authentication we are concerned with two properties of the response namely *correctness* and *completeness*, denote two different malicious activities of the server. We explain these notions with an example below:

1. **Incorrect result:** The server responds with three tuples, but changes the tuple (TRW, Tom, M, L_2), to (TRW, Tom, F, L_2). Moreover, it can be the case that the server responds with Res$\cup\{$(BRW, Bob, M, L_2), i.e., it responds with an extra tuple which is not a part of the original relation.

2. **Incomplete result:** The server may not respond with the complete result, i.e., it can delete some valid results from the response, i.e., instead of responding with Res it responds with $\mathsf{Res} - \{(\text{TRW, Tom, M}, L_2)\}$.

The problem of correctness can be easily handled in the symmetric setting by adding a message authentication code to each tuple. A secure message authentication code is difficult to forge, and thus this property would not allow the server to add fake entries in its response. The completeness problem is more difficult and its solution is achieved through more involved schemes.

The problem of query completeness has been largely addressed by some interesting use of authenticated data structures. The basic idea involved is to store the information already present in the relation in a different form using some special data structures. This redundancy along with some special structural properties of the used data structures help in verifying completeness.

A large part of the literature uses tree based authentication structures like the Merkeley hash tree [8] or its variants. Some notable works in this direction are reported in [3, 5, 7–9, 13, 14, 19]. These techniques involve using a special data structure along with some cryptographic authentication mechanism like hash functions and/or signatures schemes. The tree based structures yield reasonable communication and verification costs. But, in general they require huge storage at server side, moreover the query completeness problem is largely addressed with respect to range queries and such queries may not be relevant in certain scenarios, say in case of databases with discrete attributes which do not have any natural metric relationship among them.

Signature schemes have also been used in a novel manner for solving the problem. One line of research has focussed on aggregated signatures [10–12, 15, 16]. Signature aggregation helps in reducing the communication cost to some extent and in some cases can function with constant extra communication overhead. A related line of research uses chain signatures. If one uses chain signatures as in [11], the use of specialized data structures may no longer be required.

Our Contributions: Though there have been considerable amount of work on authenticated query processing on relational data bases, but it has been acknowledged (for example in [20]) that the problem of query authentication largely remains open. An unified cryptographic treatment of the problem is missing in the literature. In most existing schemes cryptographic objects have been used in an ad-hoc manner, and the security guarantees that the existing schemes provide are not very clear. In this work we initiate a formal cryptographic study of the problem of query authentication in a distinct direction. We propose a new scheme which does not use any specialized data structure to address the completeness problem. Our solution involves usage of bitmap indices for this purpose. Bitmap indices have gained lot of popularity in the current days for their use in accelerated query processing [18], and many commercially available databases like Oracle, IBM DB2, Sybase IQ now implement some form of bitmap index scheme in addition to the more traditional B-tree based schemes. Thus, it may be easy to incorporate a bitmap based scheme in a modern database without significant extra cost. To our knowledge, bitmaps have not been used till

date for a security goal. In addition to bitmap indices we use a secure message authentication code (MAC) as the only cryptographic object. We show that by the use of these simple objects one can design a query authentication scheme which allows verification of both correctness and completeness of query results.

In concrete terms in this paper we describe a generic scheme which we call as relational database authentication scheme (RDAS) which can provide the functionality of authenticated query processing in static databases. We define the security goals of RDAS in line with the tradition of concrete provable security. Then we propose a RDAS called RDAS1. RDAS1 is designed using message authentication codes and bitmap indices in a novel manner. RDAS1 is capable of authenticated query processing of simple select queries and select queries involving disjunctions of equality conditions. We point out various directions in which RDAS1 can be modified to incorporate other types of queries. In particular we propose a modification called RDAS2 which is capable of authenticating a larger class of queries. Finally we provide some experimental data on performance of RDAS1 and RDAS2.

2 Preliminaries and Notations

Relations: By $R(A)$ we would denote a relation over a set of attributes A. If $A = \{a_1, a_2, \cdots a_n\}$, we shall sometimes write $R(a_1, a_2, \cdots, a_n)$ instead of $R(A)$. We will assume that each attribute has a set of permitted values, i.e., the domain of the attribute. Given an attribute a, $\mathsf{Dom}(a)$ would represent its domain. We are mainly concerned with attributes whose domains are finite, note that for a static database each attribute always has a finite domain. By cardinality of an attribute we shall mean the cardinality of the domain of the attribute. We will denote the cardinality of an attribute a by $\mathsf{Card}(a) = |\mathsf{Dom}(a)|$.

A tuple t in a relation is a function that associates with each attribute a value in its domain. Specifically if $A = \{a_1, a_2, \cdots a_n\}$ and $R(A)$ be a relation then the j^{th} tuple of relation $R(A)$ would be denoted by t_j^R and for $a_i \in A$ by $t_j^R[a_i]$ we shall denote the value of attribute a_i in the j^{th} tuple in R. For $B \subseteq A$, $t_j^R[B]$ will denote the set of values of the attributes in B in the j^{th} tuple. We shall sometimes omit the subscripts and superscripts from t_j^R and denote the tuple by t if the concerned relation is clear from the context and the tuple number is irrelevant.

Binary Strings: The set of all binary strings would be denoted by $\{0,1\}^*$, and the set of n bit strings by $\{0,1\}^n$. For $X_1, X_2 \in \{0,1\}^*$, by $X_1||X_2$ we shall mean the concatenation of X_1 and X_2; and $|X_1|$ will denote the length of X_1 in bits. By $\mathsf{bit}_i(X)$ we will denote the i^{th} bit of X. We shall always consider that the domains of all attributes in the relations are subsets of $\{0,1\}^*$, this would allow us to apply transformations and functions on the values of the tuples without describing explicit encoding schemes.

