

A Data Masking Technique for Data Warehouses

Ricardo Jorge Santos
CISUC – DEI – FCTUC
University of Coimbra
Coimbra, Portugal

lionsoftware.ricardo@gmail.com

Jorge Bernardino
CISUC – DEIS – ISEC
Polytechnic Institute of Coimbra
Coimbra, Portugal

jorge@isec.pt

Marco Vieira
CISUC – DEI – FCTUC
University of Coimbra
Coimbra, Portugal

mvieira@dei.uc.pt

ABSTRACT

Data Warehouses (DWs) are the enterprise's most valuable asset in what concerns critical business information, making them an appealing target for attackers. Packaged database encryption solutions are considered the best solution to protect sensitive data. However, given the volume of data typically processed by DW queries, the existing encryption solutions heavily increase storage space and introduce very large overheads in query response time, due to decryption costs. In many cases, this performance degradation makes encryption unfeasible for use in DWs. In this paper we propose a transparent data masking solution for numerical values in DWs based on the mathematical modulus operator, which can be used without changing user application and DBMS source code. Our solution provides strong data security while introducing small overheads in both storage space and database performance. Several experimental evaluations using the TPC-H decision support benchmark and a real-world DW are included. The results show the overall efficiency of our proposal, demonstrating that it is a valid alternative to existing standard encryption routines for enforcing data confidentiality in DWs.

Keywords

Data security, Data confidentiality, Data privacy, Encryption, Data masking, Data warehousing.

1. INTRODUCTION

Data confidentiality focuses on protecting data from unauthorized disclosure. Currently, data is a major asset for any enterprise, not only for knowing the past, but also for aiding today's business or predicting future trends [6, 15]. Given its decision support nature, Data Warehouses (DWs) translate data into business knowledge, providing information for adding business value. Consequently, DWs are the core of enterprise sensitive data. Unfortunately, this makes them a major target for attackers [29]. Consequently, efficiently securing sensitive data has become an imperative concern in many enterprises [17, 29].

Data used for analyzing business performance is mostly stored in specific attributes, called facts. Facts are stored in fact tables,

which typically take up at least 90% of DW storage space [14]. To protect those attributes, data masking actions (also called data obfuscation, *i.e.*, changing data values so that their real values are not known) and encryption techniques are widely used to enforce the confidentiality of data.

Encryption at the database level has proved to be the best method to protect sensitive data and deliver performance [16, 23]. However, since DWs are usually huge, with millions or billions of rows in their fact tables, and user queries are typically *ad hoc* and access large amounts of data, encryption overheads are a major concern [4]. Although encryption solutions are efficient in their security purpose, they introduce several key costs:

- Extra storage space of encrypted data, which is usually a considerable overhead, given the typically large storage space size occupied by DW databases;
- Time and resources needed for encrypting sensitive data; and
- Overhead in query response time and allocated resources for decrypting data to process queries, probably the most significant drawback for using encryption in DWs.

As demonstrated throughout our paper, these overheads imply that the standard encryption algorithms provided by current DBMS dramatically degrade database performance, a critical issue in data warehousing. This ultimately jeopardizes their usefulness. Thus, developing a data masking/encryption strategy for DWs must balance between the requirements for security and desire for high performance [10, 16, 19, 27].

In this paper, we describe and analyze the currently available masking and encryption techniques at the database level. We thoroughly discuss the issues involving their use in what concerns database performance and requirements from a data warehousing perspective. We show that they are extremely inefficient in DW performance, introducing huge query response time overheads for many queries.

To ensure their security strength, standard encryption algorithms such as AES[2] and DES[8] are complex routines requiring high computational efforts and focus on ensuring security regardless from performance issues. In this paper we propose a masking technique for DW data based on the MOD-modulus operation (which returns the remainder of a division expression) and simple arithmetic operations, which balances strong data security with database performance. The proposed solution is not to be regarded as an alternative to standard encryption in terms of security strength, but rather as an efficient alternate data confidentiality solution in what concerns the tradeoff between performance and security. Our proposal aims for masking data while introducing low computational efforts, focusing on balancing security strength with performance to make it feasible for use in DWs.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IDEAS11 2011, September 21-23, Lisbon [Portugal]

Editors: Bernardino, Cruz, Desai

Copyright ©2011 ACM 978-1-4503-0627-0/11/09 \$10.00

The proposed technique uses three masking keys for each masked value. These keys are also encrypted and stored in what we call a “black box” file, placed in the operating system directories and file structures of the database server machine disk(s), as explained in section 4. All SQL commands and actions required by users are also encrypted and stored in this black box, in a log which can be audited by any user with database, enterprise administration or management privileges. This allows super users to watch over each other, being able to detect malicious actions.

The proposed solution is transparent, not requiring changes to the code of both DBMS and user applications. Its usage is based on rewriting user queries. The data processed in the database is encrypted at all times, never allowing breaches before the user queries are processed. Only the final processed results are returned to the authorized user applications that required them. This also allows using the database (or creating instant replicas) for testing purposes in software application development, *i.e.*, for production purposes, since the masked data is realistic but not real. Moreover, it also allows protecting the data against attackers that gain direct access to the database server.

As shown in the experimental results, our solution significantly decreases both data storage space and processing overheads, both in inserting and querying data from the DW, when compared with standard encryption algorithms like AES and 3DES, provided by major Database Management Systems (DBMS), such as Oracle and SQL Server, for nearly all queries in all tested scenarios. The experiments show that our technique’s overall results make it a valid alternative to those standard solutions.

The remainder of this paper is organized as follows. In section 2, we present related work, describing and analyzing the issues in state-of-the-art masking and encryption solutions for providing data privacy. Section 3 presents the packaged encryption routines provided by Oracle Database 11g and points out the main issues involved in their use for DWs. In section 4 we present and describe our solution. Section 5 presents experimental evaluations using the well-known TPC-H decision support benchmark and a real-world DW. Finally, section 6 concludes the paper by summarizing the main findings of our research.

2. RELATED WORK ANALYSIS

Web-based applications require supporting cooperative processes while ensuring the confidentiality of data. This research area is characterized by a number of different approaches and techniques, including privacy-preserving data mining [26], privacy-preserving information retrieval [28], and database systems specifically tailored toward enforcing privacy [3]. A selective encryption model is proposed in [28], for data access control purposes.

