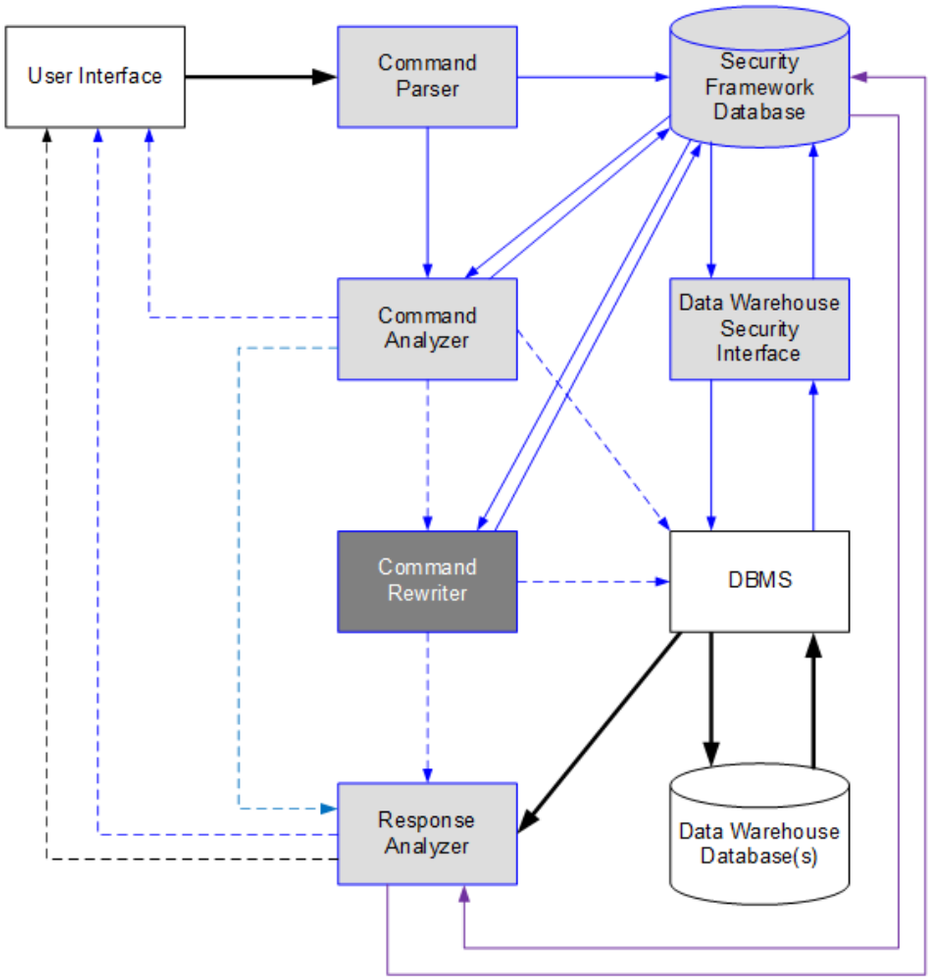
Ricardo Jorge Ribeiro dos Santos
ENHANCING DATA SECURITY IN DATA WAREHOUSING
UNIVERSIDADE DE COIMBRA



Ricardo Jorge Ribeiro dos Santos

ENHANCING DATA SECURITY IN DATA WAREHOUSING

PhD Thesis in Information Sciences and Technology supervised by Professor Jorge Bernardino and Professor Marco Vieira and presented to the Faculty of Sciences and Technology of The University of Coimbra

February 2014

· U C ·

UNIVERSIDADE DE COIMBRA

Enhancing Data Security in Data Warehousing

Ricardo Jorge Ribeiro dos Santos

Thesis submitted to the University of Coimbra in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**

February 2014

Department of Informatics Engineering
Faculty of Sciences and Technology
University of Coimbra

The work presented in this thesis has been developed within the Software and Systems Research Group of the Center for Informatics and Systems of the University of Coimbra (CISUC) as part of the requirements of the Doctoral Program in Information Science and Technology of the University of Coimbra.

This work has been supervised by **Dr. Jorge Fernandes Rodrigues Bernardino**, Professor at the Department of Informatics and Systems Engineering of the Superior Institute of Engineering of Coimbra (ISEC) of the Polytechnic Institute of Coimbra (IPC), and **Dr. Marco Paulo Amorim Vieira**, Professor at the Department of Informatics Engineering of the Faculty of Sciences and Technology (FCTUC) of the University of Coimbra (UC).

Abstract

Data Warehouses (DWs) store sensitive data that encloses many business secrets. They have become the most common data source used by analytical tools for producing business intelligence and supporting decision making in most enterprises. This makes them an extremely appealing target for both inside and outside attackers. Given these facts, securing them against data damage and information leakage is critical.

This thesis proposes a security framework for integrating data confidentiality solutions and intrusion detection in DWs. Deployed as a middle tier between end user interfaces and the database server, the framework describes how the different solutions should interact with the remaining tiers. To the best of our knowledge, this framework is the first to integrate confidentiality solutions such as data masking and encryption together with intrusion detection in a unique blueprint, providing a broad scope data security architecture.

Packaged database encryption solutions are been well-accepted as the best form for protecting data confidentiality while keeping high database performance. However, this thesis demonstrates that they heavily increase storage space and introduce extremely large response time overhead, among other drawbacks. Although their usefulness in their security purpose itself is indisputable, the thesis discusses the issues concerning their feasibility and efficiency in data warehousing environments. This way, solutions specifically tailored for DWs (*i.e.*, that account for the particular characteristics of the data and workloads are capable of delivering better tradeoffs between security and performance than those proposed by standard algorithms and previous research.

This thesis proposes a reversible data masking function and a novel encryption algorithm that provide diverse levels of significant security strength while adding small response time and storage space overhead. Both techniques take numerical input and produce numerical output, using data type preservation to minimize storage space overhead, and simply use arithmetical operators mixed with eXclusive OR and modulus

operators in their data transformations. The operations used in these data transformations are native to standard SQL, which enables both solutions to use transparent SQL rewriting to mask or encrypt data. Transparently rewriting SQL allows discarding data roundtrips between the database and the encryption/decryption mechanisms, thus avoiding I/O and network bandwidth bottlenecks. Using operations and operators native to standard SQL also enables their full portability to any type of DataBase Management System (DBMS) and/or DW. Experimental evaluation demonstrates the proposed techniques outperform standard and state-of-the-art research algorithms while providing substantial security strength.

From an intrusion detection view, most Database Intrusion Detection Systems (DIDS) rely on command-syntax analysis to compute data access patterns and dependencies for building user profiles that represent what they consider as typical user activity. However, the considerable *ad hoc* nature of DW user workloads makes it extremely difficult to distinguish between normal and abnormal user behavior, generating huge amounts of alerts that mostly turn out to be false alarms. Most DIDS also lack assessing the damage intrusions might cause, while many allow various intrusions to pass undetected or only inspect user actions *a posteriori* to their execution, which jeopardizes intrusion damage containment.

This thesis proposes a DIDS specifically tailored for DWs, integrating a real-time intrusion detector and response manager at the SQL command level that acts transparently as an extension of the database server. User profiles and intrusion detection processes rely on analyzing several distinct aspects of typical DW workloads: the user command, processed data and results from processing the command. An SQL-like rule set extends data access control and statistical models are built for each feature to obtain individual user profiles, using statistical tests for intrusion detection. A self-calibration formula computes the contribution of each feature in the overall intrusion detection process. A risk exposure method is used for alert management, which is proven more efficient in damage containment than using alert correlation techniques to deal with the generation of high amounts of alerts. Experiments demonstrate the overall efficiency of the proposed DIDS.

Keywords: Data Security, Data Warehousing, Data Masking, Encryption, Database Intrusion Detection, Database Security Frameworks.

Resumo

As Data Warehouses (DWs) armazenam dados sensíveis que muitas vezes encerram os segredos do negócio. São actualmente a forma mais utilizada por parte de ferramentas analíticas para produzir inteligência de negócio e proporcionar apoio à tomada de decisão em muitas empresas. Isto torna as DWs um alvo extremamente apetecível por parte de atacantes internos e externos à própria empresa. Devido a estes factos, assegurar que o seu conteúdo é devidamente protegido contra danos que possam ser causados nos dados, ou o roubo e utilização ou divulgação desses dados, é de uma importância crítica.

Nesta tese, é apresentada uma framework de segurança que possibilita a integração conjunta das soluções de confidencialidade de dados e detecção de intrusões em DWs. Esta integração conjunta de soluções é definida na framework como uma camada intermédia entre os interfaces dos utilizadores e o servidor de base de dados, descrevendo como as diferentes soluções interagem com os restantes pares. Consideramos esta framework como a primeira do género que combina tipos distintos de soluções de confidencialidade, como mascaragem e encriptação de dados com detecção de intrusões, numa única arquitectura integrada, promovendo uma solução de segurança de dados transversal e de grande abrangência.

A utilização de pacotes de soluções de encriptação incluídos em servidores de bases de dados tem sido considerada como a melhor forma de proteger a confidencialidade de dados sensíveis e conseguir ao mesmo tempo manter um nível elevado de desempenho nas bases de dados. Contudo, esta tese demonstra que a utilização de encriptação resulta tipicamente num aumento extremamente considerável do espaço de armazenamento de dados e no tempo de processamento e resposta dos comandos SQL, entre outras desvantagens ou aspectos negativos relativos ao seu desempenho. Apesar da sua utilidade indiscutível no cumprimento dos pressupostos em termos de segurança propriamente ditos, nesta tese discutimos os problemas inerentes que dizem respeito à sua aplicabilidade, eficiência e viabilidade em ambientes de data warehousing.

Argumentamos que soluções especificamente concebidas para DWs, que tenham em conta as características particulares dos seus dados e as actividades típicas dos seus utilizadores, são capazes de produzir um melhor equilíbrio entre segurança e desempenho do que as soluções previamente disponibilizadas por algoritmos standard e outros trabalhos de investigação para bases de dados na sua generalidade.