Bitmaps: Consider a relation $R(a_1, \ldots, a_m)$ with nT many rows. Consider that for each attribute a_i, $\mathsf{Dom}(a_i) = \{v_1^i, v_2^i, \ldots v_{\lambda_i}^i\}$, thus $\mathsf{Card}(a_i) = \lambda_i$ for $1 \leq$

$i \leq m$. We define the bitmap of an attribute a_i corresponding to its value v_j^i in the relation R as $\mathsf{BitMap}_R(a_i, v_j^i) = X$, where X is a binary string, such that $|X| = \mathsf{nT}$ and for $1 \leq k \leq \mathsf{nT}$,

$$\mathsf{bit}_k(X) = \begin{cases} 1 & \text{if } t_k^R[a_i] = v_j^i \\ 0 & \text{otherwise.} \end{cases}$$

Consider the relation $R1$ on the attributes $\{\mathtt{EmpID}, \mathtt{Name}, \mathtt{Gender}, \mathtt{Level}\}$ as shown in Table 1. Here we have $\mathsf{Dom}(\mathtt{Gender}) = \{\mathrm{M}, \mathrm{F}\}$ and $\mathsf{Dom}(\mathtt{Level}) = \{L_1, L_2\}$. Hence we can compute the following bitmaps:

$$\mathsf{BitMap}_{R1}(\mathtt{Gender}, \mathrm{F}) = 010111\ ,\ \mathsf{BitMap}_{R1}(\mathtt{Gender}, \mathrm{M}) = 101000$$
$$\mathsf{BitMap}_{R1}(\mathtt{Level}, L_1) = 010110\ ,\ \mathsf{BitMap}_{R1}(\mathtt{Level}, L_2)\ = 101001$$

Message Authentication Codes: Message authentication codes provide authentication in the symmetric key setting. It is assumed that the sender and the receiver share a common secret key K. Given a message x, the sender uses K to generate a footprint of the message. This footprint (commonly called a tag) is the message authentication code (MAC) for the message x. The sender transmits the pair $(x; \mathsf{tag})$ to the receiver. The receiver uses K to verify that (x, tag) is a properly generated message-tag pair. Verification is generally performed by regenerating the tag on the message x and comparing the generated tag with the one received. We shall call the algorithm for generating the tag as a MAC. Assuming that the size of the tag is τ bits, we see the tag generation scheme as a function $\mathsf{MAC} : \mathcal{K} \times \mathcal{M} \rightarrow \{0,1\}^\tau$, where $\mathcal{K}$ and $\mathcal{M}$ are the key and message spaces respectively. In most cases we shall write $\mathsf{MAC}_K(x)$ instead of $\mathsf{MAC}(K, x)$.

3 Relational Database Authentication Scheme (RDAS): Definitions and Basic Notions

A relational database authentication scheme (RDAS) consists of a tuple of algorithms $(\mathcal{K}, \mathcal{F}, \Phi, \Psi, \mathcal{V})$, which are described in details in the following paragraphs.

$\mathcal{K}$ is the **key generation algorithm** and it selects one (or more) keys from a pre-specified key space and outputs them.

$\mathcal{F}$ is called the **authentication transform**, which takes in a set of relations $\mathcal{R}$ and a set of keys and outputs another set of relations $\mathcal{R}'$ along with some additional data (M_s, M_c). If the set of keys is K, we shall denote this operation as $(\mathcal{R}', M_c, M_s) \leftarrow \mathcal{F}_K(\mathcal{R})$. A client who wants to store the set of relations $\mathcal{R}$ in an un-trusted server, transforms $\mathcal{R}$ to $\mathcal{R}'$ using the authentication transform $\mathcal{F}$ and a set of keys. The transform $\mathcal{F}$ produces some additional data other than the set of relations $\mathcal{R}'$, the additional data consists of two distinct parts M_s and M_c. The set of relations $\mathcal{R}'$ along with M_s are stored in the server and the keys and the data M_c are retained in the client. The key generation algorithm and the authentication transform are executed in the client side.

We call Φ as the **query translator**, it is a transformation which takes in a query for the relations in $\mathcal{R}$ and converts it into a query for relations in $\mathcal{R}'$. For ease of discussion we shall refer a query for $\mathcal{R}$ to be a $\mathcal{R}$-query and a query for $\mathcal{R}'$ to be a $\mathcal{R}'$-query. Thus, given a $\mathcal{R}$-query q, $\Phi(q)$ would be a $\mathcal{R}'$-query. Thus by use of the transform Φ, the client would be able to translate queries meant for $\mathcal{R}$ to queries which can be executed on the transformed relations in $\mathcal{R}'$.

Ψ is the **response procedure**. To execute a query q on $\mathcal{R}$, the client converts the query to $\Phi(q)$ and sends it to the server. The server executes the function Ψ, which takes in the query $\Phi(q)$ and uses $\mathcal{R}'$ and M_s. The output of Ψ is ρ, which we call as the response of the server. The server returns it to the client.

The **verification procedure** is a keyed transform $\mathcal{V}_K$ which runs in the client. It takes as input the query q, a response ρ of the server and M_c and outputs either an answer ans for the query q or outputs a special symbol $\perp$ which signifies reject.