A light-weighted database encryption scheme with low decryption overhead in column-oriented DBMS is proposed in [10]. They claim their solution is as secure as any underlying block cipher, while demonstrating the inherent insecurity of any order preserving encryption scheme, such as [4], under straightforward attack scenarios. However, their experimental evaluations show an overhead of at least 50% in query response time to retrieve the encrypted tuples, which is a very large cost for DW queries.

An Order Preserving Encryption Scheme for numeric data is proposed in [4], by flattening and transforming the plain text distribution onto a target distribution of value-based buckets. This solution allows any comparison operation to be directly applied

on encrypted data, such as equality and range queries, as well as MAX, MIN and COUNT queries. However, storage space overhead depends on the skew of the plaintext and target distributions, which can be a big problem in DWs. The mapping function for the buckets introduces a greater overhead than the technique proposed in our paper, and the definition of how the bucket distribution should be built and how it should scale is not a trivial task. A similar type of solution for processing queries without decrypting data was proposed earlier by [11], suffering from the same problems. This last solution uses only one encryption key to encrypt data, which reduces the number of hypothesis the attacker needs to consider in order to be successful.

The work in [5] presents data perturbation techniques for preserving privacy. They propose implementing perturbed tables and explain data reconstruction for responding to queries, in a DW environment. Although providing strong guarantees against privacy breaching, perturbation methods produce errors in data reconstruction, which we pretend to avoid.

The Oracle Enterprise Manager Data Masking Pack [21] enables replacing sensitive data with realist-looking values based on masking rules. It provides out-of-the-box mask primitives for various types of data, such as random numbers, random digits, random dates, and constants. However, the process is irreversible, *i.e.*, it is not possible to retrieve the original true values, which makes it useless for DWs. Oracle recognizes this and recommends using the Data Masking Pack mainly for production databases, where it can be used as an easy, efficient and fast solution to mask real data and transform it into realistic but false data, to be integrated in the development lifecycle of user applications [21].

The work in [23] proposes a security middleware that acts as a wrapper/interface between user applications and the encrypted database server, for ensuring data integrity and efficient query execution. They evaluate queries at the application server and retrieve only the required rows from the server. They use only one TPC-H query for measuring the database server costs of their solution. The results show those costs rise by a factor proportional to the size of the tested data subset, which in their experiments is extremely small (it ranges from 10MB to 50MB, where query execution time rises up to 5 times for the last). This is not a realistic dataset for data warehousing scenarios, since the network bandwidth in communication costs with the wrapper is much smaller than in a typical DW scenario, where the data size would be equal or bigger than 1GB. The fact that they only test one query is also somewhat inconclusive, due to the large scope of possibilities in building and executing decision support queries.

The Data Encryption Standard (DES) became the first encryption standard in 1977 [8]. DES is a 64 bit block cipher and uses a 56 bit encryption key. This has implications in short data lengths. Even 8 bit data, when encrypted by the algorithm will always result in a 64 bit chunk. At that time, this encryption standard suffered many attacks and methods that demonstrated it is an insecure block cipher [13]. There has considerable controversy over its design, particularly in the choice of a 56 bit key [19].

As an enhancement of DES, the Triple DES (3DES) standard [1] was proposed. In this algorithm, the encryption method is similar to original DES, but applied three times to increase the encryption level, using three different 56 bit keys. Thus, the effective key length is 168 bits. Since the algorithm increases the number of cryptographic operations to execute, it is a well known fact that the 3DES algorithm is one of the slowest block cipher methods.

The Advanced Encryption Standard (AES) is the most recent encryption standard, proposed to replace DES algorithms [2]. The AES block ciphers have a significant increase in the block size (from the old standard of 64 bits up to 128 bits). AES provides three approved key lengths: 128, 192 and 256 bits. AES is considered fast and able to provide stronger encryption than other encryption algorithms such as DES [19, 20, 23]. Brute force attack is the only known effective attack known against it.

The Blowfish encryption algorithm [24] is a public domain algorithm. Blowfish is a variable length key, 64 bit block cipher. This algorithm was first introduced in 1993. Though it suffers from the weak keys problem, no attack is known to be successful against it [19].

The work in [19] implemented the DES, 3DES, AES and Blowfish algorithms and conducted several experiments to compare their performance. This study demonstrated the Blowfish algorithm was the fastest algorithm. However, it is a public domain solution and not an encryption standard, reason why major DBMS such as Oracle, MySQL and Microsoft SQL Server do not provide it with their database servers. Regarding the open encryption standards, the AES was the best solution, both in execution time and throughput. On the other hand, 3DES presented the worst performance.

3. PACKAGED DBMS ENCRYPTION – THE ORACLE TDE

In the recent past, Oracle has integrated standard data encryption routines within their DBMS [12, 20, 22]. The Oracle Transparent Data Encryption (TDE) solution was introduced in Oracle Database 10g Release 2. TDE enables transparently applying encryption within the database avoiding expensive changes to application source code, including database triggers and views. Data is transparently encrypted when written to disk and transparently decrypted after a user application has been successfully authenticated. The TDE allows the user to choose from various standard algorithms, such as AES (with 128, 192 and 256 bit keys), and 3DES. Oracle does not allow to plug-in other encryption algorithms within the DBMS core.

3.1 How the Oracle TDE works

Oracle TDE uses a two tier encryption key architecture, consisting of a master key and one or more table and/or tablespace keys. The table and tablespace keys are encrypted using the master key. Key management is done by creating an Oracle Wallet for each case. The Oracle Wallet is an encrypted container, physically a specific folder in the directory tree of the hard disk, used to store authentication and signing credentials, including passwords, the TDE master key, PKI tablespace and table private keys, and certificates needed by SSL/TLS for data communication and access purposes. If a wallet is damaged or missing, or if the user is not authorized to open it, no encrypted data linked to that wallet can be accessed. This implies that any authorized backup of the encrypted data should also include backing up its respective wallet. File and directory permissions should be defined by the DW manager for determining who is allowed access to the wallet directory, avoiding its disclosure.

