# Handbook of Research on Computational Intelligence for Engineering, Science, and Business

Siddhartha Bhattacharyya
*RCC Institute of Information Technology, India*

Paramartha Dutta
*Visva Bharati University, India*

**Volume I**

Information Science
REFERENCE

Managing Director: Lindsay Johnston
Editorial Director: Joel Gamon
Book Production Manager: Jennifer Romanchak
Publishing Systems Analyst: Adrienne Freeland
Development Editor: Austin DeMarco
Assistant Acquisitions Editor: Kayla Wolfe
Typesetter: Lisandro Gonzalez
Cover Design: Nick Newcomer

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com



Library of Congress Cataloging-in-Publication Data

Handbook of research on computational intelligence for engineering, science, and business / Siddhartha Bhattacharyya and Paramartha Dutta, editors.
pages cm
Includes bibliographical references and index.
Summary: “This book discusses the computation intelligence approaches, initiatives and applications in the engineering, science and business fields, highlighting that computational intelligence as no longer limited to computing-related disciplines and can be applied to any effort which handles complex and meaningful information”-- Provided by publisher.
ISBN 978-1-4666-2518-1 (hardcover) -- ISBN (invalid) 978-1-4666-2519-8 (ebook) -- ISBN (invalid) 978-1-4666-2520-4 (print & perpetual access) 1. Computational intelligence. 2. Content analysis (Communication) I. Bhattacharyya, Siddhartha, 1975- II. Dutta, Paramartha.
Q342.H36 2013
006.3--dc23

2012027413

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

# Chapter 15
# Using Data Masking for Balancing Security and Performance in Data Warehousing

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

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

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

## ABSTRACT

*Data Warehouses (DWs) are the core of sensitive business information, which makes them an appealing target. Encryption solutions are accepted as the best way to ensure strong security in data confidentiality while keeping high database performance. However, this work shows that they introduce massive storage space and performance overheads to a magnitude that makes them unfeasible for DWs. This work proposes a data masking technique for protecting sensitive business data in DWs which balances security strength with database performance, using a formula based on the mathematical modular operator and simple arithmetic operations. The proposed solution provides apparent randomness in the generation and distribution of the masked values, while introducing small storage space and query execution time overheads. It also enables a false data injection method for misleading attackers and increasing the overall security strength. It can be easily implemented in any DataBase Management System (DBMS) and transparently used, without changes to application source code. Experimental evaluations using a real-world DW and TPC-H decision support benchmark implemented in leading DBMS Oracle 11g and Microsoft SQL Server 2008 demonstrate its overall effectiveness. Results show the substantial savings of its implementation costs when compared with state of the art encryption solutions provided by those DBMS and that it outperforms those solutions in both data querying and insertion of new data.*

DOI: 10.4018/978-1-4666-2518-1.ch015

## 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 (Baer, 2004; Kobielus, 2009). Given its decision support nature, the data in Data Warehouses (DWs) translates into business knowledge, providing invaluable decision making information for adding business value. Consequently, DWs are the core of the enterprise's sensitive data. Unfortunately, this makes them a main target for both inside and outside attackers (Yuhanna, 2009). Consequently, several studies have demonstrated that efficiently securing sensitive data has become an imperative concern in many enterprises (McKendrick, 2009; Yuhanna, 2009).

To protect information in databases, data masking actions and encryption techniques are widely used. Data masking routines are mainly simpler than encryption routines, but provide lower security strength. Moreover, data masking routines provided by most commercial tools typically change data in an irreversible manner, *i.e.*, after masking data it not possible to subsequently retrieve the original true values, making them useless for real live DW databases. This has made masking solutions the main choice for protecting published data or production data, instead of real-live databases (Bertino & Sandhu, 2005; Huey, 2008; Natan, 2005; Oracle, 2010a; Ravikumar et al., 2011; Gartner, 2009).

Published research and best practice guides state that encryption is the best method to protect sensitive data at the database level while maintaining high database performance (Agrawal et al., 2004; Ge & Zdonik, 2007; Huey, 2008; Natan, 2005; Oracle, 2010a, 2010b; Procopiuc & Srivastava, 2011; Vimercati et al., 2007; Hacigumus et al., 2004). However, since DWs are usually huge in size, with millions or billions of rows in their fact tables, and user queries are typically *ad hoc* and access large amounts of data, encryption and decryption overhead is a major concern (Agrawal et al., 2004). General encryption algorithms are mainly built taking under consideration desirable security strength. Although efficient in their security purpose, most encryption solutions introduce several key costs from the DW perspective:

- Large processing time/resources for encrypting sensitive data, given routine or hardware access in very large databases such as those in DWs;
- Extra storage space of encrypted data. Since DWs usually have many millions or billions of rows, even a small modification of any column size to accommodate encrypted output introduces large storage space overheads; and
- Overhead query response time and allocated resources for decrypting data to process queries. Given the huge amount of data typically accessed by DW queries, this is probably the most significant drawback for using encryption in DWs.

In this work, we present the commonly available encryption techniques for databases, thoroughly discussing the issues involving the use of these techniques, in what concerns database performance and requirements, from the data warehousing perspective. As demonstrated throughout this work, the introduced overheads imply that the standard encryption algorithms currently provided by DBMS dramatically degrade database performance, which is a critical issue in data warehousing. This ultimately jeopardizes their applicability in DWs. Consequently, developing a data masking/encryption strategy for DWs must balance between the requirements for security and desire for high performance (Ge & Zdonik, 2007; Mattson, 2004; Nadeem & Javed, 2005; Vieira & Madeira, 2005).

Data used for analyzing business performance is mostly stored in specific attributes, called facts. Facts are stored in fact tables, which typically take up 90% or more of storage space in DWs (Kimball, 2002). This work proposes 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. Its main contribution is a solution specifically tailored for DWs, which is able to produce better overall security-performance tradeoffs than most available solutions such as encryption standards, *i.e.*, that presents an acceptable and significant level of security strength while introducing small storage space and response time overheads, such as presented in (Santos et al., 2011).

The proposed solution in this work 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 setups where the tradeoff between performance and security needs to be revised. To ensure their security strength, standard encryption algorithms such as AES (AES, 2001) and DES (3DES, 2005; DES, 1977) are complex routines requiring high computational efforts and focusing almost solely on ensuring security regardless from performance issues, while the approach in this work aims for securely masking data while introducing low computational efforts, to make it feasible for use in DWs.

The proposed technique uses three masking keys for calculating each masked value. These keys are also encrypted and stored in a vault that can be seen as a "black box", which will be placed in the database server machine with a purpose similar to the Oracle Wallet (Huey, 2008), as explained further on. 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 or enterprise administration or management privileges. This enables super users to watch over each other and provides a data source that can be subsequently used for solutions able to audit and analyze user actions, such as database intrusion detection systems.

The proposed solution is transparent and does not require changing both DBMS and user applications code. Its usage is based on rewriting user queries in order to process them with the true data values. Contrarily to similar middleware solutions that pre-fetch encrypted data to process decryption locally, by simply rewriting queries network bandwidth congestion is avoided. The data processed in the database is encrypted at all times, protecting the data against attackers that gain direct access to the database server and 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. It is also a broad-scoped security solution, which besides enabling data masking, can also inject false data to increase the DW's overall security strength.

As can be seen in the experimental results, the proposed 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 DBMS such as Oracle and SQL Server, for nearly all queries in all tested scenarios. The experiments show that the proposed technique's overall results make it a valid alternative to those standard solutions.

## BACKGROUND AND RELATED WORK

### Standard Data Encryption Algorithms

Typical encryption algorithms include executing bit shifting and exclusive Or (XOR) operations within a predefined number of rounds. These operations rely on a key, which influences the "data mix" output of each round. There are mainly two types of encryption techniques: Block Ciphers and Stream Ciphers. A block cipher is a type of symmetric-key encryption algorithm that transforms a fixed-length block of *plaintext* (unencrypted text) data into a block of *ciphertext* (encrypted text) data of the same length, under the action of a user-provided secret key. Decryption is performed by applying the reverse transformation to the ciphertext block using the same secret key. Stream ciphers take a string (the encryption key) and deterministically generate a set of random-seeming text (called *keystream*) from that key. That keystream is then XORed against the message to encipher. To decipher the text, the recipient hands the same key to the stream cipher to produce an identical keystream and XORs it with the ciphertext, thus retrieving the original message.

The Data Encryption Standard (DES) became the first encryption standard in 1977 (DES, 1977). DES is a 64 bit block cipher and uses a 56 bit encryption key, which means it produces 64 bit outputs. This has implications in short data lengths. Even 8 bit data, when encrypted by the algorithm, will always result in a 64 bit chunk. This encryption standard suffered many attacks and methods that demonstrated it is an insecure block cipher (Kim et al., 2010). There has considerable controversy over its design, particularly in the choice of a 56 bit key (Nadeem & Javed, 2005). Moreover, given the computational power which exists today, performing an exhaustive key search for breaking a 56 bit key is a perfectly feasible task, which makes DES an unsecure cipher.