Nesta tese, propomos uma função reversível de mascaragem de dados e um novo algoritmo de encriptação, que providenciam diversos níveis de segurança consideráveis, ao mesmo tempo que adicionam pequenos aumentos de espaço de armazenamento e tempo de processamento. Ambas as técnicas recebem dados numéricos de entrada e produzem dados numéricos de saída, usam preservação do tipo de dados para minimizar o aumento do espaço de armazenamento, e simplesmente utilizam combinações de operadores aritméticos conjuntamente com OU exclusivos (XOR) e restos de divisão (MOD) nas operações de transformação de dados. Como este tipo de operações se conseguem realizar recorrendo a comandos nativos de SQL, isto permite a ambas as soluções utilizar de forma transparente a reescrita de comandos SQL para mascarar e encriptar dados.

Este manuseamento transparente de comandos SQL permite requerer a execução desses mesmos comandos ao Sistema de Gestão de Base de Dados (SGBD) sem que os dados tenham de ser transportados entre a base de dados e os mecanismos de mascaragem/desmascaragem e encriptação/ decriptação, evitando assim o congestionamento em termos de I/O e rede. A utilização de operações e operadores nativos ao SQL também permite a sua portabilidade para qualquer tipo de SGBD e/ou DW. As avaliações experimentais demonstram que as técnicas propostas obtêm um desempenho significativamente superior ao obtido por algoritmos standard e outros propostos pelo estado da arte da investigação nestes domínios, enquanto providenciam um nível de segurança considerável.

Numa perspectiva de detecção de intrusões, a maioria dos Sistemas de Detecção de Intrusões em Bases de Dados (SDIBD) utilizam formas de análise de sintaxe de comandos para determinar padrões de acesso e dependências que determinam os perfis que consideram representativos da actividade típica dos utilizadores. Contudo, a carga considerável de natureza *ad hoc* existente em muitas acções por parte dos utilizadores de

DWs gera frequentemente um número avassalador de alertas que, na sua maioria, se revelam falsos alarmes. Muitos SDIBD também não fazem qualquer tipo de avaliação aos potenciais danos que as intrusões podem causar, enquanto muitos outros permitem que várias intrusões passem indetectadas ou apenas inspeccionam as acções dos utilizadores após essas acções terem completado a sua execução, o que coloca em causa a possível contenção e/ou reparação de danos causados.

Nesta tese, propomos um SDIBD especificamente concebido para DWs, integrando um detector de intrusões em tempo real, com capacidade de parar ou impedir a execução da acção do utilizador, e que funciona de forma transparente como uma extensão do SGBD. Os perfis dos utilizadores e os processos de detecção de intrusões recorrem à análise de diversos aspectos distintos característicos da actividade típica de utilizadores de DWs: o comando SQL emitido, os dados processados, e os dados resultantes desse processamento. Um conjunto de regras tipo SQL estende o alcance das políticas de controlo de acesso a dados, e modelos estatísticos são construídos baseados em cada variável relevante à determinação dos perfis dos utilizadores, sendo utilizados testes estatísticos para analisar as acções dos utilizadores e detectar possíveis intrusões. Também é descrito um método de calibragem automatizado da contribuição de cada uma dessas variáveis no processo global de detecção de intrusões, com base na eficiência que vão apresentando ao longo do tempo nesse mesmo processo. Um método de exposição de risco é definido para fazer a gestão de alertas, que é mais eficiente do que as técnicas de correlação habitualmente utilizadas para este fim, de modo a lidar com a geração de quantidades elevadas de alertas. As avaliações experimentais incluídas nesta tese demonstram a eficiência do SDIBD proposto.

Palavras-chave: Segurança de Dados, Data Warehousing, Mascaragem de Dados, Encriptação, Detecção de Intrusões em Bases de Dados, Frameworks de Segurança em Bases de Dados.

Acknowledgements

Firstly, I would like to leave a warm thank you to my advisors. To Professor Marco Vieira by opening new paths and perspectives along the way, and especially to Professor Jorge Bernardino, whose support was and goes far beyond scientific and technical advice. The friendship and respect that I have for both is invaluable and will endure throughout our lives. I would also like to give a warm word of gratitude to Professor Deolinda Rasteiro her amiability and availability whenever I required her assistance.

To my parents and grandparents, who always encouraged and supported me unconditionally in the early stages of my life and provided a safe haven so that I could grow healthy and complete as a person. I would not be here today if it was not for them.

I would also like to thank my family and closer friends that encouraged me in one way or another along these past years to pursue my dreams and which helped me to become a better person, especially my brother, Jorge Santos, and my great friends Adelino Ferreira, Alfredo Cabral, Antonio Ramos, Carlos Ribeiro, Fernando Pais, Julio Valente and Manuel Vaz, among many others.

To my remaining friends, family and all those who in one way or another contributed to my aggrandizement throughout my life and made me a better person, thank you.

Finally, I would like to thank the most important people in my life. I want to thank my wife for the strength and encouragement passed onto me, always cemented in her patience and persistency, and to apologize for the time spent in which I was not available to her. To my sons, my greatest treasures, I thank them for their support and the meaning their existence gives my life, wishing that I can serve as an example of strength and courage to overcome the obstacles of life.

Agradecimentos

Em primeiro lugar, gostaria de deixar um caloroso agradecimento aos meus orientadores. Ao professor Marco Vieira, pelo abrir de novos caminhos e perspectivas ao longo desta caminhada, e especialmente ao professor Jorge Bernardino, cujo apoio foi e vai muito para além do aconselhamento e esclarecimentos técnicos e científicos. A amizade e respeito que nutro pelos dois é inestimável e perdurará pela vida fora. Queria também deixar um agradecimento especial à professora Deolinda Rasteiro, pela amabilidade e disponibilidade demonstradas sempre que necessitei da sua ajuda.

Aos meus pais e avós, que sempre me incentivaram e apoiaram de forma incondicional nas etapas iniciais da minha vida e proporcionaram um porto de abrigo para que pudesse crescer de forma saudável e completa como pessoa. Não estaria aqui hoje se não fosse por eles.

Queria também agradecer a alguns dos meus familiares e amigos mais próximos que me foram encorajando de uma maneira ou de outra ao longo destes últimos anos a perseguir os meus sonhos, e que me ajudaram a ser uma pessoa melhor, nomeadamente o meu irmão, Jorge Santos, e os meus grandes amigos Adelino Ferreira, Alfredo Cabral, António Ramos, Carlos Ribeiro, Fernando Pais, Júlio Valente e Manuel Vaz, entre tantos outros.

Aos meus restantes amigos, família e todos aqueles que, de uma forma ou de outra, contribuíram para o meu engrandecimento ao longo da vida e me tornaram uma pessoa melhor, muito obrigado.

Por último, deixo os agradecimentos às pessoas mais importantes da minha vida. Quero agradecer à minha esposa pela força e encorajamento transmitidos, sempre cimentadas na sua paciência e persistência, e pedir-lhe desculpa pelo tempo que não pude estar disponível para ela. Aos meus filhos, os meus maiores tesouros, agradeço o seu apoio e o significado que a sua existência confere à minha vida, desejando que eu possa servir de exemplo de força e coragem para vencerem os obstáculos da vida.

List of Publications

This thesis relies on the scientific research presented in the following peer reviewed papers.

Peer reviewed papers published in conference proceedings, focusing on surveying data security issues in data warehousing:

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "A Survey on Data Security in Data Warehousing", in *EUROCON 2011 - International Conference on Computer as a Tool*, Lisbon, Portugal, 2011

Peer reviewed papers published in conference proceedings, focusing on data masking and encryption:

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "A Data Masking Technique for Data Warehouses", in *IDEAS 2011 – International Database Engineering & Applications Symposium*, Lisbon, Portugal, 2011

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "Balancing Security and Performance for Enhancing Data Privacy in Data Warehouses", in *TRUSTCOM 2011 - IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, Changsha, China, 2011

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "Evaluating the Feasibility Issues of Data Confidentiality Solutions from a Data Warehousing Perspective", *DaWaK 2012 – International Conference on Data Warehousing and Knowledge Discovery*, Vienna, Austria, 2-5 September 2012

Ricardo Jorge Santos, Deolinda M. L. Rasteiro, Jorge Bernardino and Marco Vieira, "A Specific Encryption Solution for Data Warehouses", in *DASFAA 2013 - International Conference on Databases Systems for Advanced Applications*, Wuhan, China, 22-25 April 2013

Peer reviewed book chapters, focusing on data masking and encryption:

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "Using Data Masking for Balancing Security and Performance in Data Warehousing", *Handbook of Research on Computational Intelligence for Engineering, Science, and Business*, Chapter 15, IGI Global, ISBN 978-1-4666-2518-1 (hardcover) -- ISBN 978-1-4666-2519-8 (eBook), DOI: 10.4018/978-1-4666-2518-1.ch015, 2013

Peer reviewed papers published in conference proceedings, focusing on database intrusion detection:

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "DBMS Application Layer Intrusion Detection for Data Warehouses", *ISD 2012 – International Conference on Information Systems Development*, Prato, Firenze, Italy, 28-29 August 2012

Ricardo Jorge Santos, Jorge Bernardino, Marco Vieira and Deolinda Rasteiro, "Securing Data Warehouses from Web-based Intrusions", *WISE 2012 – International Conference on Web Information Systems Engineering*, Paphos, Cyprus, 2012

Currently submitted journal papers for peer reviewing, focusing on database intrusion detection:

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "Research Challenges in Data Warehouse Intrusion Detection", *ACM SIGMOD Record*, submitted 30 September 2013 (accepted for publication with changes on 21 January 2014)

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "DIDS-DW: A Database Intrusion Detection System for Data Warehouses", *IEEE Transactions on Dependable and Secure Computing (TDSC)*, submitted 26 October 2013

The following peer reviewed papers refer parallel research work that was also published during the development of this thesis, although they are not in the core of the work presented:

Peer reviewed papers published in conference proceedings, focusing on real-time data warehousing:

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira , "24/7 Real-Time Data Warehousing: A Tool for Continuous Actionable Knowledge", in *COMPSAC 2011 - IEEE Signature Conference on Computer Software & Applications*, Munich, Germany, 2011

Ricardo Jorge Santos, Jorge Bernardino and Marco Vieira, "Leveraging 24/7 Availability and Performance for Distributed Real-Time Data Warehouses", in *COMPSAC 2012 - IEEE Signature Conference on Computer Software & Applications*, Izmir, Turkey, 2012

Peer reviewed papers published in conference proceedings, focusing on health care systems:

Ricardo Jorge Santos, Jorge Bernardino and Jorge Henriques, "A 24/7 Monitorization Tool for Avoiding Hypotensive Episodes in Critical Care", in *IDEAS 2010 - International Database Engineering & Applications Symposium*, Montreal, Canada, 2010

Ricardo Jorge Santos, Jorge Bernardino and Jorge Henriques, "The HTP Tool: Monitoring, Detecting and Predicting Hypotensive Episodes in Critical Care", in *EUROCON 2011 - International Conference on Computer as a Tool*, Lisbon, Portugal, 2011

Table of Contents

List of Figures

List of Tables

.