Oracle TDE allows two types of encryption: tablespace encryption (where all data stored in the tablespace is encrypted) or column encryption, for encrypting specific table columns. Since the proposal in this paper is a column-based solution, we

shall compare it with column-based encryption. Moreover, Oracle recommends that column encryption should be preferred when it is easy to determine which columns are sensitive and which are not, or when a small number of well defined columns are sensitive [12, 22], which is typically what happens in DWs [14].

When using column encryption, a storage space overhead between 1 and 52 bytes per encrypted value is added. The generation of independently encrypted values for the same column is done by using an explicit option, implying 16 bytes of storage space overhead. If that option is not used, those extra 16 bytes are saved, but all encrypted values in the column rely on one key only in the encryption algorithm, which lowers the privacy level. TDE does not support encrypting columns with foreign key constraints, due to the fact that individual tables have their own unique encryption key. However, joining tables is transparent and allowed to users and applications, even if the columns for the join condition are encrypted.

3.2 Testing Oracle TDE for DW scenarios

Given the assumption that encryption has proved to be the best method to protect sensitive data and deliver performance [16, 23], in order to evaluate its use in data warehousing scenarios, we have performed an experimental evaluation of column-data encryption solutions provided by Oracle 11g TDE, using the well known TPC-H benchmark [25]. In these tests, we measured performance impact on a workload composed of all the benchmark's queries that access the fact table *Lineitem*, on its 1GB scale database. We used a Pentium 2.8GHz CPU, with 2GB of RAM, where 512MB were dedicated for use by Oracle database memory area (SGA), on a 1.5TByte SATA hard disk.

For fairness, the database was optimized in a standard best practice manner for all scenarios (including primary keys, foreign keys, referential integrity constraints, and bitmap join indexes). Response times for each TPC-H query, shown in Table 1, are an average obtained from six executions, for each scenario. Three scenarios were defined: 1) without using encrypted data (Standard Query Exec. Time); 2) against numerical columns encrypted with AES 128 bit (Query Exec. Time Using AES128); and 3) with 3DES (Query Exec. Time Using 3DES168). Before each execution, the database server was restarted.

We chose the AES128 and 3DES168 algorithms for the tests because they are, respectively, the simplest (and fastest) and most complex (and slowest) of the available algorithms, according to Oracle [12, 22]. This is consistent with what we discussed in section 2, given that AES is the algorithm which requires less computational resources, while 3DES requires the most.

Although Oracle argues using TDE will only increase response time between 5% and 10%, on average, [22], the results clearly show that this is not true for the tested scenarios. The results in Table 1 show that response time overhead is, on average, much higher than 10%. In fact, all overheads are much greater than 10%, registering 171% or 185% for the whole workload (last table line), depending on which encryption algorithm is used. Moreover, the individual query execution time overhead for more than a half of the queries registered more than 100% for both encryption algorithms. This also happens with higher scales of the benchmark database, as shown in section 5 of this paper, as well as in a real-world sales DW, which we also used for the experimental evaluation of our proposal.

Table 1. Oracle 11g standard query response time vs. TDE column-encryption using 1GByte TPC-H DB (in seconds)

Queries	Standard Query Exec. Time	Query Exec. Time Using AES128	% Overhead Using AES128	Query Exec. Time Using 3DES168	% Overhead Using 3DES168
Q1	11	904	8118%	977	8782%
Q3	10	23	130%	24	140%
Q5	10	23	130%	25	150%
Q6	8	30	275%	32	300%
Q7	10	24	140%	24	140%
Q8	312	373	20%	377	21%
Q9	127	192	51%	197	55%
Q10	10	23	130%	23	130%
Q12	10	22	120%	24	140%
Q14	8	24	200%	25	213%
Q15	14	21	50%	22	57%
Q17	38	52	37%	54	42%
Q18	49	184	276%	191	290%
Q19	90	121	34%	127	41%
Q20	105	184	75%	188	79%
TOTALS	812	2200	171%	2310	185%

4. MOBAT: A MODULUS-BASED TECHNIQUE FOR DATA MASKING IN DW

As shown in the previous section, the query response time overheads are extremely large, jeopardizing their usefulness in DW scenarios. A data encryption algorithm is not of much use if it is secure enough to assure data confidentiality but too slow to be acceptable in terms of performance [19]. Thus, a privacy technique that can minimize the number of required operations could make the difference between acceptable and unacceptable performance overhead. In this sense, this paper aims to provide a specific feasible column masking solution for ensuring strong privacy in DWs, while registering an acceptable measure of performance degradation and storage space overhead.

4.1 The MOBAT Data Security Architecture

The proposed architecture for the system, which can be seen in Figure 1, is comprised of three entities: 1) the masked database and its DBMS; 2) the MOBAT Security Application (MOBAT-SA); and 3) user/client applications that query data in the masked database. The MOBAT-SA acts as a middleware between the masked database DBMS and the user applications, using secure SSL/TLS connections, ensuring that the queried data is securely processed and proper results are returned to those applications.

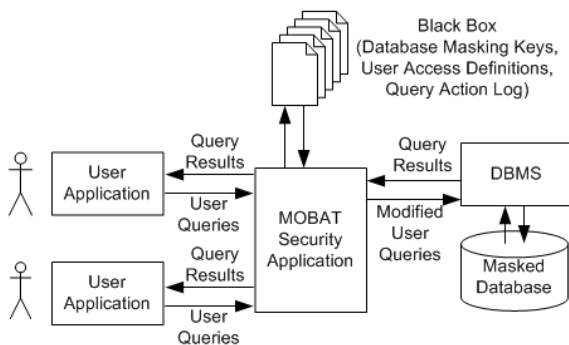


Figure 1. The MOBAT Data Security Architecture

Our solution is based on a formula that depends on three masking keys; two are private (*i.e.*, not available to any DW user, including DBAs) and one is public. The private keys for each

table and column to mask that are generated for our method are encrypted and stored in the “black box”. The black box works like the Oracle Wallet, explained in the previous sections, and is hidden in the directories and file structure of the operating system of the database server.

Technically, if a DBA or enterprise manager is allowed to control security without any restriction (which may happen in the Oracle TDE solution, for instance), the whole system becomes vulnerable to malicious DBA/manager actions. To manage this, our black box can never be manually accessed or updated by anyone, except the MOBAT-SA itself. All submitted logins and queries to the MOBAT-SA are stored and encrypted in the black box as a read-only action log. All actions executed on the DW can be controlled and audited by whoever has administration rights, *e.g.* DBA’s, CEO’s, managers, etc, so anyone can watch over anyone to check for misuse. If an attacker manages to bypass the MOBAT-SA, s/he is able to directly access the database server; however, since all sensitive stored data is masked, s/he will never be able to access its true values.