As an enhancement of DES, the Triple DES (3DES) encryption standard (3DES, 2005) was proposed. In this algorithm, the encryption method is similar to the original DES algorithm, but it is applied three times to increase the encryption level, using three different 56 bit keys. Thus, the effective key length is 168 bits. This makes 3DES a cipher capable of providing much stronger security than DES. However, since the algorithm increases the number of cryptographic operations it needs 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) was proposed to replace DES based algorithms (AES, 2001). The AES algorithms are the latest generation of block ciphers, and 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, compared to other encryption algorithms, such as DES and 3DES (Nadeem & Javed, 2005; Oracle, 2005; Radha & Kumar, 2005). Brute force attack is the only known effective attack known against it.

The Blowfish encryption algorithm (Schneier, 1994) is a public domain encryption algorithm, provided by Counterpane Systems, a consulting firm specialized in cryptography and computer security. Blowfish is a variable length key, 64 bit block cipher. Though it suffers from the weak keys problem, no attack is known to be successful against it (Nadeem & Javed, 2005).

The work in (Nadeem & Javed, 2005) implemented the DES, 3DES, AES and Blowfish algorithms and executed 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 vendors such as Oracle, MySQL and Microsoft SQL Server do not provide it with their database servers. Regarding the open encryption standards,

AES was the best solution, both in execution time and throughput. On the other hand, 3DES presented the worst performance.

## Other Solutions Based on Modifying Data for Enforcing Data Confidentiality

Data masking solutions are mainly used for generating test databases for software development environments, named production databases, or for camouflaging data values in publicly available published data (Huey, 2008; Oracle, 2005, 2010a; Ravikuhmar et al., 2011). An extensive survey on data masking techniques and their purpose is given in (Ravikuhmar et al., 2011). Many organizations have strived to solve confidentiality issues with hand-crafted solutions within the enterprise to solve the problem of sharing sensitive information. The most common solution is probably to use scripts with triggers in order to mask and unmask each value, or to embed the masking/unmasking logic within the user applications themselves. Many commercial data masking packages have also been developed, such as the Oracle Data Masking (ODM) pack (Natan, 2005; Oracle, 2010a).

The Oracle EM Data Masking Pack (DMP) (Oracle, 2010a) provides masking primitives to replace sensitive data with realistic-looking values, such as random numbers, random digits, random dates, constants, as well as built-in masking routines which shuffle the value in a column across different rows. However, the DMP masking process is irreversible, which means that is only intended for replacing true data with false data without being able to retrieve the true data after masking, making it useless for real DW users. Oracle recommends using the DMP mainly for testing databases, as an easy, efficient and fast solution in the development lifecycle of user applications (Oracle, 2010A). This is typically what happens with other standard commercial database masking solutions (Ravikuhmar et al., 2011). Since data masking techniques are often considered as lacking strong security strength and most rely on cross-referencing actions in order to retrieve the true original values (which bring huge table joins in DW databases), they are considered inadequate for use with real-world live databases such as DWs.

Web-based applications require supporting cooperative processes while ensuring confidentiality of data. This research field is characterized by a number of different approaches and techniques, including privacy-preserving data mining (Vaidya & Clifton, 2002), privacy-preserving information retrieval (Vimercati et al., 2010), and database systems specifically tailored toward enforcing privacy (Agrawal et al., 2002). A selective encryption model is proposed in (Vimercati et al., 2010), for access control.

An Order Preserving Encryption Scheme for numeric data is proposed in (Agrawal et al., 2004), by flattening and transforming plain text distribution onto a target distribution, based on 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 in the plaintext and target distributions, which can be a problem in DWs. The mapping function for the buckets introduces a greater overhead than the MOBAT technique, 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 (Hacigumus et al., 2002), 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 break security.

A light-weighted database encryption scheme with low decryption overhead in column-oriented DBMS is proposed in (Ge & Zdonik, 2007). 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 (Agrawal et al., 2004), under basic attack scenarios. However, their experimental evaluations show an overhead of at least 50% in response time to retrieve the encrypted tuples, a very large cost for DW queries.

The work in (Radha & Kumar, 2005) 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 over encrypted databases, evaluating queries at the application server and retrieving 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 that is 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 DWs. In a DW environment, previously transporting all the required data from the database to the middleware is unreasonable, since the amount of data accessed for processing decision support queries is typically much larger than a few tens of MB. This would strangle the network due to bandwidth consumption of data roundtrips between middleware and database, jeopardizing data throughput and consequently, response time. Thus, all encrypted data should be processed at the DBMS itself, eliminating network overhead from the critical path. 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.

Recently, research has proposed non-deterministic methods for masking data, such as perturbation techniques (Agrawal et al., 2005; Procopiuc & Srivastava, 2011; Xiao et al., 2009). The work in (Agrawal et al., 2005) proposes a solution based on data perturbation techniques and explains data reconstruction for responding to queries, in a DW environment. Recent similar work proposed data anonymization solutions relying on perturbation or differential techniques such as (Procopiuc & Srivastava, 2011) and (Xiao et al., 2009). Although providing strong guarantees against privacy breaching, perturbation methods produce errors in data reconstruction, which is not acceptable in most data warehousing environments.

Data injection has been mostly used for building synthetic datasets for benchmarking and production purposes, *i.e.*, filling in databases for testing the development of databases and applications (Arora et al., 2006; Callot et al., 2009; Lo et al., 2010). To our knowledge, there are no data injection solutions for enforcing data confidentiality as proposed in our technique.

## Analyzing Packaged DBMS Encrypted Solutions: The Oracle 11g TDE

In the recent past, Oracle has integrated standard data encryption routines within their DBMS (Huey, 2008; Oracle, 2005, 2010b). 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 kernel.

Oracle TDE uses a two tier encryption key architecture, consisting of a master key and one or more table and/or tablespace keys. These keys are encrypted using the master key. Key management is accomplished 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, which is used to store

authentication and signing credentials, including passwords, the TDE master key, 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 and masked 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 work is a column-based solution, it is compared 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 (Huey, 2008; Oracle, 2010b), which is typically what happens in DWs (Kimball & Ross, 2002).

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 (named *SALT*) that will use an independent key for generating each ciphertext, implying 16 bytes of storage space overhead. If this option is not used, those extra 16 bytes are saved but all encrypted values in the column rely on one key only, which lowers the security strength. 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.

In order to evaluate its use and performance impact in data warehousing scenarios, experimental evaluations of column-data encryption solutions provided by Oracle 11g TDE were executed, using the well known TPC-H benchmark (TPC, 2011). In these tests, the response time overhead for a workload composed of all the benchmark's queries that access the fact table *LineItem*, on its 1GB scale database, was measured, using 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. Before each execution, the database server was restarted. 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). The standard deviations for each scenario range within [0.56, 52.27], [0.73, 66.34] and [0.69, 67.89], respectively. The AES128 and 3DES168 algorithms were chosen for the tests because they are, respectively, the simplest (and fastest) and most complex (and slowest) of the set of available algorithms, according to Oracle (Huey, 2008; Oracle, 2010b). This is consistent with what was discussed in background and related work section, 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, (Oracle, 2010B), 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

*Table 1. Oracle 11g TPC-H 1GByte standard vs. column-encryption query response time (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%** |

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.

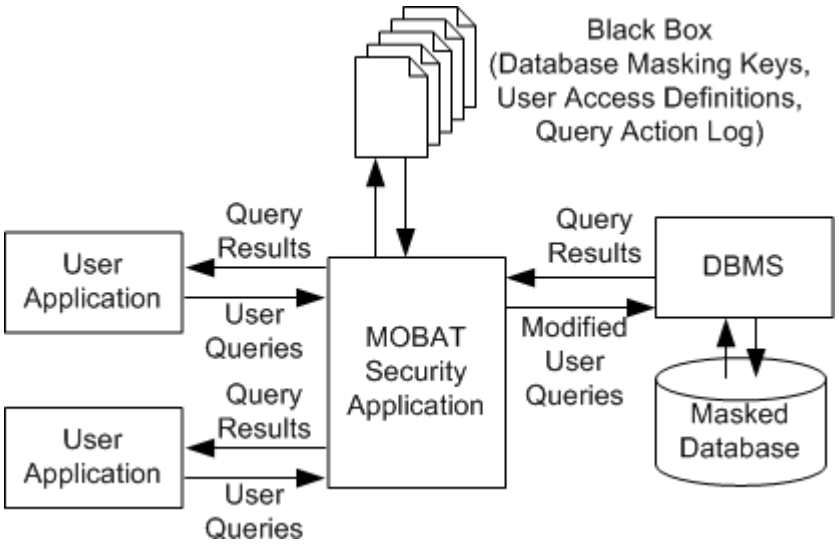
## MOBAT: A MODULUS-BASED DATA MASKING TECHNIQUE FOR BALANCING SECURITY AND PERFORMANCE IN DATA WAREHOUSES

### The Data Masking Solution's Functional Architecture

The system's functional architecture is shown in Figure 1, comprised by three entities: 1) the masked database and its DBMS; 2) the MOBAT security application (MOBAT-SA); and 3) user/client applications to query the masked database. The MOBAT-SA acts as a middleware broker between the masked database DBMS and those applications, ensuring the queried data is securely processed and proper results are returned to those applications. All communications are made through SSL/TLS secure connections, to protect SQL instructions and returned results between the system's entities, avoiding problems from intercepting messages by attackers on the network.