Chapter 1

Introduction

Data is a major asset for any enterprise, not only for knowing the past, but also to support today's business and to predict future trends [Baer, 2004; Kobielus, 2009]. Data Warehouses (DWs) gather all the relevant historical and current business data, reflecting the business measures and its results, as well as how and when it occurs. Given its nature, this data translates into business knowledge, providing invaluable information to generate added business value and support decisions.

In fact, DWs are today's backbone for enterprise business intelligence, playing a main role in the enterprise's outcome [Kobielus, 2009]. Given these facts, we may state that DWs are the core of sensitive business data and store the secrets of the business itself. This makes them a major target for both inside and outside attackers. Consequently, securing DWs against data damage and information leakage is a critical goal.

The awareness of the importance of data security has been growing in the recent years. In fact, a survey on enterprise data security conducted by the Independent Oracle Users Group (IOUG) in 2012 [McKendrick, 2012], shows that almost 50% of the inquired companies increased their investment in IT security, while 9% of the inquired companies stated that they had sustained security breaches in company data. The same report also shows that almost 40% of the companies are expecting a security breach in 2013.

Although several other studies have also demonstrated that efficiently securing sensitive data has become an imperative concern in many enterprises [McKendrick, 2012; Yuhanna, 2009], database attacks are increasing every year in number and complexity, and the caused damage is frequently only discovered after a significant loss of business or financial value [Yuhanna, 2009]. As organizations scale up, the amount of data

moving across their systems and business units, and the risk of data breaches and abuse also grows [McKendrick, 2012]. This introduces the need for integrating effective security measures into databases, given that they are the central component of enterprise information systems.

Regardless of their security purpose, the techniques that are selected for implementing data security in DWs need to consider that data warehousing environments have unique types of user activities, as well as database features[1] and performance requirements, which do not exist in any other type of database system. Therefore, the implementation and usage of the chosen techniques must not jeopardize the feasibility, efficiency and effectiveness of those features and requirements.

In this thesis, we focus on enhancing data security in databases, specifically in the context of data warehousing environments, namely in what concerns data masking, encryption and intrusion detection. The following sections characterize data security in databases, summarily describing the most commonly used techniques, and point out the main issues presented by these techniques from a data warehousing perspective. The chapter continues by presenting the thesis statement, its main achievements and contributions, as well as the structure of the document, which concludes this chapter.

1.1 Data Security in Databases

In this thesis, we adopt the security concepts and definitions described in [Avizienis *et al.*, 2004]. Thus, when referring to data security in databases, it is defined as the composite set of the following attributes:

- *Integrity:* absence of improper modification or deletion of data that may compromise its correctness, completeness, consistency or authenticity;
- *Confidentiality:* absence of improper disclosure of data, *i.e.*, users do not access data they are not supposed to access;

[1] In this thesis, we consider a feature as a variable for assessing the characteristics of a given subject. For example, a database feature can be the storage size of a database or its throughput, among other variables.

- *Availability:* readiness of service, *i.e.*, the required database service and data are always available whenever requested.

To comply with these attributes, many techniques have been proposed in the past. These can be divided in two classes: *preventive* and *reactive* techniques. Preventive data security techniques effectively protect data in advance of security problems or attacks, and independently from the occurrence of those problems or attacks (*e.g.* data masking, encryption, and data access policies, among others), while reactive security techniques are used to effectively respond to the occurrence of a security problem or attack, either while it occurs or after it has taken place (*e.g.* intrusion detection and prevention systems). The following subsections summarily describe the most common types of techniques for each class.

1.1.1. Preventive Data Security Techniques

Besides basic data integrity rules such as the enforcement of referential integrity and low-level hardware and/or software data storage integrity checks against data corruption such as data block checksum functions and error-correcting codes (*e.g.* CRC), used in all databases, the most commonly used preventive data security techniques for protecting sensitive data are probably those that include data masking, encryption and the implementation of data access policies [Huey, 2008].

As one of the earliest methods for protecting data, DataBase Management Systems (DBMS) traditionally use some form of access control to enforce policies regarding the data they manage. Using *data access policies* allows defining the data that each user is authorized to access and the actions that s/he is authorized to execute. This is accomplished through user authentication, which is the process of verifying the user's identity in the system and applying the set of policies defined for the user or the role to which s/he belongs.

Data masking, as the term itself indicates, is the process of obscuring data, either by replacing true values with false values or by hiding a part of its values, in specific data elements. In databases, the main goal of data masking is to replace stored true data with realistic but unreal data, so the true data is unavailable to unauthorized users. An extensive survey on data masking techniques is given in [Ravikumar *et al.*, 2011]. To assure a significant level of security, the false values should not allow attackers to

easily discover ways of retrieving the true values, either by comparison or inference techniques. Organizations have strived to solve privacy issues with hand-crafted solutions or repurposed data manipulation tools within the enterprise to solve the problem of sharing sensitive information. The most common solutions are probably to use scripts with triggers in order to mask and unmask each value, use built-in DBMS data masking packages such as the Oracle Data Masking (ODM) pack [Natan, 2005; Oracle, 2010c], or to embed the masking/unmasking logic within user applications themselves.

Data encryption techniques are an evolutionary and more complex form of data masking which intends to strengthen the security level, obeying to a series of universal principles defined by the encryption research community. It is defined that an encryption algorithm is a procedure or function that handles a given input, performs a series of rounds composed by mixing and transformation actions with that input or part(s) of it, depending on a given encryption key or set of keys, and generates a given output from those mixes and transformations [Vaudenay, 2006]. The algorithms of these procedures or functions are either developed internally within the enterprise to be used in a private manner, or publically disclosed for discussing its merits and proving its secureness by the research community and entities such as the National Institution of Standards and Technology (NIST), so it can be accepted for usage. In the recent past, encryption packages have been progressively implemented in many commercial and open source DBMS such as Oracle, Microsoft SQL Server and MySQL.

1.1.2. Reactive Data Security Techniques

Currently, all main DBMS have audit control, comply with ACID (Atomicity, Consistency, Isolation, Durability) requirements, and supply extensive authentication, authorization, and access control (AAA) features for assuring that the right users access and/or modify only the data that they are supposed to access and/or modify. All main DBMS also have available data masking and encryption packages that can be used transparently with databases and user applications in a straightforward manner. These preventive techniques work effectively in guaranteeing that only authorized users may access and manage the data that they are supposed to access and manage. However, they are unable to distinguish

if the user that has logged in is truly who s/he is supposed to be and/or if that user has or not malicious intentions; if a masqueraded user or malicious insider that has gained clearance by hacking or taking advantage of valid login credentials, those preventive mechanisms are unable to protect data.

Given the increase of sophisticated attacks (*e.g.* Distributed Denial of Service attacks) and rising internal theft, traditional AAA features along with data masking and/or encryption are no longer enough to protect data [McKendrick, 2012; Yuhanna, 2009]. Additionally, attackers that gain direct access to databases mostly represent authorized users logging with permission to access data, meaning that they are able to bypass traditional intrusion detection systems (IDS), which typically work at the network and operating systems (OS) levels. This has lead to the development of reactive data security techniques, which monitor and analyze user actions in the database and try to determine if they are harmful or not in order to adequately deal with them, protecting data from attackers that bypass preventive security techniques.

Gartner Research has identified Database Activity Monitoring (DAM) as one of the most important strategies for decreasing information leakage in organizations [Mogull, 2006; Nicolett and Wheatman, 2007]. Considering an intrusion as an unauthorized attempt to violate the integrity, confidentiality or availability of a system, the detection of intrusion actions against data and inherent database services is the main goal of *Database Intrusion Detection Systems (DIDS)* [Lappas and Pelechrinis, 2007]. DIDS are mainly host-based intrusion detection systems that operate at the database level, *i.e.*, they inspect user commands and/or data workloads just before, during or after that data and/or workloads are processed by the database server. In DIDS there is typically a learning phase (*i.e.*, previous to intrusion detection), in which database and/or user activity logs assumed as having "normal" or intrusion-free activity are used in order to build the "non-intrusive" normal user behavior profiles. To perform intrusion detection, there are mainly two types of approaches: misuse detection, looking for well-known predefined attack patterns; and anomaly detection, looking for deviations from the typical user behavior [Newman, 2011].

1.2 Issues concerning Data Security in Data Warehouses

In spite of the diversity of available data security techniques, their feasibility, efficiency and effectiveness in data warehousing environments has not been undoubtedly proven. On the contrary, in this thesis we demonstrate that several of the currently available data security techniques are in fact unfeasible or, at least, introduce lacks of efficiency and effectiveness or performance overheads with orders of magnitude that jeopardize their feasibility. In the next sections, we point out the issues concerning the data security techniques focused in this thesis, from a data warehousing perspective.

This thesis focuses on enhancing data masking, encryption and DIDS specifically designed for usage in DWs. Therefore, in the following subsections we point out the main issues of each of these techniques from a data warehousing perspective, which make the ground for our work.

1.2.1. Data Masking

Data masking routines are generally simpler in complexity and faster than encryption routines. However, they provide lower security strength [Ravikumar *et al.*, 2011]. As we previously mentioned, encryption algorithms intended to be accepted and widely used by the database community are typically published with open access in order to enable discussing its merits and proving its secureness by both security and database research communities and entities such as the National Institute of Standards and Technology (NIST). This means that before they are put to use, most encryption algorithms go through very thorough and exhaustive analysis and testing processes. If they have been approved, those processes confer a sense of secureness to whoever intends to use them.