In typical topologies with encryption middleware solutions, such as [23], when a user application requests data, after ensuring the request is authorized, the security application retrieves encrypted data from the database and executes the decryption process. To do this, it must send the data over the network to be decrypted. However, once decrypted, we have clear-text information that needs to be sent back over the wire to the database server. This requires re-securing the information in transit, typically through secure communication processes such as SSL/TLS. When the data arrives at the agent on the database server, it has to be returned to clear-text, and then it is served up to the calling application.

This type of topology has shown to be a poor solution. Since all the data to decrypt needs to pass through the encryption agent, the whole process gets strangled due to bandwidth consumption between itself and the database, jeopardizing data throughput and consequently, query response time. In these cases, the network roundtrips for data in DWs would become an unbearable cost, making this kind of solution unfeasible. Therefore, proposals involving encrypting and decrypting data at the database server are better solutions, because they eliminate network and computational overheads from the critical path.

Our method relies on query rewriting, avoiding expensive invocation of user defined functions for masking or retrieving true data. Only the user queries are rewritten by MOBAT-SA and sent to the DBMS to be processed, sending the results back to the user application that requested their execution, instead of retrieving encrypted data for processing them. This avoids overflowing network bandwidth in the critical path, improving response time and throughput when compared with other solutions. Communication between the MOBAT-SA and the DBMS are made through secure SSL/TLS connections for protecting the rewritten queries and their results. The MOBAT-SA also disables the DBMS history log, ensuring that the rewritten query instruction is not kept in the database history, since it passes on the values of the masking keys. After the query results are returned, the MOBAT-SA restores the history log status.

4.2 The MOBAT Data Masking Technique

Most facts in DWs are columns with numerical values [14]. MOBAT aims on masking the DW's numerical values, in a way that makes it difficult for any unauthorized user to discover their original values, while introducing minimum overhead in the necessary operations needed to retrieve the original values for query processing. We aim to ensure that sensitive data is replaced by realistic but not real data.

Suppose a table T with a set of N numerical columns $C_i = \{C_1, C_2, C_3, \dots, C_N\}$ to be masked and a total set of M rows $R_j = \{R_1, R_2, R_3, \dots, R_M\}$. Each value to mask in the table will be identified as a pair (R_j, C_i) , where R_j and C_i respectively represent the row and column to which the value refers. The masking formula for each value of T depends on the following predefinitions:

- K_1 and K_2 are private keys stored in the black box, thus, accessible only by the MOBAT-SA;
- K_1 is a 128 bit integer random generated value between 1 and 2^{127} , constant for table T ;
- K_2 is a random generated value between 1 and the maximum positive integer value of column C_i , given the maximum storage size of C_i . There is one K_2 for each column C_i , represented by $K_{2,i}$;
- K_3 is a public key based on a 128 bit integer column appended to each row R_j in T , filled in with a random value between 1 and 2^{127} , represented by $K_{3,j}$.

Each new masked value $(R_j, C_i)'$ is obtained by applying the following formula (1) for row j and column i of table T :

$$(R_j, C_i)' = (R_j, C_i) - ((K_{3,j} \text{ MOD } K_1) \text{ MOD } K_{2,i}) + K_{2,i} \quad (1)$$

Since K_1 and $K_{2,i}$ are constants for the table and each column, respectively, and $K_{3,j}$ is also a constant, stored with each row in the table, the formula (2) for retrieving the original value is:

$$(R_j, C_i) = (R_j, C_i)' + ((K_{3,j} \text{ MOD } K_1) \text{ MOD } K_{2,i}) - K_{2,i} \quad (2)$$

This technique implies that the values of $K_{3,j}$ must be stored along with each row j in table T . If the values of $K_{3,j}$ were to be stored in a lookup table separate from table T , a heavy join operation between them is required, whenever there is a need to unmask data. Given the typical immense number of rows in fact tables, this should be avoided at all cost. Thus, to avoid table joins in query processing when using MOBAT, there are two possible solutions for including $K_{3,j}$ in each row of table T :

- 1) A MODIFY TABLE ... ADD COLUMN is done for creating $K_{3,j}$ as a new column in table T ; or
- 2) Table T is rebuilt with inclusion of $K_{3,j}$ in the CREATE TABLE statement before restoring the data.

The second option will imply a certain effort and amount of time, depending on table T 's size, in order to rebuild it. However, it should reveal a gain in query response time, since the new column $K_{3,j}$ is physically included in each row from the start, instead of appended to the database after it has already been filled in, which makes it physically stored apart from the previous existing data.

A third option for defining $K_{3,j}$ values and increasing performance is to use any long integer typed column C_Z , which is already part of the original data structure of table T , as $K_{3,j}$, instead of creating an extra column for $K_{3,j}$ in T . In this case, there is no need to change the data structure of T , avoiding storage space overhead in T . However, this somewhat limits the strength of the masking formula, since the value of $K_{3,j}$ also depends on the range and cardinality of the values of C_Z , and the predictability of knowing the values of C_Z on behalf of an attacker.

4.3 Apparent Randomness and Security

Generating randomness for cryptographic or masking applications is a costly and security-critical operation [7]. There is a need to guarantee that the generated values for masking the real data is not deductible between them. Therefore, two same original real data values must generically originate different masking generated values, so a minimum level of apparent randomness is ensured. Given that our masking formula (1) uses MOD operations in conjunction with randomly generated realistic values, the generated masked values for the same original data values are mostly different. Thus, the referred minimal level of apparent randomness is assured in the new masked values, which allows achieving acceptable security strength in this sense.

Our masking formula (1) is based on the use of two consecutive MOD operations. Since the MOD function has no inverse function [10], there is no way of deducing an inverse function of (1) without applying our reverse masking function (2) to retrieve the original data values (except for trying a brute force attack). To demonstrate this, suppose a table T with two masked columns, *Column1* and *Column2*. Suppose the MOBAT-SA generated the values $K_1=7432$ for table T and $K_{2,1}=34$ and $K_{2,2}=17251$ for each column. Table 2 shows the original data for T on the left and its resulting masked content on the right, using the masking formula (1). In Table 2, it can be seen that the same original values of *Column1* result in different masked values *Column1'*, achieving the referred apparent randomness. Therefore, the only way to bypass our security solution is to obtain the values of the private masking keys K_1 and K_2 .