3.1 Correctness and Security

If we fix the set of relations $\mathcal{R}$, then a $\mathcal{R}$-query q when executed in $\mathcal{R}$ would have a fixed answer say $\mathsf{ans}(\mathcal{R}, \mathsf{q})$. Our goal is to transform $\mathcal{R}$ to $\mathcal{R}'$ using a RDAS in such a way that if the query $\Phi(q)$ is sent to the server, then the answer ans should be recoverable from the server response ρ through the procedure $\mathcal{V}$, if the server follows the protocol correctly. On the other hand, if the server is malicious, i.e., it deviates from the protocol and sends a response ρ' distinct from the correct response ρ then the procedure $\mathcal{V}$ should reject the response by outputting $\perp$. In other words, if the answer to a $\mathcal{R}$-query is ans, then after running the protocol, $\mathcal{V}$ will either produce ans or $\perp$, it would not produce an answer ans' distinct from ans.

In the security model, we allow the adversary to choose the primary set of relations $\mathcal{R}$. Given this choice of $\mathcal{R}$, we compute $(\mathcal{R}', M_c, M_s) \leftarrow \mathcal{F}_K(\mathcal{R})$, for a randomly selected set of keys K which is unknown to the adversary. We give $\mathcal{R}'$ and M_s to the adversary. The adversary chooses a $\mathcal{R}$-query q and the challenger provides the adversary with $\Phi(q)$, finally the adversary outputs a response ρ, and we say that the adversary is *successful* if $\mathcal{V}_K(\rho, q, M_c) \notin \{\perp, \mathsf{ans}(\mathcal{R}, \mathsf{q})\}$.

Definition 1. *Let* $\mathsf{Succ}_{\mathcal{A}}$ *be the event that a specific adversary* $\mathcal{A}$ *is successful in the sense as described above. We say that a RDAS is* (ϵ, t)*-secure if for any adversary* $\mathcal{A}$ *which runs for time at most* t*,* $\Pr[\mathsf{Succ}_{\mathcal{A}}] \leq \epsilon$.

4 RDAS1: A Generic Scheme for Select Queries Involving Arbitrary Disjunctions

We discuss a basic scheme for a secure RDAS which works only if the queries made are single attribute select queries or select queries involving disjunctions of an arbitrary number of equality conditions. We call this scheme as RDAS1.

We describe the scheme assuming that the set of initial relations $\mathcal{R}$ is a singleton set consisting of a single relation $R(B)$, where $B = \{b_1, b_2, \ldots, b_{|B|}\}$ is the set of attributes, and consider $A = \{a_1, \ldots, a_m\} \subseteq B$ to be a set of attributes on which queries are allowed, we shall call A the set of *allowed attributes*. It is possible that $B = A$. The procedure $\mathcal{F}$ converts R into two relations R_α and R_β, i.e, $\mathcal{R}' = \{R_\alpha, R_\beta\}$ and M_s is empty and $M_c = \mathsf{nT}$, where nT is the number of tuples in R. The only cryptographic object used by RDAS1 is a message authentication code $\mathsf{MAC} : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^\tau$, where $\mathcal{K}$ is the key space. In what follows, we shall describe the procedures involved in RDAS1 considering a generic relation $R(B)$, where the set of allowed attributes is $A \subseteq B$. Also we shall throughout consider the relation $R1$ in Table 1 as a concrete example, and for simplicity, for $R1$ we shall consider the set of allowed attributes to be $\{\texttt{Gender}, \texttt{level}\}$.

RDAS1.$\mathcal{K}$: The key space for RDAS1 is the same as the key space of the associated message authentication code MAC. The key generation algorithm selects a key K uniformly at random from $\mathcal{K}$.

RDAS1.$\mathcal{F}$: $\mathcal{F}$ produces two relations R_α and R_β by the action of the key. The relation R_α is defined on the set of attributes $B \cup \{\texttt{Nonce}, \texttt{Tag}\}$, i.e., R_α has two more attributes than in R. If R contains nT many tuples then R_α also contain the same number of tuples. The procedure for populating the tuples of R_α is depicted in Figure 1. What this procedure does is compute a MAC for each row. The relation R_β contains the attributes $\{\texttt{Name}, \texttt{SearchKey}, \texttt{RowNo}, \texttt{Tag1}\}$, irrespective of the attributes in relation R. Where $\mathsf{Dom}(\texttt{Name}) = \{a_1, \ldots, a_m\}$, i.e., the allowed attributes in R. And, $\mathsf{Dom}(\texttt{SearchKey}) = \mathsf{Dom}(a_1) \cup \mathsf{Dom}(a_1) \cup \cdots \cup \mathsf{Dom}(a_m)$. Let $\Omega = \cup_{i=1}^{m} (\{a_i\} \times \mathsf{Dom}(a_i))$, note that the elements of Ω are ordered pairs of the form (x, y) where $x \in \mathsf{Dom}(\texttt{Name})$ and $y \in \mathsf{Dom}(\texttt{SearchKey})$, and $|\Omega| = \sum_{i=1}^{m} \mathsf{Card}(a_i) = N$. Let $\mathcal{L}$ be a list of the elements in Ω in an arbitrary order. If (x, y) be the i-th element in $\mathcal{L}$, then we shall denote x and y by $\mathcal{L}_i^1$ and $\mathcal{L}_i^2$ respectively, where $1 \leq i \leq N$. The way the relation R_β is populated is also shown in Figure 1. This procedure allows the client to store all possible pairs $\mathcal{L}_i^1, \mathcal{L}_i^2$ along with the MAC calculated over this pair concatenated with the respective bitmap and RowNo. Note that the bitmap is not explicitly stored in the relation R_β. The transform $\mathcal{F}$ is executed in the client side, and the resulting relations R_α and R_β are stored in the server.