For example, the Advanced Encryption Standard (AES) [AES, 2001] became an encryption standard only after a five year long standardization process that included extensive benchmarking on a variety of platforms. Since the appearance of the encryption field within data security, both database developers and users feel much more confident and relaxed with using encryption, rather than simple masking, to protect their sensitive data. This has introduced a confidence issue concerning the use of data masking in highly sensitive databases such as those in DWs.

Data masking routines provided by most commercial tools such as Oracle Data Masking (ODM) typically change data in an irreversible manner, *i.e.*, after masking data it is not possible to subsequently retrieve the original true values. Oracle states that the ODM should be used as a fast and easy way to generate production databases for supporting outsourcing and software development. The ODM can also be used to mask Microsoft SQL Server and DB2 databases for the same purpose. ODM requires new data to be loaded into the database first, and only applies the masking procedures afterwards. It is not possible to load previously masked data; masking in the ODM is an *a posteriori* process. Most commercial solutions work in a similar fashion as the ODM [Gartner, 2009; Huey, 2008].

As not being able to retrieve the original values makes data masking solutions useless in live end user databases [Bertino and Sandhu, 2005a; Gartner, 2009; Huey, 2008; Nadeem and Javed, 2005; Natan, 2005; Ravikumar *et al.*, 2011; Yuhanna, 2009], the lack of confidence in their security strength in some cases and their irreversibility in other cases has made masking techniques the main choice for protecting published data or production data, instead of protecting data in live sensitive end user databases such as DWs.

1.2.2. Data Encryption

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 and Zdonik, 2007; Hacigumus *et al.*, 2004; Huey, 2008; Natan, 2005; Oracle, 2005; Oracle, 2010a; Oracle, 2010c; Vimercati *et al.*, 2007]. However, despite their security strength, encryption techniques introduce performance key costs from a data warehousing point of view:

- *Large processing time/resources for encrypting sensitive data*, since DWs require accessing and processing huge amounts of data, this creates a high demand on computational resources that significantly rises processing time and the required storage space in their databases;
- *Extra storage space of encrypted data*, since DWs usually have many millions or billions of rows, even a small modification of any datatype size to hold encrypted output introduces large storage space overhead;

- *Overhead of query response time and allocated resources for decrypting data to process those queries*. Given the huge amount of data typically accessed in order to process DW queries, this is probably the most significant drawback concerning the use of encryption in DWs [Agrawal *et al.*, 2004].

As the number and complexity of "data-mix" encryption rounds increase, their security strength often improves while performance degrades, and vice-versa. Balancing performance with security in DWs is a complex issue, which depends on the requirements and context of each particular environment. Most encryption algorithms are not suitable for DWs because they have been designed as a *"one fits all"* security solution for general-purpose data. Thus, they are designed for encrypting blocks of text, *i.e.*, sets of character-values by default. This has led DBMS to implement encryption routines that just output textual or binary attributes.

Since in most enterprises the business facts are essentially numerical values, it is fair to state that most DW columns store numerical values [Kimball and Ross, 2013]. Thus, using encryption means that they need to be converted to a textual or binary format. When those values are decrypted for query processing, they need to be converted back into numerical format in order to process sums, averages, etc. Since most decision support queries process mathematical functions and calculus against numerical attributes, conversion operations add computational overheads with considerable performance impact and represent a potentially critical drawback.

Although many encryption algorithms such as [Agrawal *et al.*, 2004; Ge and Zdonik, 2007; Hacigumus *et al.*, 2004; Radha and Kumar, 2005; Vimercati *et al.*, 2007] and built-in DBMS packages such as [Oracle, 2010a] for specific use within databases have been proposed in the past, the introduced performance costs in DWs are very significant and may jeopardize their feasibility or make them unacceptable to users, as we demonstrate in this thesis.

1.2.3. Database Intrusion Detection Systems

Most Database Intrusion Detection Systems (DIDS) rely on command-syntax analysis to compute data access patterns and dependencies for building user profiles [Mathew *et al.*, 2010]. However, as we have

previously mentioned, the considerable *ad hoc* nature of Data Warehouse (DW) decision support workloads makes it extremely difficult to distinguish between normal and abnormal user behavior. Although several DIDS proposed in the recent past are available to be used in DWs, they suffer from a series of drawbacks in these environments:

- Most are poor at detecting novel attacks in dealing with *ad hoc* workloads such as those in DWs and typically spawn too low true intrusion detection rates (allowing many intrusions to pass undetected) or too high false alarm rates [Pietraszek, 2004; Pietraszek and Tanner, 2005; Srivastava *et al.*, 2006; Treinen and Thurimella, 2006];
- Thresholds[2] are typically used to assess the probability of a given action being an intrusion. Given the sensitivity of DW data, using low thresholds is preferable (which consequently generates more alerts), because the potential cost of non-detection is often too high or unacceptable. However, in this case the number of false alarms is often so large that it frequently leads to wasting immense time and resources, or they are simply just too much to be checked [Pietraszek, 2004; Srivastava *et al.*, 2006];
- Although alert correlation techniques have been proposed to deal with large amounts of generated alerts and decrease false positive rates, they are not the best choice for alert management in DW environments. In fact, as these techniques filter sets of alerts in order to decide if each alert is relevant or not, they may allow true intrusions that are capable of producing a great amount of damage to pass undetected, even though they were initially alerted;
- Most DIDS do not assess the damage that each potential intrusion is capable of causing to the data and/or enterprise. Given the business value of DWs, this is a critical issue because it would allow to define which alerts should be checked first, since different data also has different importance to the enterprise;

[2] When mentioned in intrusion detection processes, the term *threshold* is typically a value that sets the limit between normal and abnormal behaviour, given a range domain of possible values that are outputted by those processes.

- Many DIDS execute the intrusion detection (ID) process *a posteriori*, *i.e.*, after the intrusion action has finished its execution. This disables intrusion response and prevention while the intrusion occurs. Given their value, avoiding corruption or exposure of data in DWs as early as possible is a critical issue, making real-time intrusion detection and response capabilities is an essential requirement.

The overstated number of alerts and false alarms, together with the potentially low reliability on correlation techniques and the hypothesis that many intrusions may only be detected and dealt with *a posteriori* jeopardizes the credibility, efficiency and effectiveness of existing DIDS [Bockermann *et al.*, 2009; Lee, 2002; Pietraszek, 2004; Pietraszek and Tanner, 2005; Treinen and Thurimella, 2006].

Another problem that makes it difficult to develop adequate DIDS is the absence of intrusion detection benchmarks at the database level. Benchmarks are an essential instrument used in the development and implementation of many systems. They are widely used because they provide a manner to test those systems and supply solution providers and clients with measures that allow comparing between different solutions, while providing feedback to developers that enables them to improve those solutions. In the past, the KDD99 benchmark [DARPA] has been widely used for testing intrusion detection solutions. However, this benchmark focuses on intrusion actions at the network and operating system (OS) level. In what concerns databases, a need arises for dealing with intruders that are able to bypass intrusion detection mechanisms working at the network and OS level. In spite of the criticality of protecting DW data against intrusions and the importance of having available benchmarks for testing and improving DIDS, to the best of our knowledge there is no benchmark focusing on the specific features[3] of intrusion detection in DW environments at the data level.

[3] The explicit mention to intrusion detection features refer to the variables that are used for building user profiles and that are employed for intrusion detection purposes. For example, the DIDS proposed in this thesis uses features such as the elapsed time for processing each SQL user command, the number of processed rows, the size of the resulting dataset, etc.

1.2.4. Data Security Research Challenges in Data Warehousing

The two main characteristics that differentiate one data confidentiality solution from the other is its ability to secure the protected data against attacks and its speed and efficiency in doing this. Given the specificities of data warehousing environments, we believe there are specific security and performance issues and tradeoffs to evaluate and discuss, regarding the use of data masking and encryption solutions in DWs, which can lead to the development of solutions with better tradeoffs. We also believe that higher efficiency and effectiveness can be achieved in DIDS for DWs if they are designed and/or improved taking in consideration those specificities of data warehousing environments. These are our motivations, which establish the foundations for the research work presented in this thesis.

1.3 Thesis Statement and Main Contributions

This thesis makes several contributions for enhancing data security in DWs at the database server level. We propose specific solutions for implementing data confidentiality, namely novel data masking and encryption techniques, as well as a Database Intrusion Detection System, which consider the unique specificities of data warehousing environments. A framework for integrating all the proposed solutions together is also proposed, supporting the implementation of a unique system that allows increasing the DW's overall security strength.

In detail, the main contributions of this thesis are:

- **A body of knowledge on performance of encryption solutions** in large analytical databases. While encryption solutions are typically characterized and analyzed from a security perspective, we present research findings concerning their performance. It is not within the scope of this thesis to discuss the scientific merit or soundness of the security strength of each technique, but rather to evaluate their impact in database performance and applicability in data warehousing environments. This is obtained by analyzing the design and measured performance of several state-of-the-art and standard encryption algorithms in DWs of various sizes.
- **A body of knowledge on performance of database intrusion detection techniques** focusing on their applicability in data

warehousing environments. We present state-of-the-art intrusion detection techniques and make a clear distinction between them given the way that they determine which features to use and how they manage intrusion detection. Based on this and on the characteristics of typical data warehousing environment workloads, we discuss the suitability or unsuitability of each distinct type of technique for detecting intrusions in DWs. We also point out alert management and intrusion response issues, which can become a critical matter in intrusion damage containment.

- **A novel data masking technique** that introduces small database performance overheads while providing considerable security strength. The technique is used transparently by means of a middleware security broker and sustains the reversible features to retrieve the true original values from the masked values, which makes it useful in live databases such as DWs. It also promotes user action auditing and accountability. Although its security strength is not as high as that of encryption techniques, we believe that this data masking technique is secure enough to be used in scenarios where the performance overheads introduced by encryption are unacceptable, presenting itself as a feasible solution by balancing security and performance tradeoffs.
- **A novel data encryption algorithm** for numerical values that provides considerable security strength while introducing small database performance overhead. Similarly to the data masking technique, our encryption solution is used transparently by means of a middleware security broker and promotes user action auditing and accountability. The proposed encryption technique avoids storage space and computational overhead by preserving each encrypted column's original datatype. Each encrypted column may have its own security strength by defining the number of encryption rounds to execute, which also defines how many encryption keys are used, since each round uses a distinct key (thus, the true key length is the number of rounds multiplied by the length of each round's encryption key). This enables columns that store less sensitive information to be protected with smaller-sized keys and rounds and thus, process faster than more sensitive columns.