Table 2. Example of an original and resulting masked dataset using MOBAT

T – Original dataset			T' – Masked dataset		
Column1	Column2	$K_{3,j}$	Column1'	Column2'	$K_{3,j}$
12	1873	817721	15	108923	817721
23	4129	936154	43	104226	936154
12	1624	61437	37	86894	61437
12	8824	94725	13	50534	94725
23	4624	497624	51	94763	497624

Given encryption or masking key lengths, a brute force attack is done by generating the total number of possible combinations for testing the algorithm, given a certain input. Taking the minimum 128 bit length key for AES, the total number of possible combinations for this algorithm is 2^{128} . Therefore, the average number of tests that attackers need to execute for discovering each encryption key is, roughly, half that number, *i.e.*, 2^{127} .

In our proposal, since K_3 is public, only K_1 and K_2 need to be discovered. K_1 is a 128 bit integer key. K_2 is also a 128 bit integer, which depends on the maximum integer storage size defined for each column, and is therefore variable between 1 and 128 bits. This means that our technique implies a minimum number of 2^{129} key combinations, for K_1 and K_2 together (at least 128 bits + 1 bit), and roughly needs an average number of 2^{128} tests for discovering the keys using brute force, for each masked column in the table, since K_2 is column dependant. Thus, the average number of combinations to discover all the needed key values for an i number of columns is $i * 2^{128}$ brute force tests. Periodically, the values of all or any one of the K_1 , K_2 , and K_3 keys may be refreshed and rebuild the masked table values, in order to ensure data is properly protected. Although it is not possible to prove that a particular algorithm is secure [9, 19], we believe that our technique is secure enough to be considered as acceptable for use.

4.4 Implementing the Masked Database

To mask a database, a DBA must require this action through the MOBAT-SA. Entering the DBA login and database connection data, this application will attempt to login to the respective database. If it succeeds, MOBAT-SA will scan all relevant data access policy definitions in the database, for identifying authorized users and respective permissions. The black box will then be updated with the user access definitions for that database.

After the previous step, the MOBAT-SA will ask the DBA which tables and columns are to be masked. After confirmation of this information, the database is masked. At this point, the DBA will have to define how the $K_{3,j}$ masking keys are to be created for each table (which has to be one of the three options mentioned in the explanation of the MOBAT masking technique): 1) an extra added column to each table to mask, without rebuilding them; 2) an added column to the structure of each table, rebuilding it entirely; or 3) using one of the numerical columns of the table.

According to the chosen options for each table, the $K_{3,j}$ masking keys will be generated and stored in the j rows of each respective table to which it refers, and all K_1 and $K_{2,i}$ masking keys for each table and column will also be generated, encrypted and stored in the respective black box. All key values are generated according to what we have explained in section 4.3. Finally, the MOBAT-SA will apply the data masking formula (1) on all rows of all columns which are to be masked, updating their value with the new masked value. Whenever the database needs to be updated with the insertion of new data or the modification or deletion of existing data, this should be performed by the MOBAT-SA, which will apply the masking routine to any value which refers to any masked column, storing the masked value directly in place.

4.5 Querying the Masked Database

When client applications request the execution of a query, they submit it to the MOBAT-SA, instead of directly querying the database. The MOBAT-SA then rewrites the query using the formula (2) to replace the respective masked columns used in the

query, checking the user access definitions in the black box to see if it comes from an authorized user. To rewrite the query, the MOBAT-SA searches for the tables and columns it needs to process, and looks up the security black box for retrieving the K_1 and $K_{2,i}$ data masking keys for those tables and columns, respectively, as well as the names of the needed $K_{3,j}$ key fields to be used by the MOBAT-SA in those tables.

As an example, suppose the *LineItem* table of the TPC-H benchmark has four numerical fact columns ($I = 4$) (*L_Quantity*, *L_ExtendedPrice*, *L_Tax* and *L_Discount*) masked by MOBAT. Suppose also that MOBAT has generated and filled in a new column *L_KeyK3* for the j rows of the *LineItem* table, which will act as the $K_{3,j}$ key values for our method, and has stored the value of 9342 (for example) for key K_1 referring to the *LineItem* table, as well as $K_{2,L_Quantity} = 12$, $K_{2,L_ExtendedPrice} = 51234$, $K_{2,L_Tax} = 6$, and $K_{2,L_Discount} = 4$ (for example also). Consider TPC-H query Q6 with the masked fields, for showing the total ordered quantity, income, discounts and tax value of each customer order:

```
SELECT SUM(L_ExtendedPrice * L_Discount) AS
      Total_Revenue
FROM LineItem
WHERE L_ShipDate>=TO_DATE('1994-01-01',
      'YYYY-MM-DD') AND
      L_ShipDate<TO_DATE('1995-01-01',
      'YYYY-MM-DD') AND
      L_Discount BETWEEN 0.05 AND 0.07 AND
      L_Quantity<24
```

The new query, rewritten by the security application and submitted to the DBMS server, would be:

```
SELECT
      SUM((L_ExtendedPrice+MOD(MOD(L_KeyK3,9342),
      51234)-51234) *
      (L_Discount+MOD(MOD(L_KeyK3,9342),
      4)-4)) AS Total_Revenue
FROM LineItem
WHERE L_ShipDate>=TO_DATE('1994-01-01',
      'YYYY-MM-DD') AND
      L_ShipDate<TO_DATE('1995-01-01',
      'YYYY-MM-DD') AND
      (L_Discount+MOD(MOD(L_KeyK3,9342),4)-4)
      BETWEEN 0.05 AND 0.07 AND
      (L_Quantity+MOD(MOD(L_KeyK3,9342),12)-12)<24
```

The changes to the user queries are handled transparently by the MOBAT-SA and kept hidden from the users. Only the query results are passed back to the users after they have been processed. The only change user applications need is to query the MOBAT-SA, instead of the database. This makes MOBAT a transparent solution and also addresses the trend towards embedding business logic within a DBMS through the use of stored procedures and triggers [16]. On the other hand, the masked database may be used for testing user software development, allowing direct queries, since the data is masked but the data schemas maintain all their original definitions. A direct query on the masked database will be harmless, producing realistic results, but with different values from the real ones.