For a concrete example, if RDAS1.$\mathcal{F}$ has as input the relation $R1$ (see Table 1) and the set of allowed attributes is $\{\texttt{Gender}, \texttt{level}\}$, then it would produce as output the relations $R1_\alpha$ and $R1_\beta$ as shown in Table 2. The relation $R1_\alpha$ is almost the same as that of $R1$, except that it has two additional attributes, Nonce and Tag. The attribute Nonce just contains the row numbers and is thus unique for each row. The attribute Tag is the message authentication code computed for a message which is produced by concatenating all the values of the attributes in that tuple.

The relation $R1_\beta$ contains the attributes $\{\texttt{Name}, \texttt{SearchKey}, \texttt{RowNo}, \texttt{Tag1}\}$, where in this case, $\mathsf{Dom}(\texttt{Name}) = \{\texttt{Gender}, \texttt{Level}\}$, $\mathsf{Dom}(\mathsf{SearchKey}) = \{M, F\} \cup$

Creating R_α
1. **for** $j = 1$ to nT
2. **for** $i = 1$ to $|B|$
3. $t_j^{R_\alpha}[b_i] \leftarrow t_j^R[b_i]$;
4. **end for**
5. $t_j^{R_\alpha}[\mathtt{Nonce}] \leftarrow j$;
6. $H \leftarrow t_j^R[b_1]||| \ldots ||t_j^R[b_m]|||j$;
7. $t_j^{R_\alpha}[\mathtt{Tag}] \leftarrow \mathsf{MAC}_K(H)$;
8. **end for**

Creating R_β
1. **for** $j = 1$ to N
2. $t_j^{R_\beta}[\mathtt{Name}] \leftarrow \mathcal{L}_j^1$;
3. $t_j^{R_\beta}[\mathtt{SearchKey}] \leftarrow \mathcal{L}_j^2$;
4. $t_j^{R_\beta}[\mathtt{RowNo}] \leftarrow \mathsf{nT} + j$;
5. $L \leftarrow \mathcal{L}_j^1||\mathcal{L}_j^2||\mathsf{BitMap}_R(\mathcal{L}_j^1, \mathcal{L}_j^2)|||(\mathsf{nT} + j)$;
6. $t_j^{R_\beta}[\mathtt{Tag1}] \leftarrow \mathsf{MAC}_K(L)$;
7. **end for**

Fig. 1. Creating R_α and R_β

$\{L_1, L_2\}$. The tuples in $R1_\beta$ are populated according to the procedure as shown in Figure 1, and the specific relation $R1_\beta$ is shown in Table 2.

RDAS1.Φ: The transform Φ, transforms a query meant for the original relation R to a set of queries which are meant to be executed on the relations R_α and R_β which are stored in the server side. As mentioned, the allowed queries for RDAS1 are of the following form:

Q: `SELECT * FROM` R `WHERE` $a_1 = v_1$ `OR` $a_2 = v_2$ `OR` ..... `OR` $a_l = v_l$

Table 2. Relations $R1_\alpha$ and $R1_\beta$

Relation $R1_\alpha$

EmpId	Name	Gender	Level	Nonce	Tag
TRW	Tom	M	L_2	1	Y_1
MST	Mary	F	L_1	2	Y_2
JOH	John	M	L_2	3	Y_3
LCT	Lucy	F	L_1	4	Y_4
ASY	Anne	F	L_1	5	Y_5
RZT	Rosy	F	L_2	6	Y_6

Relation $R1_\beta$

Name	SearchKey	RowNo	Tag1
Gender	F	7	Y_7'
Gender	M	8	Y_8'
Level	L_1	9	Y_9'
Level	L_2	10	Y_{10}'

The allowed set of queries are thus select queries on arbitrary numbers of disjunctions on different or repeated attributes [1], which includes select queries on a single attribute of the form `SELECT * FROM` R `WHERE` $a_i = v$. Given as input a valid query q, $\Phi(q)$ outputs two queries one for the relation R_α (which we call q_α) and the other for R_β (which we call q_β). For the specific query Q, $\Phi(Q)$ will output the following queries:

Q_α: `SELECT * FROM` R_α `WHERE` $a_1 = v_1$ `OR` $a_2 = v_2$ `OR` ..... `OR` $a_l = v_l$
Q_β: `SELECT * FROM` R_β `WHERE (Name` $= a_1$ `AND SearchKey` $= v_2$`) OR` ..... `OR (Name` $= a_l$ `AND SearchKey` $= v_l$`)`

In the concrete example, consider the following query $Q1$ on the relation $R1$

$Q1$: `SELECT * FROM` $R1$ `WHERE Gender = 'M' OR Level= '`L_2`'`

[1] By a query of disjunction on repeated attributes we mean a query like: `SELECT * FROM` R `WHERE` $a_1 = v_1$ `OR` $a_1 = v_2$ `OR` $a_2 = v_3$. Here the attribute a_1 is repeated twice.

Applying the transformation $\Phi(Q1)$, the output queries $Q1_\alpha$ and $Q1_\beta$ would be the following:

$Q1_\alpha$: `SELECT * FROM` $R1_\alpha$ `WHERE` $Gender =$ `'M' OR` $Level =$ $'L_2$`'`
$Q1_\beta$: `SELECT * FROM` $R1_\beta$ `WHERE (`$Name =$ `'Gender' AND` $Searchkey =$`'M') OR (`$Name =$ `'Level' AND` $Searchkey =$$'L_2$`')`

The reason for the specific structure of the q_β queries would be clear from the description of the verification process and the associated example.