Both data masking and encryption techniques maintain the stored data masked or encrypted at all times, requiring only rewriting SQL user commands to function properly and minimal changes to the original data schemas. They use only standard SQL operations and operators, which makes them directly implementable and executable in any DBMS and database setup in a low-cost and straightforward manner. Contrarily to solutions that pre-fetch data to perform masking and unmasking or encryption and decryption, by simply rewriting SQL commands we avoid I/O and network bandwidth congestion due to data roundtrips between the database and the encryption/decryption or masking/unmasking mechanisms, and consequent response time overhead.

- **A specifically designed DIDS for DWs** that works as an extension of any DBMS, adding real-time intrusion detection and response capabilities for each user action executed. The solution acts transparently at the application layer between user applications and the database without affecting their joint functionality. While other DIDS just analyze the user command or its resulting dataset, the proposed DIDS analyzes four distinct aspects of the user's action: SQL command, plus the accessed and processed data, plus the resulting dataset, and enables stopping the user actions, both before and after they are executed by the DBMS, with the ability to avoid the disclosure of their results to the user or application that requested the execution. A declarative SQL-like form for defining intrusion detection and response rules at a fine-grain level is also proposed. These rules allow defining a large spectrum of possibilities for the detection of a wide range of intrusions as well as adequately dealing with them.
- **A risk exposure approach** to be used in the DIDS for ranking alerts, improving the efficiency of damage or leakage containment by pointing out the intrusions that might cause more damage. In cases where the number of generated alerts to be checked is high, the approach enables handling intrusions that indicate a potentially higher risk to the enterprise more rapidly, efficiently and effectively than using correlation techniques.

- **A security framework** that integrates the proposed data masking and encryption solutions with the DIDS into a single conformed workflow between users and the database, which provides a mean for increasing the overall security strength of any DW by enabling each solution to optionally function individually or all together. The framework also defines the guidelines for each type of solution, given the characteristics of DWs and each solution's individual purpose.
- Although not included as fully developed and tested research and therefore, not included as a regular chapter, in Appendix D we include an initial proposal for a **DW Intrusion Detection Benchmark** to test DIDS in DWs at the SQL level, given a controlled DW environment with mixed intrusion and non-intrusion SQL workloads. The main contribution of the benchmark is to provide a feasible and objective mean for evaluating the efficiency of the intrusion detection processes and impact on database performance at the SQL level for DW DIDS. The proposed measures intend to produce insight for aiding developers in the improvement of their solutions and allow providers and users to compare between different solutions.

1.4 Thesis Structure

This chapter discussed the importance of DWs and securing them. It presented key definitions and issues concerning data security in data warehousing environments, creating the ground for the research presented in this thesis. The chapter also presented the objectives and main contributions of the thesis.

Chapter 2 discusses background and related work in the domain background. We characterize data warehousing environments and describe the current state-of-the-art solutions and techniques in data masking, encryption and database intrusion detection. We conclude the chapter by pointing out the issues in each of these subjects from a data warehousing perspective [Santos *et al.*, 2011a; Santos *et al.*, 2012a, Santos *et al.*, 2014].

Chapter 3 presents the integrated security framework, describing each of its components and how they work together to accomplish their security

goals. The framework is defined in a generic way to demonstrate how each individual solution can come together to form a broad scoped overall security approach. The set of principles that drived the development of each data masking, encryption and intrusion detection solution proposed in this thesis is also included.

Chapter 4 proposes a novel reversible data masking technique for numerical values that provides significant security strength and complies with the principles defined in the security framework [Santos *et al.*, 2011b; Santos *et al.*, 2011c]. Besides demonstrating the proof of the masking solution's security strength, this chapter also includes an experimental evaluation to demonstrate that the proposed approach is computationally faster than existing standard and state-of-the-art encryption solutions.

Chapter 5 proposes a novel encryption algorithm for numerical values [Santos *et al.*, 2013]. This technique also complies with the set of principles defined by the security framework. The chapter includes the proof of the proposed solution's security strength along with an experimental evaluation that also show that it is computationally faster than standard and state-of-the-art encryption solutions.

Chapter 6 presents our approach to develop a DIDS focusing on the specificities of data warehousing environments, which is based on the detection of anomalous user activities by joining the syntax-based analysis of the user commands with features of the processed data and the command execution's resulting datasets [Santos *et al.*, 2012b; Santos *et al.*, 2012c]. The DIDS works transparently as an extension of the database server, placed between the user interface(s) and the DBMS, and uses a risk exposure alert management approach that enables it to be more efficient than commonly used alert correlation techniques. An experimental evaluation is included to demonstrate its efficiency against other state-of-the-art intrusion detection solutions proposed in former research.

The last chapter presents the conclusions on this thesis and points out future research directions derived from our work.

Appendix A describes the Sales DW purpose along with its data schema, scale and storage sizes, as well as a list of queries that make up the decision support workloads used in the experimental evaluations presented in the thesis.

Appendix B shows the data masking and encryption experimental results included in Chapters 4 and 5, with its respective statistical measures (averages and standard deviations).

Appendix C shows the intrusion detection experimental results included in Chapter 6, with its respective statistical measures.

Finally, Appendix D describes in detail our initial proposal for a DW intrusion detection benchmark.

Chapter 2

Background and Related Work

Data Warehouses present unique characteristics that differ from other types of database systems. In order to discuss data security from a data warehousing perspective, summarizing those characteristics along with those belonging to the data security solutions focused in this thesis is an essential requirement. This chapter summarily describes the concepts concerning DWs and presents relevant background and related work focusing on standards and state-of-the-art solutions proposed by research in the data security domains focused in this thesis, namely data masking, encryption and DIDS.

The chapter is structured as follows: Section 2.1 summarizes the concepts of data warehousing and the typical characteristics of those environments in what concerns database features and workloads, while Sections 2.2 and 2.3 respectively describe the state-of-the-art data masking and encryption techniques that are currently available for usage in DWs and discusses the issues concerning their use in these analytical environments. Section 2.4 describes the main intrusion techniques and methods used in DIDS and discusses their applicability from a data warehousing perspective. Finally, Section 2.5 concludes the chapter.

2.1. Data Warehousing

In an enterprise, the transactional (alias operational) systems typically consist of a set of applications and data sources that enable accomplishing and storing business transactions, and guarantee their operationability [Kimball and Ross, 2013]. Transactional databases are designed to manage the data for supporting each individual business transaction instead of cross-enterprise business analysis. Transactional systems typically consist of many users reading and writing small amounts of data; for example, on an ATM bank system, there are hundreds or thousands of users accessing

their account balances at the same time, or withdrawing/transferring a given amount of money. Another characteristic of the ATM system is that it does not require keeping long periods of historical data; it only needs the current balance and latest movement records to be able to adequately support user requests and business transactions.

In contrast, Decision Support Systems (DSS) are usually accessed by fewer users but that query large amounts of data to obtain business analysis information to aid decision making. Using the same bank ATM system as an example, the difference is that the people from the bank that need to make decisions regarding the business (*i.e.*, business managers, administrators, executives, etc.) want to know the average balance for the last six months or a year for the accounts with certain geographical region, for instance, in order to make strategic decisions like opening a new branch office or encourage people to increase their investments by offering better interests. To execute this kind of query, the system needs to keep historical data of the balances plus it would read millions of records of all clients within certain region and compute that average.

These type of analytical actions result in very demanding data access patterns, that if running on top of a transactional database can lock large amounts of data and consume computational resources in a way that could compromise the transactional system's availability, ultimately making it incapable of supporting the business transactions. Moreover, many transactional systems operate isolated from each other with little or no integration, and each system typically manages its own dataset. As a result, the same data is represented and stored in many different ways throughout the enterprise, one for each system. Consequently, there can be multiple distinct versions of the truth, which can be inaccurate, outdated or simply invalid.

To relieve resource consumption, reduce the operational risk in the transactional applications that support business, deliver a unique source of true information and provide an optimized data structure for analytical cross-enterprise decision support purposes, Data Warehouses are used, clearly separating the analytical business processes from the transactional business processes. In the next subsection, we present the concepts and definitions concerning DWs.

2.1.1. The Data Warehouse: Concepts and Definitions

The origin of the *Data Warehouse* concept can be traced back to the research carried out at the Massachussets Institute of Technology and at IBM in the late 1970s, focusing on ways to define an architectural model for the flow of data from operational systems to decision support environments. From this research work at MIT, for the first time a differentiation between the operational and analytical processing is made. In 1988, Devlin and Murphy from IBM introduced the term "Business Data Warehouse" [Devlin and Murphy, 1988] that precedes the actual "Data Warehouse" term.

In 1992, Bill Inmon published the first edition of his book "Building the Data Warehouse" [Inmon, 1992] where he defines the term "Data Warehouse" and also consolidated the terms and techniques that have been the foundation for DWs since then. In 1996, Ralph Kimball defined the Star Schema and Multidimensional modeling techniques [Kimball, 1996], which enriched the DW definitions. The Inmon and Kimball approaches were widely accepted by research and commercial database communities and became the common guidelines for building DWs.

In the past, there have been many definitions on what is a Data Warehouse. Although the Inmon and Kimball approaches differ from each other, as well as other derived approaches, they agree on most characteristics that define the concept of what a DW is.

A generic definition of a DW is given by [Kimball, 1996; Kimball and Ross, 2002; Kimball and Ross, 2013]:

> "A Data Warehouse consists of a considerable sized database, which consistently aggregates all the historical data belonging to a given specific business field or business area, in a previously well-defined level of detail that is considered adequate and relevant for decision support purposes by the business itself. The data in a DW can be separated and combined by means of every possible measure in a business".

According to [Simitsis, 2005], the most popular definition of DW is that in [Inmon, 1996; Inmon, 2002]:

> "A Data Warehouse is a centralized repository for the entire enterprise, containing data that is used for analyzing the business and

> supporting decision making. The Data Warehouse has four main attributes: it is *subject-oriented* (meaning the data in it is organized so that all the data elements relating to the same business event or subject are linked together and that the DW is developed in a way that satisfies the analytical requirements of the users that will query it); it is *non-volatile* (meaning that the data loaded into the database is never erased or over-written, *i.e.*, once the data is committed it remains static and read-only and is retained for future reporting and analysis); it is *integrated* (meaning it joins data from several operational data sources into a conformed format in a consistent way); and it is *time-variant* (meaning that it stores the history of the business to which it was designed for)."