5. EXPERIMENTAL EVALUATION

To evaluate our proposal, we used the 1GB and 10GB scale sizes of the TPC-H decision support benchmark, and a real-world sales DW storing one year of commercial data. To build the DWs, we used the Oracle 11g DBMS, on a Pentium CPU with 2GB RAM and 1.5TB SATA hard disk. 512MB were dedicated for Oracle memory cache (SGA). All results shown in this section were

obtained under the exact same conditions as referred in section 3 of this paper (see subsection 3.2). The data schema of TPC-H is a database with one fact table (*LineItem*), and seven dimension tables attached to it. The data schema of the real-world sales DW is a database with one fact table (*Sales*) and four dimension tables attached to it. The storage size and number of rows for each TPC-H table and sales DW are shown in Tables 3 and 4, respectively.

Table 3. TPC-H decision support benchmark table sizes

Table	1GB TPC-H Database		10GB TPC-H Database	
	Nr. of Rows	Table Size	Nr. of Rows	Table Size
LineItem	6.001.215	740MB	59.986.052	7.386MB
Orders	1.500.000	152MB	15.000.000	1.520MB
Customers	150.000	32MB	1.500.000	320MB
Suppliers	10.000	4MB	100.000	40MB
Part	200.000	32MB	2.000.000	312MB
PartSupp	800.000	112MB	8.000.000	1.120MB
Nation	25	<1MB	25	<1MB
Region	5	<1MB	5	<1MB
TOTALS	8.661.245	1.072MB	86.586.082	10.968MB

Table 4. Sales DW table sizes

Table	Sales DW Database	
	Nr. Of Rows	Table Size
Sales	31.536.124	1.927MB
Products	50.109	7MB
Customers	251.514	90MB
Promotions	89.812	10MB
Time	8.760	<1MB
TOTALS	31.936.319	2.034MB

In the TPC-H setups, four columns of the *LineItem* fact table were chosen for masking (*L_Quantity*, *L_ExtendedPrice*, *L_Tax* and *L_Discount*), given that they are the numerical fact columns used in the benchmark queries. In the Sales DW, five numerical columns were chosen (*S_ShipToCost*, *S_Tax*, *S_Quantity*, *S_SalesAmount*, and *S_Profit*), for the same reasons.

For the TPC-H workload we used the benchmark queries 1, 3, 6, 7, 8, 10, 12, 14, 15, 17, 19, and 20, which access the masked table *LineItem*. For the Sales DW, the workload was composed by a set

of 29 queries, all processing the *Sales* fact table, representing a sample of typical decision support queries. All data schemas, queries and results may be consulted at [18]. The results show each query's average response time (with standard deviations between [0.52, 54.65] for 1GB TPC-H, between [0.64, 70.10] for 10GB TPC-H, and between [0.57, 71.20] for the Sales DW). The experimental scenarios are shown in Table 5.

Table 6 shows data loading time and storage space results for each scenario. Without masking or encrypting, the storage size of the *LineItem* fact table for the 1GB TPC-H measured 772MB, taking up 310 seconds to perform a complete load using the Oracle SQL*Loader. As seen in Table 6, with MOBAT the storage size grows from 0% (when using column *L_OrderKey* as the masking key $K_{3,j}$) to a maximum of 5.7%, while loading time overhead respectively ranges from 0% (for the same scenario) to 7.7%. When using the Oracle TDE column encryption scenarios, both storage space and loading time overheads are much greater (at least 100% more storage space and loading time overheads).

Figure 2 shows each individual TPC-H query execution time result for the 10GByte scale database, while Figure 3 show query execution time overheads (in percentage) referring to each data encryption/masking scenario, for the same database. Figures 4 and 5 show the query response time and respective overheads, for the Sales DW database. The results for the 1GByte TPC-H database where similar to those of the 10GByte TPC-H database.

Observing Figures 2 to 5, almost all MOBAT results are better than those obtained by the AES128 and 3DES168 Oracle TDE encryption algorithms, for all the TPC-H and real-world Sales DW setups. MOBAT response time overheads in TPC-H are smaller than 10% for almost all the queries, except Q1 and Q17, while the data encryption scenarios show overheads mainly around 100%, except for Q1 (where they reach 10000%), Q17, Q19 and Q20. In the results concerning the 2GByte Sales DW, shown in Figures 4 and 5, MOBAT results are also much better, where queries Q1, Q2, Q3, Q5, Q7 and Q9 registered almost no significant query response time overheads at all, while with the encryption algorithms AES128 and 3DES168, 26 out of all 29 queries presented an overhead equal or greater than 100%.

Table 5. Experimental Data Encryption/Masking Scenarios







Reference	Graphic	Description
Standard		Unencrypted/Unmasked data
AES128 Column		Data encrypted with Oracle TDE AES128 column
3DES168 Column		Data encrypted with Oracle TDE 3DES168 column
MOBAT AddCol		Data masked by MOBAT formula (1), where masking key column $K_{3,j}$ has been added to the fact table
MOBAT CreateCol		Data masked by MOBAT formula (1), where masking key column $K_{3,j}$ was added to the fact table, which has been completely recreated
MOBAT ColKey		Data masked by MOBAT formula (1), using a numerical column from the original fact table data structure as key $K_{3,j}$

Table 6. TPC-H 1GB LineItem storage size and data loading

	Using Oracle TDE Encryption		Using MOBAT		
	AES128 Column	3DES168 Column	MOBAT AddCol	MOBAT CreateCol	MOBAT ColKey
Storage Space	1960MB	1572MB	816MB	804MB	772MB
Overhead	153,9 %	103,6 %	5,7 %	4,1 %	0 %
Loading Time(sec)	898s	904s	334s	321s	310s
Overhead	189,7 %	191,6 %	7,7 %	3,5 %	0 %