The Black Box is a set of files in a directory of the database server, created for each database to be masked and managed by the MOBAT-SA. This process is similar to the creation of an Oracle Wallet, which keeps all the encryption keys and definitions for each Oracle Database (Huey, 2008). However, contrarily to what happens in Oracle, where the DBA is free to access the Oracle Wallet whenever s/he wishes, only the MOBAT-SA can access the Black Box, *i.e.*, absolutely no user has direct access to its content. In the Black Box, the MOBAT-SA will store all the generated masking keys and predefined data access policies for the database to which it con-

*Figure 1. The data masking solution functional architecture*




cerns. The MOBAT-SA will also create a history log for recording a duplicate of all the further instructions and actions executed in the database, for auditing and control purposes. All Black Box contents are encrypted, using the AES standard encryption algorithm with a 256 bit key. In case of losing the Black Box of a certain database, there is no way to restore its true data, except to crack the masking keys.

The masking keys' security depends on where they are stored and who has access to them. MOBAT uses three masking keys; two are private and one is public. The private masking keys are generated by MOBAT-SA, and encrypted and stored by it in the Black Box. The values of those keys are never shown or known by the DBA or any other user. To obtain true results, all user queries or actions must pass through MOBAT-SA, which will store a copy of those instructions in the Black Box history log. Each time a user requests the execution of any action, the MOBAT-SA will receive and parse the instructions, fetch the necessary masking keys, rewrite the query, send it to be processed by the DBMS and retrieve the processed results, and finally send those results back to the application that issued the request.

## The Data Masking Formula

Generally, most facts in DWs are columns with numerical values (Kimball & Ross, 2002). Since fact tables usually represent more than 90% of the DW's total size (Kimball & Ross, 2002), it is fair to assume that numeric type columns also represent the largest portion of business data. MOBAT aims on masking the DW's numerical values while introducing small overheads in the computational efforts for query processing. The aim is to ensure that sensitive data is replaced by realistic (but not real) data.

Suppose a table $T$ with a set of $N$ numerical columns $C_i = \{C_1, C_2, C_3, \ldots, C_N\}$ to be masked and a total set of $M$ rows $R_j = \{R_1, R_2, R_3, \ldots, R_M\}$. 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 depends on the following predefinitions:

- $K_1$ and $K_2$ are private keys stored in the Black Box, known only by MOBAT-SA;
- $K_1$ is a 128 bit random generated value, constant for $T$;

- $K_2$ is a 128 bit random generated value, ranging between the minimum and maximum positive integer value possible of column $C_i$, given the maximum storage size of $C_i$. There is a $K_2$ for each column $C_i$ to be masked, represented by $K_{2,i}$;
- $K_3$ is a public key based on a 128 bit column appended to each row $R_j$ in $T$, filled in with a random value between 1 and $2^{128}$, represented by $K_{3,j}$.

Suppose each value to mask as $(R_j, C_i)$. Each new masked value $(R_j, C_i)'$ is obtained by applying the following Formula (1) for row $j$ and column $i$ of table $T$:

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

MOD is the modular operator that returns the remainder of a division expression. Since $K_1$ and $K_{2,i}$ are constant values for the table and each column, respectively, and $K_{3,j}$ is stored along with each row in the table, the inverse Formula of (1) for retrieving the original value is shown as Formula (2):

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

If the values of $K_{3,j}$ were stored in a lookup table separate from table $T$, a heavy join operation between those tables would be required to unmask data. Given the typical enormous number of rows in fact tables, this is unfeasible and should be avoided at all cost. In order to avoid this when using MOBAT, the values of $K_{3,j}$ must be stored along with each row $j$ in table $T$. To manage this, there are two possible solutions:

1. A new column is created and added to table $T$ for storing each $K_{3,j}$ value;
2. Table $T$ is recreated with the inclusion of $K_{3,j}$ in the CREATE TABLE statement from the start and then restoring the table's data.

The second option implies additional efforts and time to rebuild table $T$, depending on its size. However, it should speed up query response time, when compared with the first option, since the new column $K_{3,j}$ is included with the original data in each row from the start; the second option makes it physically stored apart from the remaining original data in the table. The impact on database performance can be compared be observing their results in the experimental evaluation section. The results for MOBAT where the new $K_{3,j}$ columns are added to the fact tables *a posteriori* are referenced as *MOBAT_AddCol*; and the results for MOBAT where the $K_{3,j}$ columns are included in the fact tables from the start and completely rebuilt are referenced as *MOBAT_CreateCol*.

A third option for defining $K_{3,j}$ values and speed up MOBAT 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, no changes in table $T$ data structure are required, eliminating storage space overhead in $T$. However, this limits the strength of the masking Formula (1), 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. The results for this third option for defining $K_{3,j}$ are also shown in the experimental evaluation section, referred to as *MOBAT_ColKey*, where *L_OrderKey* and *S_SaleID* are used as $C_Z$ in the TPC-H and real-world sales DW, respectively; *i.e.*, each value of *L_OrderKey* and *S_SaleID* in each row $j$ of tables *LineItem* and *Sales*, respectively, function as $K_{3,j}$ for MOBAT. The next subsection explains how to query the masked database.

## Implementing the Data Masking Solution in a Database

To mask a database, a DBA must require this action through MOBAT-SA. Entering the DBA login and database connection data, the MOBAT-SA will attempt to login to that database. If it succeeds, the MOBAT-SA will scan all the data access policies defined in the database for identifying authorized users and respective permissions. The Black Box will then be created and updated with those user access definitions and data policies, and an action log for recording all further user actions requested to execute in the database will also be created, as explained earlier. After the previous step is executed, the MOBAT-SA will ask the DBA which tables and columns are to be masked. All the needed private masking keys for each table and column will then be generated, encrypted and stored in the respective Black Box.

Finally, the MOBAT-SA will apply the data masking Formula on all rows of all columns to be masked, replacing the original values with the new masked values. Whenever the database needs to insert new data or modify or delete existing data, this should be done through the MOBAT-SA, which will apply the masking routine to any value which refers to any masked column, and store the masked value directly in place. Contrarily to most standard commercial data masking solutions, the MOBAT-SA also allows reversing the masked database back to its original data, if intended. The next subsection explains how the data is masked by the MOD-based technique.

## Querying the Masked Database

Whenever user applications wish to execute a query, they submit it to the MOBAT-SA instead of directly querying the database. The MOBAT-SA then rewrites the received query in order to process it with the real data values, using Formula (2) to replace the respective masked columns used in the query, and checking the user access definitions in the Black Box to see if it comes from an authorized user. To rewrite the user query, the MOBAT-SA searches for which tables and columns it needs to process, and looks up the Black Box for retrieving the needed $K_1$ 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 (TPC, 2011) 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 public $K_{3,j}$ key values, and has stored the value of 9342 (for example) for key $K_1$ referring to the *LineItem* table, as well as $K_{2,\ L\_Quantity} = 12$, $K_{2,\ L\_ExtendedPrice} = 51234$, and $K_{2,\ L\_Discount} = 4$ (for example also). Consider TPC-H query 6:

```
SELECT SUM(L_ExtendedPrice * L_Dis-
count) AS Revenue
FROM   LineItem
WHERE  L_ShipDate>=TO_DATE('1994-01-
01')
   AND L_ShipDate<TO_DATE('1995-01-
01')
   AND L_Discount BETWEEN 0.05 AND
0.07
   AND L_Quantity<24
```

The new query, rewritten by the MOBAT-SA and submitted to the DBMS will be:

```
SELECT SUM((L_
ExtendedPrice+MOD(MOD(L_
KeyK3,9342),51234)-51234) *
           (L_Discount+MOD(MOD(L_
KeyK3,9342),4)-4)) AS Revenue
FROM   LineItem
WHERE  L_ShipDate>=TO_DATE('1994-01-
```

```
01')
   AND L_ShipDate<TO_DATE('1995-01-
01')
   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
```

As seen in the example, query parsing and adaptation is a straightforward operation, replacing each masked column with their respective reverseFormula (2). This is valid for any type of query, including equality and range queries, as well as built-in functions. The changes to the queries are transparently handled by the broker and kept hidden from users. Only query results are returned to user applications.