Based on these definitions, in this thesis we consider a DW as a large-sized non-volatile cross-enterprise analytical database that stores historical, non-volatile, integrated, consolidated, updated and consistent data, in a level of detail and format considered adequate for providing decision support information in a given business area or field by the business stakeholders.

Having explained the principles and concepts that define a DW, it is also important to understand the environments in which they function. Therefore, in the next subsection we characterize data warehousing environments.

2.1.2. Data Warehousing Environments

The DW obtains its data from the operational data sources (which may consist of transactional databases, flat files, legacy systems, etc.) through the execution of *Extraction, Transformation and Loading* (*ETL*) processes, but clearly separates the analytical decision support processes (which mainly consist on executing On-Line Analytical Processing (OLAP) operations in order to generate a diverse variety of Business Intelligence (BI) reports) from the transactional business processes.

Figure 2-1 shows the traditional generic functional architecture of a DW, composed by the *ETL Layer*, and the *Data and Metadata Repository Layers*[4]. The *ETL Layer* is responsible for the execution of ETL processes and typically contains a staging area which is used to store extracted and transformed data until it is time to load that data into the DW database(s). The *Data and Metadata Layer* contains all DW databases, in which the *Metadata Repository* is used to describe in detail all DW objects and their relationships. In some DWs, the databases are divided into data structures named as *Data Marts*, which focus on storing the decision support data for a specific business subject within the enterprise. The *Presentation Layer* represents all front-end interfaces that are available to the DW end users for accessing its data.

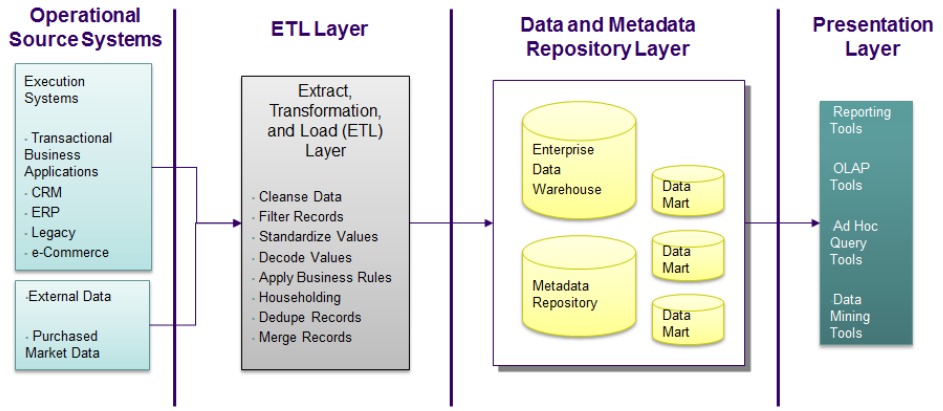


Figure 2-1. Generic Data Warehouse Functional Architecture

Separating the analytical business processes from the operational transactional business processes allows the enterprise to gain at least two major advantages:

- It enables physically and logically separating the transactional databases from the analytical databases and defining adequate specific allocated resources for each type of process. This means that each

[4] Diverse architectures such as that defined in [Kimball and Ross, 2013] also include the *Presentation Layer* as part of the DW itself, but in this thesis we consider the first two layers as the DW core and the third as a separate tier representing the user interfaces.

database can be designed and defined the best possible way in order to adequately fulfill its purposes and maximize its performance regarding those purposes;

- Reporting and *ad hoc* decision support querying is requested by the *Presentation Layer* to the mechanisms existing in the *Data and Metadata Layer*, which are isolated apart from the transactional business databases and thus, does not affect the functionality and/or availability of the operational source systems and vice-versa.

Bearing in mind the way a DW operates, there have been several definitions of what is considered a data warehousing environment. According to [Chaudhuri *et al.*, 1997]:

> "Data Warehousing is a collection of decision support technologies that aim at enabling an enterprise to make better and faster decisions."

Another definition of data warehousing is given in [Castro, 2009]:

> "The concept of data warehousing consists of architectures, tools, technologies, algorithms and methodologies that allow for the construction, usage, management and maintenance of the hardware and software used for a data warehouse, as well as the data itself."

Based on these definitions, in this thesis we consider data warehousing environments as the full setup of hardware and software in which the ETL processes and databases belonging to DWs operate, plus their user workloads.

In what concerns the characterization of the type of users in data warehousing environments - considering users as anyone who may regularly access the DW database(s) for any reason - we consider three main classes of users, given their typical activities:

1) The *Database Administrator (DBA)* or similar, which is anyone that can create or modify any database object. Typical actions on behalf of this user are the creation or modification of tables, indexes and views in the DW, for example. DBAs typically have full (or almost) access privileges to the databases.

2) The *ETL User*, which is any person or software responsible for updating the contents of the DW. Typical actions are new row inserts in fact tables and new row inserts or row updates in dimensional tables.

3) The *DW End User*, which is any person belonging to the business that queries the databases with the purpose of obtaining decision support information or produce business knowledge, either by directly querying it or by using business intelligence and OLAP tools.

In the next subsection, we describe the differences between operational systems and data warehousing environments.

2.1.3. Data Warehousing Environments vs Operational Systems

From a perspective attending to its purpose, as we have previously explained, a DW is mainly a database (or set of databases) system that has been specifically designed to provide decision support information and produce business knowledge, while an operational system is specifically designed to support individual business transactions and store its respective data. Given that the business often requires the operational system to be online in order to accomplish a transaction, operational system requirements focus on enabling high availability in order to avoid compromising the accomplishment of the transactions themselves. On the other hand, since most decision support queries often require processing a large amount of data, DWs focus on enabling high throughput [Kimball and Ross, 2013].

From a perspective attending to the size and shape of its contents, a DW is composed of consolidated historical business data, mostly conformed and within data schemas that allow optimizing the execution of OLAP queries by tools that deliver the intended decision support information and produce the intended business knowledge. In most cases, storing the complete business history implies taking up a very large amount of storage space, often ranging from gigabytes to terabytes. In contrast, operational systems aim to keep their data sources "light", *i.e.*, small in size and content, in order to minimize processing efforts and consequently keep their availability as high as possible, therefore keeping only the exact amount of data which is required to support current and near-future business transactions.

In what concerns their data schemas, transactional databases have highly normalized schemas, mainly to avoid data redundancy and keep each table small-sized, while DWs have denormalized schemas. Most DW database schemas are based on star schemas, where business facts are stored in a central table called fact table (*e.g.* sales table) and the tables containing the business descriptors are called dimension tables (*e.g.* customer and product tables) [Kimball and Ross, 2013]. Dimension tables are linked to the fact table by their primary keys (*e.g.* CustomerID and ProductID) and are usually small in size (typically less than 10% of DW total storage space) and have a small amount of rows (up to tens of thousands), when compared with fact tables, which are typically very large in size and a huge amount of rows (millions or billions). Business facts are mainly stored in numerical-typed attributes within fact tables; since fact tables typically take up at least 90% of the DW total storage size, in many cases DW databases are mostly composed by numerical values [Kimball and Ross, 2013].

Attending to the user's responsibility among the business, the typical DW user is a business manager or someone that holds a considerable role of responsibility in the enterprise, while the typical user of operational systems are mainly transactional operators with low responsibility and with few or none decision making privileges. Since they mainly consist of business managers and decision makers, the number of DW users it typically low (a few tens).

While in operational systems end users typically execute intensive read and write instructions, DW end users only execute read-only instructions such as queries, *i.e.*, they are not allowed to change data, while DBAs and ETL users may insert or modify data. More than 90% of actions executed in DWs are typically decision support queries, (*i.e.* SELECT statements), mainly executed against fact tables [Kimball and Ross, 2013]. Reporting (*i.e.* periodically running reports for answering predefined decision support queries) is typical in DWs. Besides predefined reporting, in many cases a very significant amount of decision support queries are *ad hoc*, which makes them mostly unpredictable in their syntax and frequency. In operational systems, the queries are almost fully simple and predefined and repetitive.

Although decision support queries may typically access huge amounts of data, their response usually results in small datasets with a few hundred bytes and a relatively low number of columns (no more than a few tens). Most queries in DWs are CPU intensive and can take up to hours, while operational system queries are intended to be computationally fast and deliver very small response times.

Table 2-1 summarizes the main differences between operational systems and DWs, based on [Inmon, 2002; Kimball and Ross, 2013; Ponniah, 2010].

Table 2-1. Main Differences between Operational Systems and Data Warehouses

	Operational Systems	**Data Warehouses**
Workload nature/purpose	Transactional	Analytical
Temporal nature of the data	Current	Historical and current
Typical database storage size	As small as possible	Very large to huge
Typical number of tables	Medium to high	Small
Typical data schema type	Highly normalized	Denormalized
Typical number of users	Medium to large	Small
Typical user's responsibility towards the business	Low	High
Typical type of command	Read/Write of small amounts of data	Read-only on large amounts of data
Typical command complexity	Simple	Medium to High
Typical operation dynamics	Static, predefined, predictable and repetitive	Dynamic, ad hoc, random and iterative
Typical command response time	Small	Large
Typical command action	Read/write of a single row or few rows	Reporting and aggregation on many rows, with roll-up, drill-down, slice and dice
Amount of data typically processed by each command	Small	Large or Very Large
Typical data update frequency	Often in a given period of time	Once periodically
Dataset size typically resulting from a command execution	Small	Variable (often Small)

Conclusively, it is widely recognized that DW/BI systems have profoundly different needs, clients, structures, and rhythms than those of operational systems. DW users have drastically different needs than operational system users [Kimball and Ross, 2013]. Thus, we can make the assumption that data warehousing environments also require distinct security solutions that are designed taking those specific characteristics under account and that are able to cope with those specific requirements and needs.

The following sections present the background in data masking, encryption and intrusion detection.

2.2. Data Masking

An extensive survey on data masking (alias data obfuscation) techniques is given in [Ravikumar *et al.*, 2011]. The main goal of data masking is to replace true data with realistic but not real data, so the true data is not readable by unauthorized users. To assure a significant level of security strength, the masked values should not allow attackers to easily discover ways of retrieving the true values.