RDAS1.Ψ: As discussed, Ψ is the transform executed in the server to generate the response for a set of queries produced by Φ. In RDAS1 the response of the server is constructed just by running the queries specified by Φ on R_α and R_β. We denote the response by $S = (S_\alpha, S_\beta)$ where S_α and S_β corresponds to responses of q_α and q_β respectively. Thus, for the example, the server executes the queries $Q1_\alpha$ and $Q1_\beta$ on $R1_\alpha$ and $R1_\beta$ respectively and thus returns the response $S1 = (S1_\alpha, S1_\beta)$ which is shown in Table 3.

Table 3. Left side: Answer $S1_\alpha$, Right side: Answer $S1_\beta$

Relation $S1_\alpha$

EmpId	Name	Gender	Level	Nonce	Tag
TRW	Tom	M	L_2	1	Y_1
JOH	John	M	L_2	3	Y_3
RZT	Rosy	F	L_2	6	Y_6

Relation $S1_\beta$

Name	SearchKey	RowNo	Tag1
Gender	M	8	Y_8'
Level	L_2	10	Y_{10}'

RDAS1.$\mathcal{V}$: The verification procedure receives as input the response $S = (S_\alpha, S_\beta)$ from the server, the original query and the keys. The response of the server consists of two parts. We denote these two parts as two sets S_α and S_β which are responses to the queries q_α and q_β respectively. Thus, S_α and S_β contains tuples from the relations R_α and R_β respectively.

The transformed queries q_α and q_β are also disjunctions of conditions, for a q_α query the conditions are of the form $a_i = v_i$, where a_i is an attribute and v_i its value, and for a q_β query the conditions are of the form `Name` $=$ v AND `SearchKey` $=$ `w`. Thus, for the description below, we consider that C_1^α `OR` C_2^α `OR` $\ldots$ C_l^α is a α query where each C_i^α is an equality condition and C_1^β `OR` C_2^β `OR` $\ldots$ C_l^β is a β query where each C_i^β is a conjunction of two equality conditions. Note that the number of conditions in q_α and q_β would always be the same. Let SaT be a predicate which takes as input a tuple t and a condition C (which can also be a query q) and outputs a 1 if the tuple t satisfies the condition C, otherwise outputs a zero. With these notations defined, we are ready to describe the verification algorithm. The verification algorithm consists of three procedures: α-Verify, makeBitMap and β-Verify. The procedures are shown in Figure 2, and they are applied sequentially in the same order as stated above.

The verification procedure checks for both the correctness and the completeness of the server response against the original query q. Note that the server response consists of two distinct parts S_α and S_β, the S_α part corresponds to

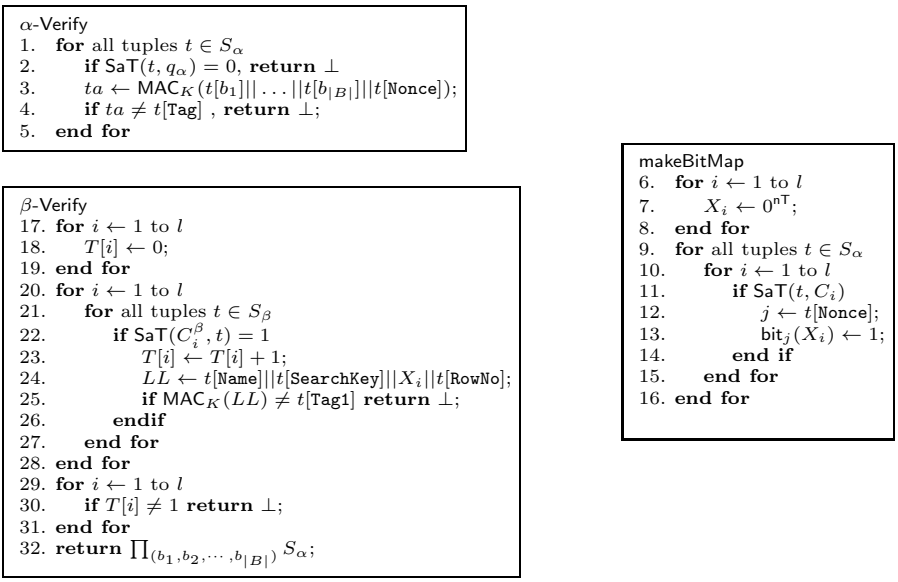

Fig. 2. The procedures involved in the verification process

the real result of the original query q and the S_β part assists the verification process to verify the completeness of the result in S_α. In the part α-Verify, the verification procedure checks for the correctness of the tuples returned by the server. As in the transformed relation R_α a message authentication code is associated with each tuple of the original relation, hence the α-Verify part of the verification procedure checks whether the contents of the tuples in S_α are not modified. If any of the the tuples in S_α are modified then the computed message authentication code on the tuple will not match the attribute `Tag`. If the computed value of tag does not match with the attribute `Tag` for any tuple then the verification process rejects by returning $\perp$. Moreover in line 2 it checks whether each tuple in S_α do satisfy the specified query. If the verification process does not terminate in the α-Verify phase then it means that the tuples in S_α are all valid tuples of the relation R_α and they all satisfy the specified query q_α. The other two parts of the verification process checks the completeness of the response.

Corresponding to each condition `Name` $= v$ AND `SearchKey` $=$ `w` in q_β the procedure makeBitMap constructs the corresponding bitmap $\mathsf{BitMap}_{R_\alpha}(v, w)$ using the server response S_α. Note that if the server response S_α is correct then makeBitMap would be able to construct the bitmaps corresponding to each condition in q_β correctly. This is possible due to the specific type of the allowed queries. Recall that an allowed query is formed only by the disjunctions of equality conditions. In the procedure corresponding to the l conditions in q_β, l bitmaps are constructed which are named $X_1, \ldots, X_l$ (See the example later for more explanation).