## Using False Data Injection

MOBAT may also be used for injecting false rows throughout the fact tables, making it increasingly difficult to distinguish true and false data, in order to mislead attackers that gain direct access to the database. To achieve this, instead of generating independent random numbers for the values of the $K_{3,j}$ keys in each row, as previously described, $K_{3,j}$ is redefined as a multiple of the sum of the true original values of all $C_{i,j}$ columns to be masked, for each true row $j$:

$K_{3,j} = (\sum C_{i,j}) * k$, $\{ i = 1 \ldots n \}$ where $k$ is a random integer constant that does not overflow 128 bits for $K_{3,j}$ and $n$ is the number of masked columns $C$ in row $j$ (3)

For false rows, random values for filling each column $C_{i,j}$ would be generated, and the value of $K_{3,j}$ would be equal to any value different from those possibly generated by Formula (3). Thus, true rows are verifiable through testing if $K_{3,j}$ is a multiple of the sum of the true unmasked values of all masked columns, using the MOD remainder operator. Formula (4) shows how to test if a certain row $j$ is true or false:

Given R = $K_{3,j}$ MOD $(\sum C_{i,j})$ , $\{ i = 1 \ldots n \}$ as in (3)

IF R=0 THEN row $j$ is True ELSE row $j$ is False (4)

There is a tradeoff between security and performance when using this false data injection method. The more false data is injected, the stronger is the level of security of the table. However, the more data is injected, the more data is scanned and verified by the queries, decreasing database performance. The increased overall security strength for each fact table is directly dependent on how many false rows should be injected into each table, and how to distribute the false rows throughout the existing data.

## Security Issues

### Encrypting the Contents of the Black Box

In what concerns the Black Box, all of its content is encrypted using the standard AES 256 bit algorithm, making it as secure in this aspect as any other similar encryption solution for stored data (*e.g.* Oracle 11g TDE and Microsoft SQL Server 2008 TDE). The only allowed access to the Black Box's content is done by the MOBAT-SA, which is managed only by the application itself.

### Handling Transparency and Securing Communications

All user queries and instructions are managed by the MOBAT-SA, which transparently parses and rewrites them to query the DBMS and retrieve the intended results. The users never see the

rewritten instructions. For security purposes, the MOBAT-SA shuts off any historical log on the database managed by the DBMS before requesting the execution of the rewritten instructions, so that they are not stored in the DBMS, since this could disclose the private keys. All communications between user applications, MOBAT-SA and the DBMS are performed through encrypted SSL/TLS connections. All these actions prevent attackers from accessing the masking keys, rewritten queries/instructions and true data.

*Attack scenarios*. Given their massive amount of data and complex performance optimization structures, DW databases are only updated at very specific moments in an extremely controlled environment, with the database offline to its users (Kimball & Ross, 2002). Since DWs are typically in a read-only mode when they are online, attack scenarios on the masked database can come from two types of attackers, interested in data theft: 1) a masqueraded user (an authorized user with malicious intent or someone who has stolen password and login information and logs in the system as an authorized user); or 2) an attacker that manages to bypass the MOBAT-SA and gain direct access to the database. It is assumed that the attackers have access to the masking Formula and the masked values, and do not know the values of masking keys $K_1$ and $K_2$ for each masked value.

## Handling Masqueraded Attackers

In what concerns masqueraded attackers, all logins and queries submitted to the MOBAT-SA are automatically encrypted and permanently stored in the Black Box historical action log without change. The Black Box can never be manually accessed or updated by anyone except the MOBAT-SA itself. Technically, if a DBA is allowed to control security without any restriction (which may happen in the Oracle 11g TDE solution, for instance), the whole system becomes vulnerable to malicious DBA actions. To manage this, MOBAT-SA allows any user with administration privileges to query the read-only historical log, so any DBA can watch over other DBA to check for misuse. Since all database access is controlled by the MOBAT-SA, extracting the predefined data access policies in the first instantiation with the database to mask, from data access policies previously defined using the DBMS. Subsequent changes in data access policies by DBAs must be done through the MOBAT-SA. Since these requested changes are also stored in the Black Box history log, changes in data access policies with the purpose of executing malicious actions can always be detected. This log can also be used to develop an intrusion detection system (which is not within the scope of this work).

## Handling Attackers with Direct Access to the Database

The masked data in the database remains masked at all times. For attackers that bypass the MOBAT-SA and gain direct access to the database, they will only see masked data, reflecting realistic but false values. The exposure in this case of attack is similar to those of similar scenarios where the attacker can see the encrypted data. Similarly, the attacker's task is to try and crack the masking/encryption keys by exhaustive key search in chosen ciphertext attacks.

## Generating Apparent Randomness for the Masked Values

Generating randomness for masking and cryptographic applications is a costly and security-critical operation (Barbosa & Farshim, 2009). In order to guarantee their security strength, two same original real data values must generically originate different masking generated values, so a level of apparent randomness is ensured. Given that the proposed masking Formula (1) uses two MOD operations in conjunction with randomly generated realistic values, the generated masked values for

*Table 2. Example of original and resulting MOBAT dataset*

| T – Original dataset | | | T‘ – MOBAT Masked dataset | | |
|---|---|---|---|---|---|
| *Column1* | *Column2* | $K_{3,j}$ | *Column1‘* | *Column2‘* | $K_{3,j}$ |
| 11 | 91873 | 7537 | 22 | 162590 | 7537 |
| 2 | 94129 | 1808 | 6 | 170575 | 1808 |
| 18 | 71624 | 29636 | 22 | 148034 | 29636 |
| 19 | 38824 | 50877 | 22 | 112521 | 50877 |
| 15 | 84624 | 34997 | 22 | 155673 | 34997 |
| 12 | 46926 | 41395 | 17 | 120841 | 41395 |
| 15 | 92503 | 23744 | 19 | 165541 | 23744 |
| 19 | 28562 | 46700 | 23 | 101600 | 46700 |
| 19 | 41042 | 58902 | 25 | 114080 | 58902 |

the same original data values are mostly different. To demonstrate this, suppose a table $T$ with two masked columns, *Column1* and *Column2*. Suppose the MOBAT-SA generated the values $K_1 = 9264$ for table $T$ and $K_{2,1} = 12$ and $K_{2,2} = 78254$ for each column. Table 2 shows the original data for $T$ on the left and its resulting masked content on the right. It can be seen that the same original values of *Column1* result in different masked values, and vice-versa, achieving apparent randomness. Of course, this is a very small dataset used only to illustrate these features.

## Non-Invertibility of the Masking Formula

The MOBAT masking Formula uses two consecutive MOD operations. For a function to be invertible, each output must correspond to no more than one input, *i.e.*, more than one different inputs cannot generate the same output; a function with this property is called one-to-one, or information-preserving, or an injection. An injective function is a function that preserves distinctness: it never maps distinct elements of its domain to the same element of its codomain. Since the MOBAT operator is non-injective, given that for $X$ MOD $Y = Z$, the same output $Z$, considering $Y$ a constant, can have an undetermined number of possibilities in $X$ as an input which will generate the same value $Z$ when applying the operator (*e.g.* 15 MOD 4=3, 19 MOD 4=3, 23 MOD 4=3, 27 MOD 4=3, etc). Since MOD operations are non-injective, this means the MOBAT Formula (1) is also non-injective. Given that injectivity is a required property for having invertibility, MOBAT is therefore, non-invertible. The only way to break its security is to crack the masking key values of each column.

## Key Management

As known (and it is assumed that the attackers have access to the masking Formula), the level of security of data masking or encryption solutions does not depend on its secrecy, but on its keys (Nadeem & Javed, 2005). The quality of each set of operations in achieving the intended “data mix” affects the performance of the algorithm. Thus, there is always a tradeoff between security and performance in these algorithms. As discussed in (Kim et al., 2010; McKendrick, 2009; Nadeem & Javed, 2005), there is no easy way of obtaining impartial and widely accepted values for defining the minimum number of secure rounds for each algorithm.

In the proposed masking Formula, there are three keys: $K_1$ is a unique value generated once

for each table and made constant for all values to mask in that table; $K_2$ is a unique value generated once for each column in each table and made constant for all values to mask in that column; and $K_3$ is a value generated for each row in the table, made constant for all the values in the columns to mask in that row. Since $K_3$ is public (given that it is stored in the fact table), only key values $K_1$ and $K_2$ need to be discovered for retrieving the real data values. $K_1$ is a 16 byte integer key, *i.e.*, a set of 128 bits. $K_2$ depends on the maximum storage size defined for each column, variable between 1 and 128 bits. This means that the masking Formula implies a minimum of $2^{129}$ key combinations, for $K_1$ and $K_2$ together (at least 16 bytes + 1 bit), and roughly needs an average number of $2^{128}$ tests (half of the total amount of possible brute force tests – 50% chance) for discovering the keys using brute force, for each masked column in the table, since $K_2$ is column dependant. Consequently, the minimum number of combinations needed to discover all the needed key values for a $i$ number of columns is $i * 2^{129}$, resulting in an average of $i * 2^{128} \approx i * 3.4 \times 10^{38}$ brute force tests in order to discover the keys.