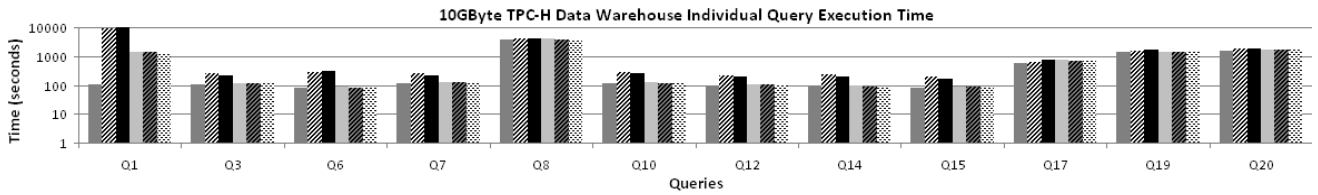


Figure 2. 10GB TPC-H Data Warehouse Query Execution Times

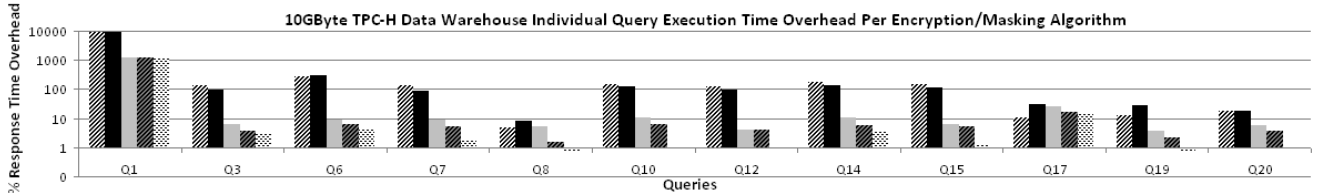


Figure 3. 10GB TPC-H Data Warehouse Query Execution Time Overheads per Encryption/Masking Algorithm

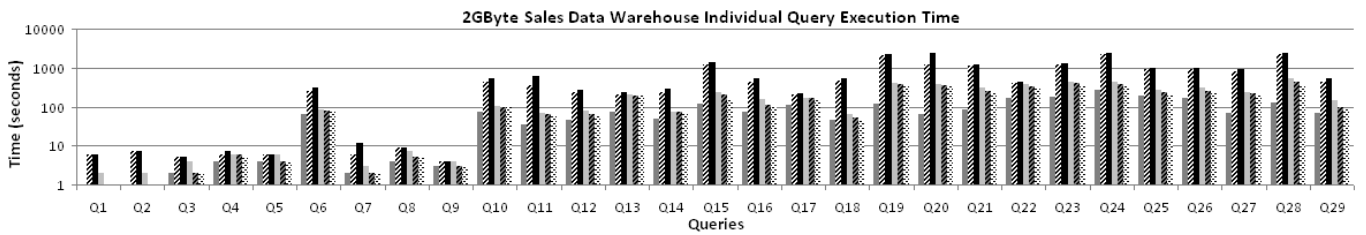


Figure 4. Sales Data Warehouse Query Execution Times

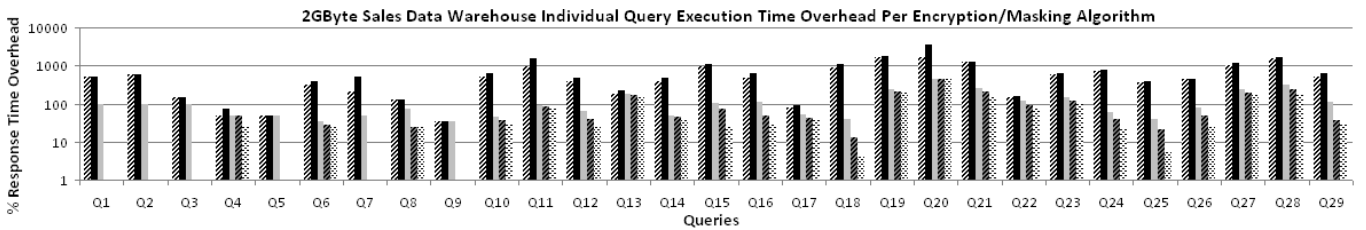


Figure 5. Sales Data Warehouse Query Execution Time Overheads per Encryption/Masking Algorithm

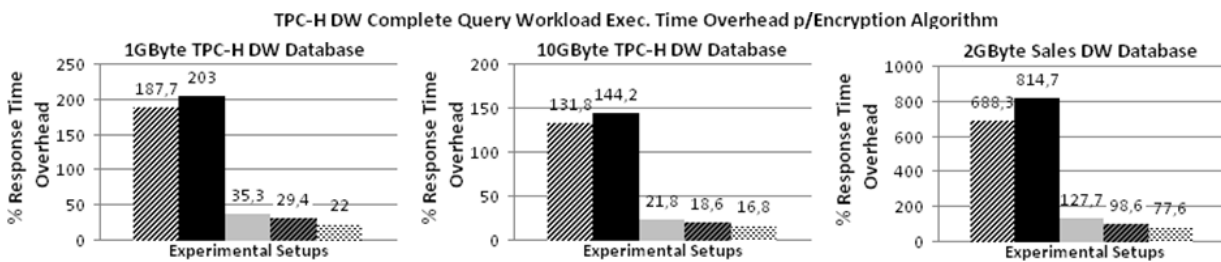


Figure 6. Complete Query Workload Execution Time Overheads per Encryption/Masking Algorithm

Figure 6 shows the overheads for the whole workload of each experimental setup. Comparing the overheads introduced by each technique, they confirm that MOBAT is much better than both AES128 and 3DES168 Oracle Column TDE, given the complete query workload in each setup. MOBAT ranges from at least 5.32 (187.7/35.3) times better than the standard encryption solutions for the 1GB TPC-H database, to 9.23 (203/22) times better. In the 10GB TPC-H database, the gains range from 6.05 (131.8/21.8) to 8.58 (144.2/16.8) times better, and for the Sales DW, from 5.39 (688.3/127.7) to 10.5 (814.7/77.6) times better. All results also show that the performance of CreateCol Masking is better than AddCol Masking, which was expected, as we mentioned in section 4.3, when explaining the MOBAT technique. The performance results of ColKey Masking experiments are the best

for MOBAT, since there was no need for creating an extra column in the fact tables, which allowed maintaining fact table storage size and to avoid retrieving an extra column for processing the queries, when compared to AddCol Masking and CreateCol Masking.