In this section, we shall explain the diverse forms of masking data, refer available commercial masking packages and discuss the issues concerning the use of data masking in data warehousing environments.

2.2.1. Forms of Data Masking

One way to accomplish data masking is to use **value referencing**, *i.e.*, to create and use a reference table for each masked value, as shown in Figure 2-2.

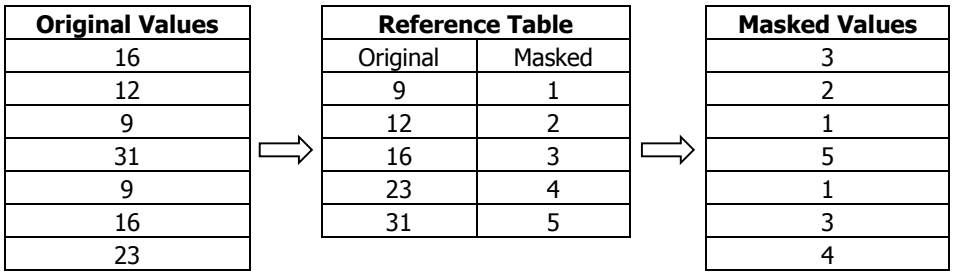


Figure 2-2. Data masking using a reference table

Another way is to use a **function** against each original value to produce a new masked value, such as shown in Figure 2-3.

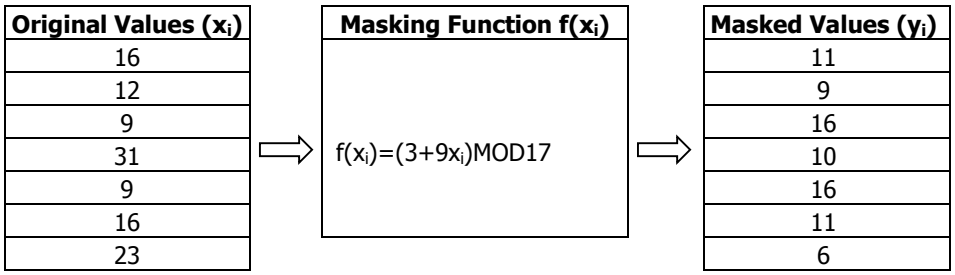


Figure 2-3. Data masking using a masking function

There are several types of functions as shown in Figure 2-3 that can be used for data masking, such as:

- **Deterministic masking:** A deterministic function $f(x) = y_i$, where $f(x_i)$ always produces the same y_i for a given value x_i;
- **Condition-based masking:** Applying different mask formats to the same dataset depending on the rows that match specific conditions (*e.g.* applying different national identifier masks based on country of origin);
- **Compound masking:** A set of related columns is masked as a group to ensure that the masked data across the related columns retain the same relationship (*e.g.* city, state, and zip code values may need to be consistent after masking, for maintaining referential integrity).

These functions are mainly used in two ways, which can be used separately or mixed together:

- **Substituting**, where each value is replaced by the output of a deterministic function or reference (*e.g.* Figures 2-2 and 2-3);
- **Shuffling**, where values switch places. This occurs by swapping the values between two or more predefined similar typed columns in the same row or in different rows, or swapping the characters that compose the value in a predefined form (*e.g.* 12345 becomes 52143), or mixing both these types of swap.

The references and functions shown in Figures 2-2 and 2-3 show data masking operations that are reversible, *i.e.*, the original value can be retrieved from the masked value if an authorized user executes a valid query that should obtain a true result. This is the typical DW setting, where data should be masked for avoiding disclosure to unauthorized users, but all authorized user queries must be able to retrieve the true exact response. However, there are situations in which the released data should not reveal their true values or, at least, not all their true values, in any case (including authorized users). These cases mostly refer to published data for public consulting or outsourcing purposes, or the creation of production and testing databases for aiding software development processes. In these cases, several typical types of techniques allow the disclosure of only part of the true data or entirely false data, such as:

- **Random number generators (RNG)**, widely used for generating statistically independent and apparently random values for simply replacing the original true values in whole or in part;
- **Random shuffling**, where shuffling is used in conjunction with RNG for randomly swapping the values;
- **Nulling**, where sensitive values that should not be disclosed are simply replaced by a null value;
- **Deleting**, where rows with sensitive values are erased;
- **Masking out**, where predefined parts of the sensitive values are replaced by universal characters (*e.g.* credit card number 9255 0614 0015 8925 becomes 9255 XXXX XXXX 8341 or 9X5X 0X1X 0X1X 8X2X);
- **A mix of the previous techniques.**

More recently, research has also proposed non-deterministic methods for masking data, such as **perturbation techniques** [Agrawal *et al.*, 2005; Procopiuc and 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 data warehousing environment. Recent similar work proposing data anonymization solutions which rely on perturbation or differential techniques have been published in [Procopiuc and Srivastava, 2011] and [Xiao *et al.*, 2009].

2.2.2. Commercial Data Masking Solutions

Many similar commercial data masking packages have been developed. Oracle, for instance, has developed the Oracle Data Masking (ODM) pack [Oracle, 2010c], protecting data by replacing real values with realist-looking data with the same type and characteristics as the original data. ODM provides masking primitives such as random numbers, dates and constants, as well as other built-in routines that shuffle the values in a column across different rows. However, once applied, the ODM does not allow retrieving the real values, *i.e.*, the original values are forever inaccessible.

ODM provides a centralized library of out-of-the-box mask formats for common types of sensitive data such as credit card and phone numbers, national identifiers (*e.g.* social security numbers), etc. By leveraging the ODM Format Library, data privacy rules can be applied across enterprise-wide databases from a single source, ensuring consistent compliance with regulations. ODM supports the concept of application masking templates, which are XML representations of the mask definitions. Security administrators, software vendors or service providers can then import these predefined XML templates into the ODM in order to ease and accelerate the data masking implementation process. The ODM automatically identifies and ensures referential integrity.

Oracle states that ODM is to be used mainly as a fast and easy way to generate production databases for supporting outsourcing and software application development. The ODM can also be used to mask Microsoft SQL Server and DB2 databases for the same purposes. ODM requires new data to be loaded into the database first, and only applies the masking procedures afterwards. It is not possible to load previously masked data. Masking in ODM is an *a posteriori* process. Most commercial data masking solutions work in a similar fashion as ODM.

2.2.3. Using Data Masking in Data Warehouses

Organizations have partly strived to solve confidentiality and privacy issues by using hand-crafted solutions or repurposed data manipulation tools developed 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, keeping their secrecy aspects mostly within the development team.

However, these proprietary solutions are not the best way to achieve a standard data masking solution. On one hand, embedding them into applications makes their maintenance complex and costly. On the other hand, not disclosing them to the security and database research communities and keeping them as a hidden black box solution keeps their security strength unproven. Another common solution is to use standard commercial masking tools such as ODM.

Since DWs mainly require masking solutions to guarantee that the masked values can be reengineered to retrieve their original true values, we can state that using RNG, random shuffling, nulling, deleting, and masking out techniques are mostly not suitable for data warehousing environments. Thus, most leading commercial data masking packages such as ODM are also not applicable to most data warehousing scenarios. Consequently, to be useful, DW data masking routines must be based on reversible shuffling or substituting techniques.

Designing an efficient substitution or shuffling routine is far from being a trivial task. If the masking values produced by those methods can be easily determined by comparison or other type of inference then the original true data can be easily retrieved by attackers, making the routines useless. For example, if the shuffle algorithm simply runs down a table swapping the column data of the sensitive columns in between every group of two rows, it would not take much effort from the attacker to break security.

Shuffling routines can ensure higher security strength than simple substitution routines, because they shuffle the values and can add the use of a value-function for changing their values before or after the shuffle. However, shuffling routines may become extremely complex, namely in determining how to swap the values in order to guarantee both an acceptable security strength and processing time overheads. On one hand, limiting the shuffling between columns of the same row allows minimizing data access time for the masking actions but reduces security strength, compared with shuffling throughout the typically huge number of rows in DW sensitive fact tables. On the other hand, shuffling the values spaced throughout those table rows greatly increases the leaps the DBMS engine needs to execute in the datafile to retrieve the true data in the correct order,

since the masked values for each row are distributed up and down the table. Given the large amount of data typically processed in DW queries, the number of leaps to orderly access the data may easily produce dramatic and unacceptable response time overheads.

When using data referencing, if the number of possible values to substitute a certain value has low cardinality (*e.g.* swapping values TRUE and FALSE for a boolean column with masking values 1 and 2) the reference lookup is fast but there is a security problem because the attacker can easily infer which is which. On the other hand, if the cardinality of the column to mask by referencing is high, then the number of rows to seek in the reference table will also be high, increasing security but decreasing response time for retrieving each value. Thus, there is always a tradeoff between security and performance to deal with: if the security level increases, performance typically decreases.

Substitution and shuffling techniques also present important security and performance issues. The main problem is that developing a value substitution function that uses one or more linear transformations is not secure because the attacker can build systems of equations and inference models to discover how the function masks a value. To deal with this problem, other bit-level manipulation operations need to be included, along with the execution of a significant number of rounds. These features are the basis for data encryption, which we explain in the next section. Data encryption solutions are the successors of the simpler forms of data masking substitution techniques and obey common principles and rules established by the security research and regulations communities and organizations.

For decision support purposes, in most DWs the user queries need to obtain a true and accurate result. Given this requirement, since perturbation techniques produce errors in data reconstruction, they should be avoided and are mainly inadequate from a data warehousing perspective.

Therefore, the following needs to be considered when applying data masking in DWs:

- Since it is not easy to ensure strong security strength (mainly when compared with encryption solutions), data masking has been considered a poor solution to protect data for real live DW databases, from the security perspective;
- The data masking routines provided by most standard commercial tools typically change data in an irreversible manner, *i.e.*, transform data in a way that makes it not possible to subsequently retrieve the original true values, making them useless for real live DW databases;
- On the other hand, solutions that allow retrieving the true original data mostly rely on cross-referencing actions, which imply huge table joins in DWs. Given the consequent high performance degradation, they have been discarded for use with real-live DWs;
- Given those security, usability and performance issues and drawbacks (assumed by the research and commercial communities), data masking is mostly recommended as an easy, efficient and fast solution in the development lifecycle of user applications and not for real-live databases. These facts have pushed data masking to a type of solution used mainly for testing software development rather than protect live sensitive data [Gartner, 2009; Huey, 2008; Natan, 2005; Oracle, 2005; Oracle, 2010a; Oracle, 2010c; Ravikumar *et al.*, 2011; Yuhanna, 2009].