In the procedure β-Verify the response S_β is verified using the bitmaps $X_1, \ldots, X_l$ constructed before. The procedure β-Verify first verifies whether S_β contains tuples corresponding to each condition in q_β, this is done using the counter $T[i]$, where i runs over the conditions in q_β. Notice, that for every condition C_i^β the server must return only one tuple in S_β. The other parts of the procedure involves in verifying the tags of the tuples against the tag's of the computed bitmaps.

To make the exposition clearer let us consider the same example we have so far considered, i.e., the relation $R1$ the queries $Q1_\alpha$, $Q1_\beta$ and the corresponding server responses of $S1_\alpha$ and $S1_\beta$ (which are shown in Table 3). Given these responses the procedure α-Verify will not terminate, as all the tuples in $S1_\alpha$ do satisfy the conditions in $Q1_\alpha$ and as they are correct responses in the sense that they are just copies of the tuples present in the relation R_α, hence the corresponding message authentication codes will match. Given the responses in $S1_\alpha$, one can compute the bitmaps $\mathsf{BitMap}_{R_\alpha}(\texttt{Gender}, M)$ and $\mathsf{BitMap}_{R_\alpha}(\texttt{Level}, L_2)$. To see this, see the response $S1_\alpha$ in Table 3, where it says that the tuples satisfying the condition $\texttt{Gender}{=}M$ $\texttt{OR}$ $\texttt{Level}{=}L_2$ are the tuples with the nonce values 1, 3 and 6. Now, as the verification procedure has as input the whole of response $S1_\alpha$, hence it can predict correctly that the rows with the nonce value 1 and 3 satisfies the condition $\texttt{Gender}{=}M$ and all the tuples in $S1_\alpha$ (i.e., with nonce values 1, 3, 6) satisfies the condition $\texttt{Level}{=}L_2$. Thus, knowing that the total number of tuples in R_α to be 6, and assuming that server response is complete then the bitmap can be computed as $\mathsf{BitMap}_{R_\alpha}(\texttt{Gender}, M) = 101000$. Note that the 1st and 3rd bits of this bitmap are only one, as it corresponds to the response in $S1_\alpha$. Similarly one can compute $\mathsf{BitMap}_{R_\alpha}(\texttt{Level}, L_2) = 101001$. This is precisely what the procedure makeBitMaps would do for the example that we consider. The computation of the individual bitmaps $\mathsf{BitMap}_{R_\alpha}(\texttt{Gender}, M)$ and $\mathsf{BitMap}_{R_\alpha}(\texttt{Level}, L_2)$ are possible from $S1_\alpha$ as the $Q1_\alpha$ query is a disjunction of equality conditions, if in the contrary the query was a conjunction of conditions then there would be no way to compute the individual bitmaps in a straightforward way, this explains the reason for the query restriction that we impose.

Once these bitmaps are computed by using the procedure β-Verify one can verify the correctness of the response $S1_\beta$. As one can concatenate corresponding the bitmaps computed by the procedure makeBitMaps with the other attributes of the tuples in S_β and compute the tag using the message authentication code and thus verify if the computed tag matches the attribute `Tag1`.

The procedure β-Verify basically verifies the correctness of the response S_β, this verification is done by using the bitmaps constructed using the response S_α. The correctness of the response S_β implies the completeness of the response S_α.

4.1 Security of RDAS1

We can distinguish two possibilities for breaking RDAS1: infringe the correctness or violate the completeness of the response for a fixed query. To break the

correctness the opponent must make changes in one or more tuples of S_α and still pass the verification process. This implies that the adversary must forge the respective MACs. On the other hand, to violate the completeness, the adversary must change the respective bitmaps in S_β which also implies forging the respective MACs. Now, we introduce this notion in a formal way.

Theorem 1. *Consider an arbitrary adversary $\mathcal{A}$ attacking* RDAS1 *in the sense of definition 1. Let $\mathcal{A}$ choose a relation R with* nT *tuples and the relation be such that the transformed relation R_β contains n' tuples. Then there exist an adversary $\mathcal{B}$ attacking the message authentication code* MAC *such that*

$$\Pr[\mathsf{Succ}_{\mathcal{A}}] \leq \Pr[\mathcal{B} \textit{ forges }].$$

Also, B asks at most $\mathsf{nT} + n'$ queries to its oracle and runs for time $t_{\mathcal{A}} + (\mathsf{nT} + n')(c + t_{\mathsf{MAC}})$, where $t_{\mathcal{A}}$ is the running time of $\mathcal{A}$, t_{MAC} is the time for one MAC computation and c is a constant.

For space limitations we skip the proof, it would be presented in the full version, which would be published in the IACR eprint archive.