The security strength of standard encryption algorithms is higher, given both the mixes produced in each round and usage of 128 to 256 bit keys, resulting in a higher number and complexity of combinations. Although MOBAT is not so strong in security, it requires much less computational resources, while maintaining a considerable level of security, given the high number of possible brute force attack combinations. Periodically, the masking may be refreshed by rebuilding the masked table values, switching the values of all or any one of the $K_1$, $K_2$, and $K_3$ keys, in order to ensure data is properly protected. Moreover, the data injection method also allows increasing MOBAT's overall security strength. Although it is not possible to absolutely prove that a particular algorithm is secure (Ge & Zdonik, 2007; Kim et al., 2010; Mattson, 2004; McKendrick, 2009; Nadeem & Javed, 2005; Natan, 2005), we believe the proposed technique is secure enough to be acceptable for use.

## Performance and Transparency Issues

### Performance in Middleware Data Confidentiality Solutions

Topologies involving middleware data confidentiality solutions, such as (Radha & Kumar, 2005), typically request all the masked/encrypted data from the database and perform the unmasking/decrypting actions themselves locally. This strangles the network due to communication costs with bandwidth consumption between the middleware and the database, jeopardizing data throughput and consequently, response time. In a DW environment, previously acquiring all the data from the database needed for processing a query at the middleware solution is unreasonable, given the typically large amount of data accessed for answering decision support queries. In this sense, MOBAT-SA just rewrites user queries and then sends them to be processed directly by the DBMS, sending only the results back to the user application that requested the execution of the query. This eliminates network overhead from the critical path, optimizing response time and throughput when compared to other similar middleware security solutions.

### CPU Processing Costs in Data Confidentiality Algorithms

This work is focused on protecting numerical values. This type of data typically represents up to 16 bytes of storage size for each column. The number of clock cycles for encrypting these values depends directly on the algorithms and on the CPU architecture in which they are executed. As an example on a Pentium II CPU (required

by NIST for comparison tests) from (Elminaam et al., 2010), the AES Rijndael algorithm with a 128 bit key, implemented in C, takes up 29 clock cycles per encrypted byte, for encrypting a 16 byte value, resulting in a total of 29*16 = 464 clock cycles. The same algorithm with a 256 bit key takes up 39 clock cycles per encrypted byte, which means it will need 39*16 = 624 clock cycles for encrypting the same 16 byte value. Since a MOD operation on the same CPU takes up 142 clock cycles and an arithmetic sum or subtraction takes up 3 clock cycles (Fog, 2011), for masking the same 16 byte value MOBAT's masking Formula needs 2*142+2*3=290 clock cycles (2 MOD operations, plus one addition and one subtraction). This means MOBAT is 1.6 times faster than AES using a 128 bit key and 2.15 times faster than AES using a 256 bit key to process each 16 byte value on that CPU. However, although the proposed Formula is faster to compute than standard encryption algorithms, this work is not mainly focused on making it faster than those algorithms. Most implementations of encryption algorithms are CPU optimized, designed and programmed for specific processor models and therefore, depending on those CPUs, while the solution proposed in this work is assumed as processor-independent. This is achieved by the high-level SQL reprogramming of the user queries, making it usable in any DBMS, regardless of the used CPU.

### Encryption in Microsoft SQL Server 2008 and MySQL 5.5

Microsoft SQL Server and MySQL 5.5 only encrypt textual or varbinary type values (char, varchar, varbinary, etc). Given that most sensitive columns in DW fact tables store numerical values, when using these DBMS they must be converted to a textual or varbinary format. Once decrypted for processing, these values also must be transformed back into numerical format in order to apply arithmetical operations such as sums, averages, etc. This is a significant drawback, introducing extra computational overheads with evident impact in performance. On the contrary, MOBAT is specifically designed for masking numerical values, and in this sense, is therefore much more appropriate for protecting DW facts.

*Transparently querying masked data.* Query instructions in MOBAT become longer due to replacing each masked column with the masking or unmasking Formulas, but this is automatically and transparently managed by the MOBAT-SA, eliminating user application code changes. The only change the user applications need is to send the query to the MOBAT-SA, instead of querying the database directly.

## EXPERIMENTAL EVALUATION

The TPC-H decision support benchmark (TPC, 2011) (1GB and 10GB scale sizes) and a real-world sales DW storing one year of commercial data (taking up 2GB of data) was used to evaluate the proposed approach. All scenarios were tested in the leading commercial DBMSs, Oracle 11g and Microsoft SQL Server 2008 R2, on a Pentium 2.8GHz CPU with a 1.5TB SATA hard disk and 2GB RAM, 512MB of which devoted to database memory cache. Oracle 11g ran on Windows XP Professional, while SQL Server ran on Windows 2003 Server.

The database schema of TPC-H has one fact table (*LineItem*), and seven dimension tables. The Sales DW database schema has one fact table (*Sales*) and four dimension tables attached to it. In the TPC-H setups, four columns of *LineItem* were masked (*L_Quantity*, *L_ExtendedPrice*, *L_Tax* and *L_Discount*), given they are the numerical fact columns. In the Sales DW, five numerical columns were masked (*S_ShipToCost*, *S_Tax*, *S_Quantity*, *S_Profit*, and *S_SalesAmount*), for the same reasons.

Since MOBAT is column-based, for fairness it is compared with the column-based AES128 and 3DES168 encryption algorithms provided by both DBMS, given that tablespace encryption has functional primitives that speedup performance, making it unfair to compare MOBAT with tablespace-based techniques (Huey, 2008; Oracle, 2010b). Moreover, best practice documentation for encryption in documentation from both DBMSs (Huey, 2008; Oracle, 2010b) recommends using column-based encryption when the sensitive data consists on a small number of well-defined columns. The AES128 and 3DES168 algorithms were used for comparison because they are, respectively, the fastest and slowest available algorithms in those DBMS (Huey, 2008; Oracle, 2010b), as previously mentioned. Table 3 shows the defined experimental encryption/masking scenarios.

## Analyzing Data Loading Performance

Tables 4 and 5 show the results concerning data storage size and loading time (in seconds), respectively, for all data of the TPC-H 1GB *LineItem* fact table, in each defined scenario, in each DBMS. Figures 2 and 3 show the overhead percentages concerning these results. The results in the remaining databases are similar to those shown in the Figures, with absolute values approximately proportional to their database sizes, and to avoid redundancy are not included. The *MOBAT ColKey* setup is not included, since it does not require changing the fact table data structure and thus, presents no overhead in loading data.

In what concerns storage space, MOBAT presents overheads ranging from 4.1% (32MB) to 5.7% (44MB) of extra storage space in Oracle and 2.8% (35MB) in SQL Server. AES128 and 3DES168 present storage space overheads from 103.6% (800MB) to 153.9% (1188MB) in Oracle and 76.3% (944MB) to 94.8% (1173MB) in SQL Server, corresponding to a much higher increase of extra storage space.

In what concerns data loading time, MOBAT presents an overhead ranging from 3.5% (11 seconds) to 7.7% (24 seconds) in Oracle and from 4.3% (9 seconds) to 6.5% (14 seconds) in SQL Server, of extra loading time. AES128 and 3DES168 present much greater loading time overheads, from 189.7% (588 seconds) to 191.6% (594 seconds) in Oracle and 123.1% (261 seconds) to 129.2% (274 seconds) in SQL Server, corresponding to a much higher increase of extra loading time.

As seen in these results, MOBAT is much more efficient than the standard encryption algorithms, introducing very small overheads in both storage space and loading time. Since these results are for the TPC-H 1GB sized database and that the overhead percentages are similar for the remaining tested scenarios, it can be noticed that for the TPC-

*Table 3. Experimental data encryption/masking scenarios*

| Reference/Label | Description |
|---|---|
| Standard | Standard data without masking/encryption |
| AES128 Col | Data encrypted with TDE AES 128 bit key column encryption |
| 3DES168 Col | Data encrypted with TDE 3DES168 column encryption |
| MOBAT AddCol | Data masked by MOBAT Formula (1), where a column for masking keys $K_{3,j}$ has been added to the existing fact table |
| MOBAT CreateCol | Data masked by MOBAT Formula (1), where a column for masking keys $K_{3,j}$ 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 4. TPC-H 1GB line item fact table storage sizes for each experimental scenario*

| DBMS | Standard | AES128 Col | Absolute/ Relative Overhead | 3DES168 Col | Absolute/ Relative Overhead | MOBAT AddCol | Absolute/ Relative Overhead | MOBAT CreateCol | Absolute/ Relative Overhead |
|---|---|---|---|---|---|---|---|---|---|
| **Oracle 11g** | 772MB | 1960MB | *+1188MB / 154%* | 1572MB | *+800MB / 104%* | 816MB | *+44MB / 6%* | 804MB | *+32MB / 4%* |
| **SQL Server 2008** | 1237MB | 2410MB | *+1173MB / 95%* | 2181MB | *+944MB / 76%* | 1272MB | *+35MB / 3%* | 1272MB | *+35MB / 3%* |