6. CONCLUSIONS AND FUTURE WORK

In this paper we discussed the existing solutions for data privacy and the issues involving their use in DWs. We demonstrated that the introduced storage space and performance overheads can make their use unfeasible from a data warehousing perspective. This leads us to state that the existing techniques are too complex to be used in DW scenarios.

Given that business data in DWs mainly consist on numerical values, we have proposed a data masking technique for numerical values. The solution is based on a masking formula with two modulus (division remainder) and two simple arithmetic operations. The formula introduces low computational efforts and, consequently, relatively small overheads in query response time, while providing a considerable level of security, given the length of the masking keys and non-invertible properties of the modulus operation. It also requires a very small overhead in storage space, compared with other column-based data privacy solutions.

By simply rewriting user queries, it avoids network bandwidth overflow. Stored data is masked at all times, which allows using the masked database for testing software production; directly querying the database will retrieve realistic data, but never real data. This also avoids access to real original data if any attacker is able to bypass access control and retrieve data directly from the database. MOBAT also allows authorized database and enterprise managers and administrators to audit the actions of everyone else and each other, by being able to lookup the masking actions and SQL commands history executed by the MOBAT-SA.

Experimental evaluations show the overheads produced by our proposal in both appending and consulting data are lower than those of standard encryption algorithms such as AES and 3DES, provided by major DBMS. The results show that our technique is a more efficient overall solution, making it a valid alternative for protecting DW numerical data.

As future work, we will develop our technique to accomplish masking alphanumeric values also, to provide a complete data protection solution. We will also strive to prove and improve its security strength without jeopardizing database performance. We also intend to take advantage of the inclusion of the extra column in the masked tables for detecting malicious or incorrect data changes in each row, broadening the scope of our solution to embrace data integrity issues.

7. REFERENCES

- [1] 3DES, Triple DES, National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS) Pub. 800-67, ISO/IEC 18033-3, 2005.
- [2] AES, "Advanced Encryption Standard", National Inst. of Standards and Technology (NIST), FIPS-197, 2001.
- [3] R. Agarwal, J. Kiernan, R. Srikant, and Y. Xu, "Hippocratic Databases", Int. Conf. Very Large DataBases (VLDB), 2002.
- [4] R. Agarwal, J. Kiernan, R. Srikant, and Y. Xu, "Order-Preserving Encryption for Numeric Data", ACM SIG Conf. on Management Of Data (SIGMOD), 2004.
- [5] R. Agrawal, R. Srikant, and D. Thomas, "Privacy Preserving OLAP", ACM SIG Conf. Management Of Data (SIGMOD), 2005.
- [6] H. Baer, "On-Time Data Warehousing with Oracle Database 10g – Information at the Speed of Your Business", Oracle Whitepaper, Oracle Corporation, 2004.
- [7] M. Barbosa and P. Farshim, "Randomness Reuse: Extensions and Improvements", Inst. Mathematics and its Applications (IMA) Int. Conference on Cryptography and Coding, 2009.
- [8] DES, Data Encryption Standard, National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS) Publication 46, 1977.
- [9] N. Ferguson, "AES-CBC + Elephant Diffuser – A Disk Encryption Algorithm for Windows Vista", Microsoft Corp. Whitepaper, 2006.
- [10] T. Ge and S. Zdonik, "Fast, Secure Encryption for Indexing in a Column-Oriented DBMS", Int. Conf. Data Engineering (ICDE), 2007.
- [11] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra, "Executing SQL over Encrypted Data in the Database-Service-Provider Model", ACM SIG International Conference on Management Of Data (SIGMOD), 2002.
- [12] P. Huey, "Oracle Database Security Guide 11g", Oracle Corp., 2008.
- [13] J. Kim, Y. Lee, and S. Lee, "DES with any reduced masked rounds is not secure against side-channel attacks", Elsevier Int. Journal Computers and Mathematics with Applications, 60, 2010, www.elsevier.com/locate/camwa
- [14] R. Kimball and M. Ross, "The Data Warehouse Toolkit", 2nd Edition, Wiley & Sons, Inc., 2002.
- [15] J. Kobiulus, "The Forrester Wave: Enterprise Data Warehousing Platforms", Forrester Research Report, 2009.
- [16] U. T. Mattson, "Database Encryption – How to Balance Security with Performance", Protegrity Corporation Technical Paper, 2004.
- [17] J. McKendrick, "IOUG Data Security 2009: Budget Pressure Lead to Increased Risks", The Independent Oracle Users Group (IOUG) Security Report, 2009.
- [18] MOBAT Testing Queries, available at <http://213.13.123.56/MOBAT/queries.html>
- [19] A. Nadeem and M. Y. Javed, "A Performance Comparison of Data Encryption Algorithms", IEEE Int. Conference on Inform. and Communication Technologies (ICICT), 2005.
- [20] Oracle Corporation, "Security and the Data Warehouse", Oracle White Paper, 2005.
- [21] Oracle Corporation, "Data Masking Best Practices", Oracle White Paper, 2010.
- [22] Oracle Corporation, "Oracle Advanced Security Transparent Data Encryption Best Practices", Oracle White Paper, 2010.
- [23] V. Radha and N. H. Kumar, "EISA – An Enterprise Application Security Solution for Databases", Int. Conf. on Information Systems Security (ICISS), S. Jajodia and C. Mazumdar (Eds), Springer LNCS 3803, 2005.
- [24] B. Schneier, "Description of a New Variable-Length Key, Block Cipher (Blowfish), Fast Software Encryption", Cambridge Security Workshop, 1994.
- [25] Transaction Processing Council, "The TPC Decision Support Benchmark H", <http://www.tpc.org/tpch/default.asp>
- [26] J. Vaidya and C. Clifton, "Privacy Preserving Association Rule Mining in Vertically Partitioned Data", ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002.
- [27] M. Vieira and H. Madeira, "Towards a Security Benchmark for Database Management Systems", Int. Conference on Dependable Systems and Networks (DSN), 2005.
- [28] S. C. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Over-encryption: Management of Access Control Evolution on Outsourced Data", Int. Conf. on Very Large DataBases (VLDB), 2007.
- [29] N. Yuhanna, "Your Enterprise Database Security Strategy 2010", Forrester Research, September 2009.