4.2 Costs and Overheads

Storage Cost: Given a relation $R(B)$ with nT tuples, let $\mathsf{size}(t_i[b])$ denote the size of the attribute b in the tuple t. Then the total size of R (which we also denote by $\mathsf{size}(R)$) would be given by

$$\mathsf{size}(R) = \sum_{i=1}^{\mathsf{nT}} \sum_{b \in B} \mathsf{size}(t_i[b]).$$

If this relation R is converted into (R_α, R_β) with RDAS1.F, then we would have,

$$\mathsf{size}(R_\alpha) = \mathsf{size}(R) + \sum_{i=1}^{\mathsf{nT}} (\mathsf{size}(t_i[\texttt{Nonce}]) + \mathsf{size}(t_i[\texttt{Tag}])),$$

if we assume a tag of constant length of τ bits then we would have

$$\mathsf{size}(R_\alpha) \leq \mathsf{size}(R) + \mathsf{nT}(\lg \mathsf{nT} + \tau).$$

Again considering the set of allowed attributes of R as $A = \{a_1, a_2, \ldots, a_m\}$, and $N = \sum_{i=1}^{m} \mathsf{Card}(a_i)$, we will have

$$\mathsf{size}(R_\beta) = \sum_{i=1}^{N} (\mathsf{size}(t_i[\texttt{Name}]) + t_i[\texttt{SearchKey}] + t_i[\texttt{RowNo}] + \mathsf{size}(t_i[\texttt{Tag1}])).$$

If we consider s_{Name} and s_{sk} the maximum size of the values of the attributes `Name` and `SearchKey`, then we would have

$$\mathsf{size}(R_\beta) \leq N(s_{\mathsf{Name}} + s_{\mathsf{sk}} + \lg(\mathsf{nT} + N) + \tau).$$

The total cost of storage at the server side would be $\mathsf{size}(R_\alpha) + \mathsf{size}(R_\beta)$, and at the client side would be $\lg(\mathsf{nT})$ as in the client we need to store the number of tuples in the original relation.

Communication Cost: Consider the query `SELECT * FROM` R_α `WHERE` $a_1 = v_1$ `OR` $a_2 = v_2$ `OR` ..... `OR` $a_l = v_l$, let the number of tuples satisfying the query be num. Let siz be the size of the response in a normal scenario without authentication. Then the maximum size of the server response in case of RDAS1 would be

$$\mathsf{siz}_{\mathsf{RD1}} = \mathsf{siz} + \mathsf{num} \times (\lg \mathsf{nT} + \tau) + l \times (s_{\mathsf{Name}} + s_{\mathsf{sk}} + \lg(\mathsf{nT} + N) + \tau), \quad (1)$$

where the first two terms corresponds to the S_α response and the remaining term counts for the S_β response.

5 Selects Involving Arbitrary Boolean Connectives

Here we propose an extension of RDAS1 which can support queries of the form

Q: `SELECT * FROM` R `WHERE` $(a_1 = v_1)\ \Delta_1\ (a_2 = v_2)\ \Delta_2$ ..... $\Delta_{l-1}\ (a_l = v_l)$,

where Δ_is are arbitrary Boolean connectives. An easy solution to this case would be to change RDAS1 to a new protocol RDAS2 along the following lines:

1. The relation R_β produced by RDAS2.$\mathcal{F}$ would contain explicit bitmaps corresponding to the attributes and the values. Specifically, the attributes present in R_β should be $\{\mathtt{Name}, \mathtt{SearchKey}, \mathtt{RowNo}, \mathtt{bitmap}, \mathtt{tag1}\}$. Thus, for creating the relation R_β we need to add a line $t_j^{R_\beta}[\mathtt{bitmap}] \leftarrow \mathsf{BitMap}_R(\mathcal{L}_j^1, \mathcal{L}_j^2)$ after line 5 in the procedure **Creating** R_β in Fig. 1.
2. The query translation procedure and the response procedure for RDAS2 remains same as that of RDAS1.
3. The response procedure also remains the same, i.e., the server just answers the q_α and q_β queries, but as the R_β relation now explicitly contains the bitmaps, hence the bitmaps would also be a part of the query.
4. For the verification procedure in RDAS2 it is not required to create the bitmaps any more, the client verifies the S_α response by the procedure α-Verify in Fig. 2, then it verifies the tags of the individual bitmaps returned in S_β and finally computes the result bitmap using the returned bitmap and checks if the result bitmap matches with the result returned.

We now state the storage and communication costs for RDAS2 following the notations in Section 4.2. The size of R_α in case of RDAS2 would be the same as in RDAS1, the size of R_β would be

$$\mathsf{size}(R_\beta) \leq N(s_{\mathsf{Name}} + s_{\mathsf{sk}} + \lg(\mathsf{nT} + N) + \tau + \mathsf{nT}).$$

The size of a server response in case of RDAS2 would be

$$\mathsf{siz}_{\mathsf{RD2}} = \mathsf{siz}_{\mathsf{RD1}} + l \times \mathsf{nT} \tag{2}$$

where $\mathsf{siz}_{\mathsf{RD1}}$ is the size of the response of RDAS1, as given in Eq. (1). In case of RDAS2, though we state that the bitmaps are to be explicitly stored in the relation R_β, but as most commercial data bases uses bitmaps indices for accelerating query processing, hence this may not amount to extra storage in some systems. Moreover bitmaps can be compressed, there has been substantial work on suitable encoding of bitmaps such that their sizes can be reduced and the Boolean operations be applied on the compressed bitmaps [1, 2]. Applying proper encoding of the bitmaps can drastically reduce both storage and communication costs. Details about this would appear in the full version of the paper.

6 Experimental Results

In this section we discuss some experimental results on the performance of RDAS1 and RDAS2. Both RDAS1 and RDAS2 can be implemented with any secure MAC, we chose PMAC instantiated with an AES with 128 bit key (we use the description in [17]). For implementation of AES we use the new Intel dedicated instructions for it.

All results were obtained by testing the implementation in a four-core i5-2400 Intel processor (3.1GHz) machine, with a Ubuntu 12.04.02 LTS operating system. We used PostgreSQL 9.1.9 for our database and used the gcc 4.7.3 compiler.

We used Census-Income data set [4] to test performance of our schemes. This data contains weighted census data extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau. The number of instances in the data set is 199523. The data contains 42 demographic and employment related variables, the sum of the cardinalities of all the attributes is 103419, and the total size of the dataset size is 99.1 MB.