*Table 5. TPC-H 1GB line item fact table data loading time for each experimental scenario*

| DBMS | Standard | AES128 Col | Absolute/ Relative Overhead | 3DES168 Col | Absolute/ Relative Overhead | MOBAT AddCol | Absolute/ Relative Overhead | MOBAT CreateCol | Absolute/ Relative Overhead |
|---|---|---|---|---|---|---|---|---|---|
| **Oracle 11g** | 310s | 898s | *+588s / 190%* | 904s | *+594s / 192%* | 334s | *+24s / 8%* | 321s | *+11s / 4%* |
| **SQL Server 2008** | 212s | 473s | *+261s / 123%* | 486s | *+274s / 129%* | 226s | *+14s / 7%* | 221s | *+9s / 4%* |

*Figure 2. Storage space overheads for TPC-H 1GB*

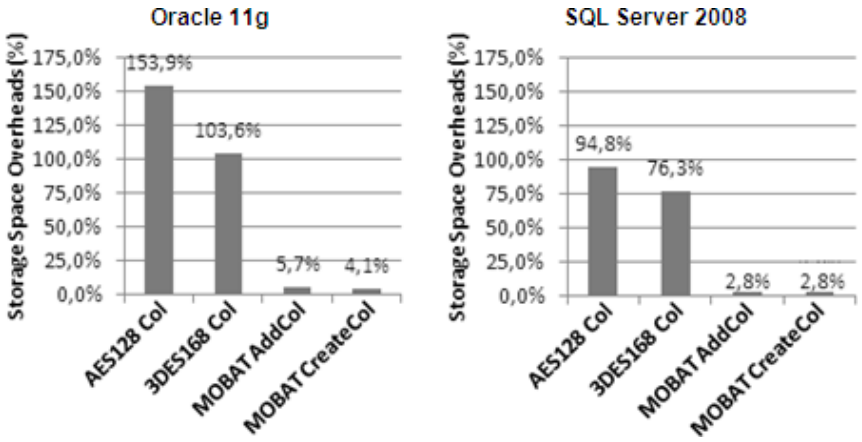
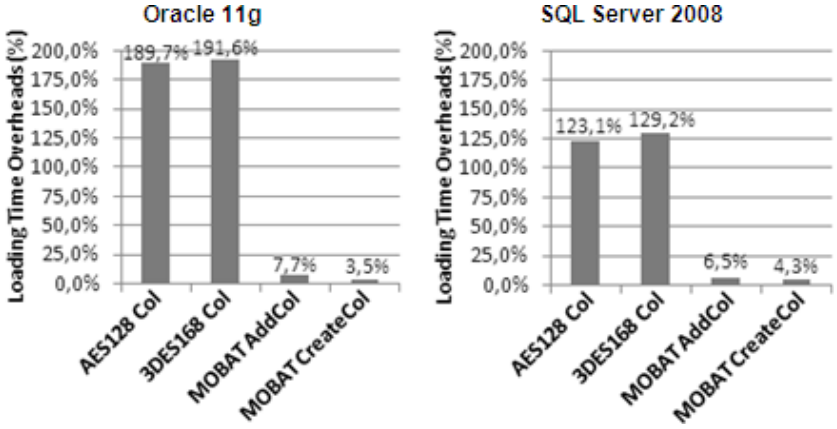



*Figure 3. Loading time overheads for TPC-H 1GB*



H 10GB, which is ten times bigger, the absolute values of the overheads are also approximately ten times bigger. Proportionally, this means that TPC-H 10GB has approximately 8GB to 12GB of increased storage space, and approximately 43 to 99 minutes of increased loading time. Given that 10GB is actually a small size for a DW database, it is easy to conjecture that the overheads introduced by DBMS data encryption algorithms in DWs are extremely significant and may in fact be impracticable.

## Analyzing Query Performance

For TPC-H, the test workload was composed of the benchmark's queries 1, 3, 6, 7, 8, 10, 12, 14, 15, 17, 19, and 20, representing queries accessing the masked fact table *LineItem*. For the Sales DW, the workload was a set of 29 queries, all processing facts in the *Sales* masked fact table, representing a sample of typical decision support queries, such as customer product and promotion sales daily (9 queries), monthly (9 queries) and annual (11 queries) values, with actions as row selection, joining, aggregates, and ordering.

For fairness, databases were optimized in a standard best practice manner in all scenarios (including primary keys, foreign keys, referential integrity constraints and join indexes). Before each execution the database server was restarted. Response time results for each query's execution time are an average obtained from six executions for each tested scenario on each DBMS (with Oracle 11g standard deviations between [0.52, 54.65] and [0.64, 70.10] for 1GB and 10GB TPC-H, respectively, and [0.57, 71.20] for the Sales DW, and SQL Server 2008 standard deviations between [0.49, 52.13] and [0.60, 67.93] for 1GB and 10GB TPC-H, respectively, and [0.55, 70.02] for the Sales DW).

Figures 4 and 5 show the total query workload execution time overhead for each scenario, for each database. The Standard execution time for each scenario, i.e., the execution time of the workload against a non-encrypted/masked database is 626, 6155, and 2233 seconds in Oracle 11g, and 580, 5301, and 2211 seconds in SQL Server 2008, for the 1GB TPC-H, 10GB TPC-H, and Sales DW, respectively. Comparing the overheads introduced by each technique in both DBMS, shown in Figures 4 and 5, MOBAT is much better than AES128 and 3DES168, given the complete query workload in each setup.

In Oracle 11g, MOBAT ranges from at least 5.32 (187.7/35.3) times better than those standard column 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. Notice that column encryption introduces a minimum overhead of 131.8% (8112 seconds) in the TPC-H 10GB setup (total workload response time takes almost 4 hours, instead of the standard time, which is less than 2 hours), and 688.3% (15370 seconds)

*Figure 4. Query execution time overheads for each tested database in Oracle 11g*

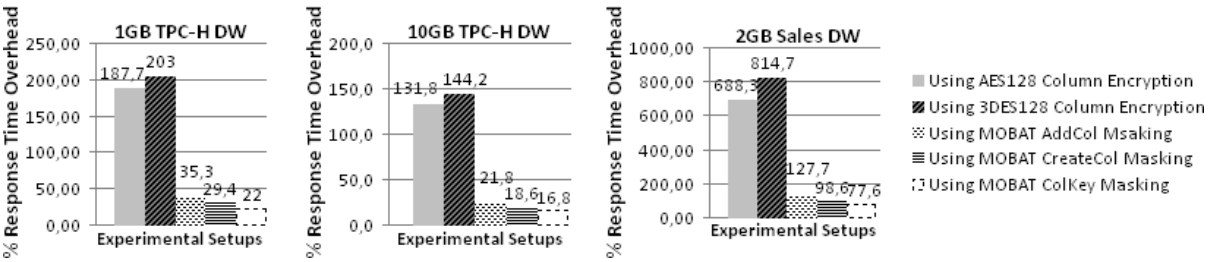
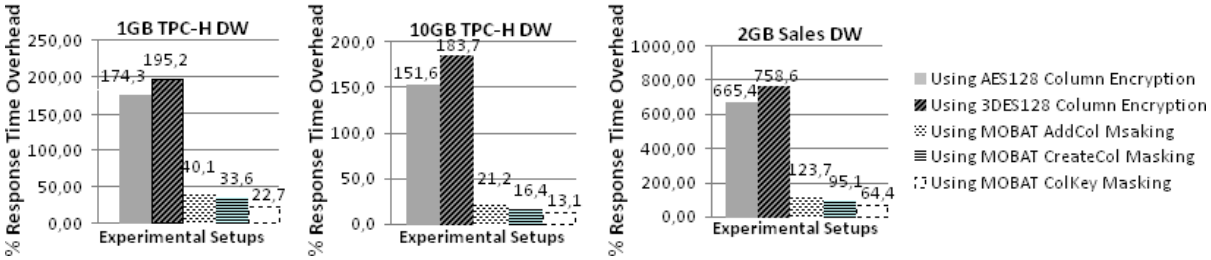


*Figure 5. Query execution time overheads for each tested database in SQL Server 2008*

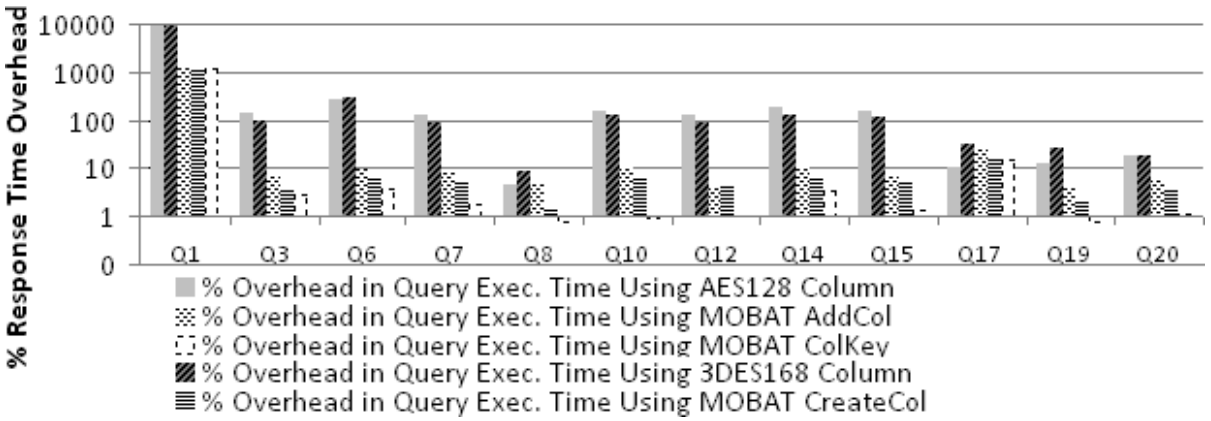



in the Sales DW setup (workload response time takes almost 5 hours, instead of the standard 37 minutes). On the other hand, MOBAT introduces a maximum overhead of 21.8% (1342 seconds) in the TPC-H 10GB setup (total workload response time takes little over 2 hours), and 127.7% (2851 seconds) in the Sales DW setup (total workload response time takes almost 1.5 hours).