The experiments were performed using the set of queries presented in Table 4 (a). Table 4 (a) shows the characteristics of the queries in terms of the number of restrictions and the size of the query response, all of them are disjunctions of equality conditions. The last column shows the percentage of the response size in terms of the whole database size. Note that the number of restrictions corresponds to the number of tuples which would be included in a correct and complete S_β response and the response size would be same as the number of tuples in the S_α result. In Table 4 (b) we report the time required for executing the set of queries in Table 4 (a). We report times for normal execution (i.e. without any authentication) and RDAS1 and RDAS2. All reported times are the average of 250 executions of the same query. The response sizes for the queries can be easily computed using equations (1) and (2). For concrete numerical values see the full version.

Table 4. Performance Information

(a) Summary of the different queries used for performance testing

Query Id	Number of Restrictions	Response Size (tuples)	Database Percentage
Q1	10	20115	10
Q2	20	35452	18
Q3	30	92791	46
Q4	40	106065	53
Q5	50	198869	99

(b) Execution times for OR queries. All times are in milliseconds.

		RDAS1		RDAS2	
Query Id	Normal time	Avg time	Extra Overhead(%)	Avg time	Extra Overhead(%)
Q1	680.93	829.33	21.79	827.06	21.46
Q2	1223.09	1652.09	47.10	1516.33	35.01
Q3	2784.97	4076.28	46.36	3604.09	29.41
Q4	3192.06	4582.93	43.58	4004.43	25.45
Q5	6130.07	10781.05	75.87	9222.51	50.45

7 Conclusion

We presented RDAS a generic framework for authenticated query processing and provided the syntax and security definition of a RDAS. We also provided two concrete constructions RDAS1 and RDAS2 which uses bitmap indices and message authentication codes in a novel way. There are other ways in which RDAS1 and RDAS2 can be improved, for example communication costs can be drastically reduced using aggregate message authentication codes. These possibilities would be discussed in the full version of the paper.

Acknowledgements. The authors acknowledge the support from CONACYT project 166763.

References

1. Chan, C.Y., Ioannidis, Y.E.: Bitmap index design and evaluation. In: Haas, L.M., Tiwary, A. (eds.) SIGMOD Conference, pp. 355–366. ACM Press (1998)
2. Chan, C.Y., Ioannidis, Y.E.: An efficient bitmap encoding scheme for selection queries. In: Delis, A., Faloutsos, C., Ghandeharizadeh, S. (eds.) SIGMOD Conference, pp. 215–226. ACM Press (1999)
3. Devanbu, P.T., Gertz, M., Martel, C.U., Stubblebine, S.G.: Authentic data publication over the internet. Journal of Computer Security 11(3), 291–314 (2003)
4. Frank, A., Asuncion, A.: UCI machine learning repository (2010)
5. Goodrich, M.T., Tamassia, R., Triandopoulos, N.: Super-efficient verification of dynamic outsourced databases. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 407–424. Springer, Heidelberg (2008)
6. Hacigümüs, H., Mehrotra, S., Iyer, B.R.: Providing database as a service. In: ICDE, p. 29. IEEE Computer Society (2002)
7. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Chaudhuri, S., Hristidis, V., Polyzotis, N. (eds.) SIGMOD Conference, pp. 121–132. ACM (2006)

8. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
9. Mouratidis, K., Sacharidis, D., Pang, H.: Partially materialized digest scheme: an efficient verification method for outsourced databases. VLDB J. 18(1), 363–381 (2009)
10. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. In: NDSS, The Internet Society (2004)
11. Narasimha, M., Tsudik, G.: DSAC: integrity for outsourced databases with signature aggregation and chaining. In: Herzog, O., Schek, H.-J., Fuhr, N., Chowdhury, A., Teiken, W. (eds.) CIKM, pp. 235–236. ACM (2005)
12. Narasimha, M., Tsudik, G.: Authentication of outsourced databases using signature aggregation and chaining. In: Li Lee, M., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882, pp. 420–436. Springer, Heidelberg (2006)
13. Nuckolls, G.: Verified query results from hybrid authentication trees. In: Jajodia, S., Wijesekera, D. (eds.) Data and Applications Security 2005. LNCS, vol. 3654, pp. 84–98. Springer, Heidelberg (2005)
14. Palazzi, B., Pizzonia, M., Pucacco, S.: Query racing: Fast completeness certification of query results. In: Foresti, S., Jajodia, S. (eds.) Data and Applications Security and Privacy XXIV. LNCS, vol. 6166, pp. 177–192. Springer, Heidelberg (2010)
15. Pang, H., Jain, A., Ramamritham, K., Tan, K.-L.: Verifying completeness of relational query results in data publishing. In: Özcan, F. (ed.) SIGMOD Conference, pp. 407–418. ACM (2005)
16. Pang, H., Zhang, J., Mouratidis, K.: Scalable verification for outsourced dynamic databases. PVLDB 2(1), 802–813 (2009)
17. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer, Heidelberg (2004)
18. Wrembel, R., Koncilia, C.: Data warehouses and OLAP: concepts, architectures, and solutions. Gale virtual reference library. IRM Press (2007)
19. Yang, Y., Papadopoulos, S., Papadias, D., Kollios, G.: Spatial outsourcing for location-based services. In: Alonso, G., Blakeley, J.A., Chen, A.L.P. (eds.) ICDE, pp. 1082–1091. IEEE (2008)
20. Zheng, Q., Xu, S., Ateniese, G.: Efficient query integrity for outsourced dynamic databases. IACR Cryptology ePrint Archive, 2012:493 (2012)