In SQL Server 2008, shown in Figure 5, MOBAT ranges from at least 4.35 (174.3/40.1) times better than those standard column encryption solutions for the 1GB TPC-H database, to 8.60 (195.2/22.7) times better. In the 10GB TPC-H database, the gains range from 7.15 (151.6/21.2) to 14.02 (183.7/13.1) times better, and for the Sales DW, from 5.38 (665.4/123.7) to 11.78 (758.6/64.4) times better. Notice that column encryption introduces a minimum overhead of 151.6% (8036 seconds) in the TPC-H 10GB setup (total workload response time takes almost 4 hours, instead of the standard time, which is less than 2 hours), and 665.4% (14712 seconds) in the Sales DW setup (workload response time takes almost 5 hours, instead of the standard 37 minutes). On the other hand, MOBAT introduces a maximum overhead of 21.2% (1124 seconds) in the TPC-H 10GB setup (total workload response time still takes less than 2 hours), and 123.7% (2735 seconds) in the Sales DW setup (total workload response time takes almost 1.4 hours).

The results for each individual query execution time in the Oracle 11g for the TPC-H 10GB scenarios can be seen in Figure 6. These results show that all queries have an overhead similar to those of the complete workload (shown in Figures 4 and 5). This is also true for all the other scenarios in both DBMS, making it redundant to include all results in this section.

*Figure 6. 10GB TPC-H Individual Query Exec. Time Overhead per Encryption/Masking Algorithm*



Query Q1 presents the most significant results because it processes more than 90% of the total fact table data, while the remaining queries process less than 10%. It can be seen that mostly all queries processed by AES and 3DES have introduced overheads of several orders of magnitude higher than MOBAT.

All the results in all scenarios in both DBMS also show that the performance of *CreateCol Masking* is better than *AddCol Masking*, which was expected as previously mentioned, explaining the technique. The performance results of *ColKey Masking* are the best, given the absence of changes in the original fact table data structure and storage size.

## Analyzing the Impact in Performance Using False Data Injection

In order to test false data injection scenarios, 25%, 50%, 75% and 100% of false rows (relatively to the total number of true rows) were inserted into the TPC-H 1GB database using Oracle 11g, according to what was previously explained, using Formula (3) for defining the values of key $K_{3,j}$. All queries used in the tests were modified using Formula (4), to distinguish between true and false data. The false rows were uniformly distributed throughout the fact table. Table 6 shows the results for each scenario, with total workload execution time overhead relatively to the total workload of *MOBAT AddCol* and *CreateCol*, since the *ColKey* setup cannot be used because it already uses a previous existing column and does not allow using the public $K_{3,j}$ key for the false data injection purpose according to Formula (3).

The introduced overheads in each query workload refer to executing the MOD operation from Formula (4) and the extra amount of false rows each query needs to access and test. As it would be expected, the overhead in each scenario is more or less proportional to the amount of false data injected. The results for the remaining database setups, TPC-H 10GB and the Sales DW, are similar to those shown in Table 6, and to avoid redundancy are not included here.

## FUTURE RESEARCH DIRECTIONS

In a traditional DW, users execute much more queries than updates against their databases. Thus, data is static, *i.e.*, there is no loading of new data when the databases are available to its users. In these environments, the main performance issue in most cases is not encryption overheads, but decryption overheads. Since loading of new data is done in well defined time windows in which the database is offline, there is no impact in user query response time; it only affects DW maintenance time. However, the higher is the amount of data to load, the higher the storage space and loading time overheads. Nevertheless, this static data state paradigm has been changing, with the increasing implementation of real-time data warehousing solutions. Thus, given the size of DWs and the amount of data typically processed by decision support queries, the overheads introduced by both

*Table 6. TPC-H 1GB workload exec. time(seconds)/overhead (%) with false data injection using MOBAT in Oracle 11g*

| | +25% False Data (Time/Overhead) | +50% False Data (Time/Overhead) | +75% False Data (Time/Overhead) | +100% False Data (Time/Overhead) |
|---|---|---|---|---|
| **1G TPC-H MOBAT AddCol** | 1110 sec / 31% | 1355 sec / 60% | 1601 sec / 89% | 1855 sec / 119% |
| **1G TPC-H MOBAT CreateCol** | 1045 sec / 29% | 1264 sec / 56% | 1482 sec / 83% | 1709 sec / 111% |

encryption and decryption as well as masking and unmasking algorithms need to be dealt with, for the sake of their feasibility. The development of future data confidentiality solutions must consider the performance of both encryption/masking and decryption/unmasking as critical.

To improve CPU performance and scalability of data confidentiality algorithms, the results obtained by the Salsa20 family of algorithms (Bernstein, 2005, 2008) indicate that using long chains of simple operations instead of short chains of complex operations may allow developing faster solutions, while still being able to maintain significant levels of security.

The basic argument for increasing the block size of the standard 16 bytes to a higher size of 256 bytes, for example, is that it does not need as many cipher rounds to achieve the same conjectured security level. Using a larger block size should provide just as much mixing as the first few cipher rounds and thus, saves time. The basic counterargument is that a larger block size also loses time in CPU models. On most CPUs, the communication cost of sweeping through a 256-byte block is a bottleneck, because they have been designed for computations that do not involve so much data. However, current CPU trends show that their evolution will allow them to compute larger amounts of bits. Thus, future algorithms should take advantage of this, increasing the typical 128 bit block size used in AES. Being able to do parallel processing is a performance booster in speed and scalability.

Some ciphers sacrifice security strength attempting to obtain higher speed. Until this moment, 256 bit keys have been used and considered secure, since the computational efforts in trying to break their security were considered nearly impracticable. However, the recent multi-core CPU trends indicate that this key length will be rapidly surpassed as hardware processing power evolves. Therefore, to avoid rapidly becoming useless, at least 256 bit or higher key lengths should be used in the development of new solutions. Although higher keys should, in principle, bring worse performance, in our opinion the problem is not centered on the key length, but on the used block size and the algorithm itself.

There is always a tradeoff between performance and security; research will probably lead to solutions that are better in database performance, but have less security strength. The main issue is to significantly decrease storage space, resource consumption and response time, while maintaining substantial security strength. A possibility is to develop variable-based dynamic algorithms that enable the user to choose between different key lengths and block sizes, the number of encryption/masking rounds, and any other parameter that could allow DBAs and application developers to fine tune the security-performance tradeoff's balance according to the specific features and requirements of each DW. To our knowledge, this type of solution has never been proposed.

## CONCLUSION

This work presents a data masking solution specifically designed for enhancing data confidentiality in DWs. It also takes advantage of one of the masked fact tables masking key for enabling false data injection, increasing the overall security strength against attackers that gain direct access to the database.

The proposed data masking Formula is composed by a set of two consecutive modulus (division remainder) operations and two simple arithmetic operations. It requires small computational efforts and can be straightforward and easily implemented in any DBMS. Since it basically works by transparently rewriting user queries, it minimizes efforts in changes to user applications driven by changes in DW data structures, nor does it jeopardize network bandwidth. The masked database can be directly used for production purposes, enabling developing applications to directly query it without passing through the

MOBAT application, therefore retrieving realistic data, but never the real data, for testing software development. This also avoids disclosure of the real original data if any attacker bypasses database access control and is able to retrieve data directly from the database.

Although it was not conceived as a direct alternative to standard encryption solutions, it has been compared with the AES and 3DES algorithms, provided by leading commercial DBMS. The experimental results show that the storage space increase and degradation of database performance introduced by these standard solutions is very significant from the DW perspective. This enforces stating that those techniques are in fact too complex and costly in performance to be used in DW scenarios. Given that most DW data consists on numerical values, the proposed masking technique is tailored for this kind of data. Our technique shows better database performance than the encryption standards, while managing to maintain a significant level of security strength, enforced by the false data injection method. Thus, it is an efficient overall solution and a valid alternative for balancing the performance and security issues from the DW perspective.

As future work, we intend to develop our technique in order to accomplish also masking textual values, in order to provide a broader data confidentiality solution. Another research challenge is to take advantage of the history log stored in the MOBAT-SA Black Box to manage intrusion detection.

## REFERENCES

AES. (2001). *Advanced encryption standard*. Nat. Inst. of Standards and Technology (NIST), FIPS-197.

Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2002). *Hippocratic databases*. International Conference on Very Large DataBases (VLDB).

Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2004). *Order-preserving encryption for numeric data*. ACM Special Interest Group Conference on Management Of Data (SIGMOD).

Agrawal, R., Srikant, R., & Thomas, D. (2005). *Privacy preserving OLAP*. ACM SIG International Conference on Management Of Data (SIGMOD).

Arora, E., Ertin, R., Ramnath, M., Nesterenko, M., & Leal, W. (2006). Kansei: A high-fidelity sensing testbed. *Internet Computing, 10*. 3DES. (2005). *Triple DES*. National Bureau of Standards, National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS) Pub. 800-67, ISO/IEC 18033-3.

Baer, H. (2004). *On-time data warehousing with Oracle Database 10g – Information at the speed of your business. Oracle Whitepaper*. Oracle Corporation.

Barbosa, M., & Farshim, P. (2009). *Randomness reuse: Extensions and improvements*. 12th Institute of Mathematics and its Applications (IMA) International Conference on Cryptography and Coding.

Bernstein, D. J. (2005). *Snuffle 2005: The Salsa encryption function*. Retrieved from http://cr.yp.to/snuffle.html

Bernstein, D. J. (2008). The Salsa20 family of stream ciphers. *New Stream Cipher Designs - The eSTREAM Finalists 2008*. *Springer LNCS*, *4986*, 2008.

Bertino, E., & Sandhu, R. (2005). Database security – Concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing*, *2*(1). doi:10.1109/TDSC.2005.9

Callot, O., Frank, M., Gaspar, C., et al. (2009). *High-speed data injection for data-flow verification at LHCb*. Real-Time Conference (RT), 16th IEEE-NPSS.

Data Warehouse Masking Technique – Testing Queries. (2011). *Our project's website*. Retrieved from http://213.13.123.56/MOBAT/queries.html

DES. (1977). *Data encryption standard*. National Bureau of Standards, National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS) Publication 46.

Elminaam, D., Kader, H., & Hadhoud, M. (2010). Evaluating the performance of symmetric encryption algorithms. *International Journal of Network Security*, *10*(3).

Gartner Inc. (2009). *Selection criteria for data-masking technologies*. Research Report ID G00165388, Feb 2009.

Ge, T., & Zdonik, S. (2007). *Fast, secure encryption for indexing in a column-oriented DBMS*. International Conference on Data Engineering (ICDE).

Hacigumus, H., Iyer, B., & Mehrotra, S. (2004). *Efficient execution of aggregation queries over encrypted relational databases*. International Conference on Database Systems for Advanced Applications (DASFAA).

Hacigumus, H., Iyer, B. R., Li, C., & Mehrotra, S. (2002). *Executing SQL over encrypted data in the database-service-provider model.* ACM SIG International Conference on Management of Data (SIGMOD).

Huey, P. (2008). *Oracle database security guide 11g*. Oracle Corporation.

Kim, J., Lee, Y., & Lee, S. (2010). DES with any reduced masked rounds is not secure against side-channel attacks. *Elsevier International Journal of Computers and Mathematics with Applications, 60*. Retrieved from www.elsevier.com/locate/camwa

Kimball, R., & Ross, M. (2002). *The data warehouse toolkit* (2nd ed.). Wiley & Sons, Inc.

Kobielus, J. (2009). *The Forrester wave: Enterprise data warehousing platforms*. Forrester Research Report.

Lo, E., Cheng, N., & Hon, W. (2010). *Generating databases for query workloads.* International Conference on Very Large DataBases (VLDB).

Mattson, U. T. (2004). *Database encryption – How to balance security with performance*. Protegrity Corporation Technical Paper. doi:10.2139/ssrn.670561

McKendrick, J. (2009). *IOUG data security 2009: Budget pressure lead to increased risks*. The Independent Oracle Users Group (IOUG) Security Report.

Nadeem, A., & Javed, M. Y. (2005). *A performance comparison of data encryption algorithms*. IEEE International Conference on Information and Communication Technologies (ICICT).

Natan, R. B. (2005). *Implementing database security and auditing*. Digital Press.

Oracle Corporation. (2005). *Security and the data warehouse*. Oracle White Paper.

Oracle Corporation. (2010A). *Data masking best practices*. Oracle White Paper.

Oracle Corporation. (2010B). *Oracle advanced security transparent data encryption best practices*. Oracle White Paper.

Procopiuc, C. M., & Srivastava, D. (2011). *Efficient table anonymization for aggregate query answering.* International Conference on Data Engineering (ICDE).

Radha, V., & Kumar, N. H. (2005). EISA – An enterprise application security solution for databases. In S. Jajodia & C. Mazumdar (Eds.), *International Conference on Information Systems Security (ICISS), Springer LNCS 3803.*

Ravikumar, G. K., Manjunath, T. N., Ravindra, S. H., & Umesh, I. M. (2011). A survey on recent trends, process and development in data masking for testing. *International Journal of Computer Science Issues*, *8*(2).

Santos, R., Bernardino, J., & Vieira, M. (2011). *A data masking technique for data warehouses.* International Database Engineering and Applications Symposium (IDEAS).

Schneier, B. (1994). *Description of a new variable-length key, block cipher (Blowfish).* Fast Software Encryption (FSE), Cambridge Security Workshop.

Transaction Processing Council. (2011). *The TPC decision support benchmark H*. Retrieved from http://www.tpc.org/tpch/default.asp

Vaidya, J., & Clifton, C. (2002). *Privacy preserving association rule mining in vertically partitioned data.* ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Vieira, M., & Madeira, H. (2005). *Towards a security benchmark for database management systems*. International Conference on Dependable Systems and Networks (DSN).

Vimercati, S. C., Foresti, S., Jajodia, S., Paraboschi, S., & Samarati, P. (2010). *Over-encryption: Management of access control evolution on oursourced data*. International Conference on Very Large DataBases (VLDB).

Xiao, X., Bender, G., Hay, M., & Gehrke, J. (2009). *iReduct: Differential privacy with reduced relative errors*. ACM SIG International Conference on Management of Data (SIGMOD).

Yuhanna, N. (2009). *Your enterprise database security strategy 2010*. Forrester Research Report.

## ADDITIONAL READING

Aggarwal, C. (2005). *On k-anonymity and the curse of dimensionality*. International Conference on Very Large DataBases (VLDB).

Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., & Motwani, R. … Xu, Y. (2005). *Two can keep a secret: A distributed architecture for secure database services.* International Conference on Innovative Data Systems Research (CIDR).

Bernstein, D. J., & Schwabe, P. (2010). *New AES software speed records.* International Conference on Cryptology in India (INDOCRYPT).

Du, W., Deng, J., Han, Y., Varshney, P., Katz, J., & Khalili, A. (2005). *A pairwise key predistribution scheme for wireless sensor networks. ACM Transactions on Information and System Security*. TISSEC.

Esponda, F., Ackley, E., Helman, P., Jia, H., & Forrest, S. (2007). Protecting data privacy through hard-to-reverse negative databases. *International Journal of Information Security*, *6*(6), 403–415. doi:10.1007/s10207-007-0030-1

Ferguson, N. (2006). *AES-CBC + elephant diffuser – A disk encryption algorithm for Windows Vista*. Microsoft Corporation Whitepaper.

Gehrke, J. (2002). Data mining for security and privacy. *SIGKDD Explorations*, *4*(2).

Law, Y. W., Doumen, J., & Hartel, P. (2006). Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks, 2*(1).

Russell, M. D. (2004). *Tinyness: An Overview of TEA and Related Ciphers*.

Wunnava, S. V., & Rassi, E. (2002). *Data encryption performance and evaluation schemes.* International Southeast Conference (SoutheastCon).

## KEY TERMS AND DEFINITIONS

**Data Confidentiality:** Domain of Data Security focused on protecting unauthorized data disclosure.

**Data Encryption:** Advanced form of Data Masking with a well defined set of procedures for encrypting and decrypting data, usually in the form of a multi-step algorithm.

**Data Masking:** Also known as Data Obfuscation, it stands for any technique that is able to replace true data with false data.

**Data Security:** Information technology expert field involving the protection of any form of data.

**Data Warehousing:** Analytical databases focused on providing decision support information and deriving business intelligence for enterprises.

**Database Performance Optimization:** Domain of database performance focused on all aspects and techniques that may enhance the performance of inserting, querying, updating or deleting data structures, namely in what concerns response time, throughput, and storage space, among other features.