The next section describes standard and state-of-the-art encryption techniques and discusses the issues involving the use of data encryption solutions in DWs.

2.3. Data Encryption

The high security requirements for confidentiality in many scenarios involving end-to-end data communication have led to the development of encryption algorithms. The frontier between data masking and encryption is often blurry, since they mainly aim to achieve the same purposes. However, while data masking can be simply considered as any action that changes a given value or set of values into another value or set of values that should not allow retrieving the original value or set of values by unauthorized users, encryption is mainly defined as a set of actions that obey a strict number of principles and rules defined and accepted by the

security communities and is always a reversible action [Vaudenay, 2006]. Encryption makes ground on cryptography, defined as applying a coding algorithm to a plaintext (alias original input value) that results in a ciphertext (encrypted output value), which allows reversible action in order to retrieve the plaintext once again [Vaudenay, 2006].

Typical encryption algorithms include iterative bit shifting and exclusive Or (XOR) operations executing in a predefined number of rounds. These operations rely on a key, which influences the "data mix" output of each round. The higher the key length and the number of rounds executed, the higher is the assumed security strength, given that the attacker typically needs to generate a large amount of possible key values and decryption rounds in order to break security [Elminaam *et al.*, 2010]. Thus, encryption is an advanced form of data masking, with well-accepted and well-defined assumptions and high complexity, in order to make it extremely difficult for attackers to break security when compared with simpler forms of data masking.

We consider describing and analyzing ciphers according to the principles following the Shannon Theory, where the Shannon Encryption Model is as illustrated in Figure 2-4 [Vaudenay, 2006]:

> "The purpose of encryption is to ensure communication secrecy. We assume that we want to communicate, which means to transmit information through a channel."

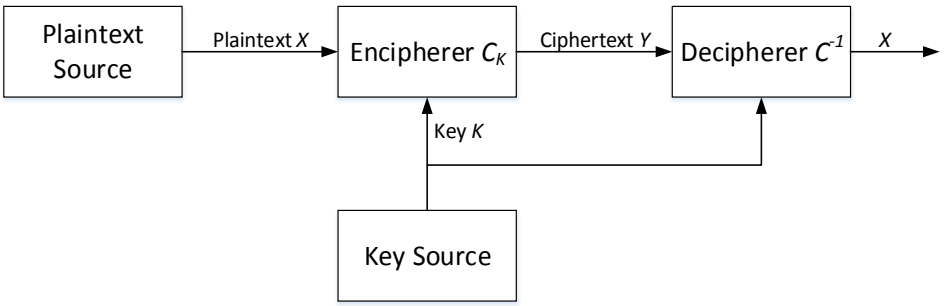


Figure 2-4. The Shannon Encryption Model (adapted from [Vaudenay, 2006])

Following the Shannon Theory, a cipher is given by:

1) A plaintext source (with its corresponding distribution);
2) A secret key or keys;
3) A ciphertext space;
4) A rule or set of rules represented as C_K, which transform any plaintext X with a key K into a ciphertext Y as $Y = C_K(X)$;
5) A rule or set of rules represented as C_K^{-1} which enables recovering plaintext X from the ciphertext Y using key K as $X = C_K^{-1}(Y)$.

Categorization methods for encryption techniques commonly used in data security are based on the form of the input data they operate on. The two types are *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. All intermediate blocks are called states. This transformation takes place 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. The fixed length is called the block size.

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 simply hands the same key to the stream cipher to produce an identical keystream and XORs it with the ciphertext, thus retrieving the original message.

In the following subsections, we shall describe the standard encryption techniques and algorithms as well as state-of-the-art encryption algorithms that have been specifically proposed by research to be applied in databases, and discuss the issues concerning their use in data warehousing environments.

2.3.1. Standard Encryption Techniques and Algorithms

Data Encryption Standard (DES). The Data Encryption Standard (DES) was the first encryption standard to be approved and recommended by the National Institute of Standards and Technology (NIST), and became a standard in 1977 [DES, 1977]. DES is a 64 bit block cipher, which means that data is encrypted and decrypted in 64 bit chunks, 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. Its algorithm is thus a set of permutations over the set of 64 bit block strings.

DES consists of a 16-round Feistel scheme, which is the most popular block cipher skeleton [Vaudenay, 2006]. It is fairly easy to use a random function in order to construct a permutation. In addition, encryption and decryption hardly require separate implementations. A Feistel scheme is a ladder structure which creates a permutation from a function. In each round, the input string is split into two parts of equal length, and the result of passing one part through a round function is XORed to the other part, then obtaining two parts which are then exchanged (except in the final round). The round function uses subkeys derived from a secret key.

The round function of DES has a 32-bit input, 48-bit subkey parameter input, and a 32-bit output. For every round, the 48-bit subkey is generated from a secret key by a key schedule. Basically, every 48-bit subkey consists of a permutation and a selection of 48 out of the 56 bits of the secret key. The round function is illustrated in Figure 2-5, consisting of [Vaudenay, 2006]:

- An expansion of the main input (one out of two input bits is duplicated in order to get 48 bits);
- A XOR with the subkey;
- Eight Substitution Boxes (S-boxes) which transform a 6-bit input into a 4-bit output; and
- A permutation of the final 32 bits.

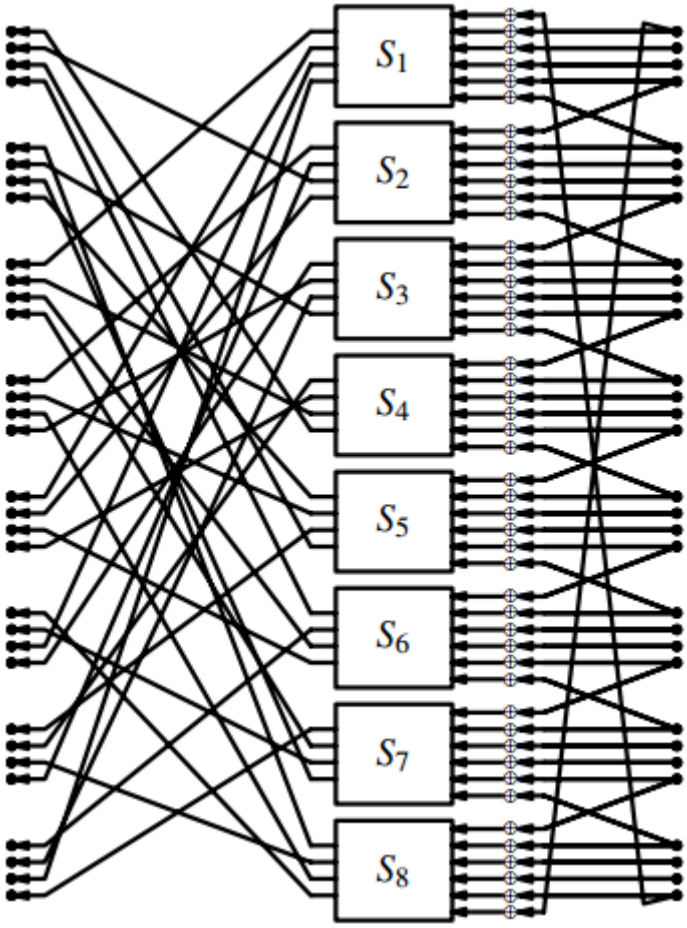


Figure 2-5. DES Round Function [Vaudenay, 2006]

As referred, the DES cipher uses eight S-boxes in its round function (S_1 to S_8). In cryptography, an *S-box* (Substitution-box) is a basic component of symmetric key algorithms which performs substitution. In block ciphers, they are typically used to obscure the relationship between the key and the ciphertext. In many cases, the S-boxes are carefully chosen to resist cryptanalysis. In general, an S-box takes some number of input bits, m, and transforms them into a number of output bits, n: an $m \times n$ S-box can be implemented as a lookup table with 2^m words of n bits each. Fixed tables are normally used, as in DES, but in some ciphers the tables are generated dynamically from the key; *e.g.* the Blowfish encryption algorithm [Radha and Kumar, 2005].

3DES. DES has been proven to be an insecure cipher [Kim *et al.*, 2010]. There has considerable controversy over its design, particularly in the choice of a 56 bit key [Nadeem and Javed, 2005]. As an enhancement of DES, the Triple DES (3DES) encryption standard was proposed [3DES, 2005]. In 3DES encryption algorithm 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. 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.

Advanced Encryption Standard (AES). After the DES standard was deemed as no longer appropriate, the US Government started a process leading to the Advanced Encryption Standard (AES). The AES is a symmetric block cipher algorithm defined in the Federal Information Processing Standard (FIPS) no. 197 [AES, 2001]. The AES algorithms are block ciphers with 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.

The AES consists of several rounds of a substitution-permutation network. Its design consists of writing 128-bit message blocks as a 4x4 square matrix of bytes. Encryption is performed through 10, 12 or 14 rounds depending on whether the key size is 128, 196 or 256 bits. Each round (except the final one) consists of four transformations:

1) *SubBytes*, a byte-wise substitution defined by a single table of 256 bytes;
2) *ShiftRows*, a circular shift of all rows (row i of the matrix is rotated by i positions to the left for $i = 0, 1, 2, 3$);
3) *MixColumns*, a linear transformation performed on each column and defined by a 4x4 matrix of $GF(2^8)$ elements (explained further);
4) *AddRoundKey*, a simple bitwise XOR with a round key defined by another matrix.

The final round is similar, except for MixColumns which is omitted. The round keys are generated by a separate key schedule.

More formally, one block s is encrypted by the following process, in which W is the output subkey sequence from the key schedule algorithm, as shown in Figure 2-6.

AES encryption(s, W)
1: **AddRoundKey**(s, W_0)
2: **for** $r = 1$ to $\mathrm{Nr} - 1$ **do**
3: **SubBytes**(s)
4: **ShiftRows**(s)
5: **MixColumns**(s)
6: **AddRoundKey**(s, W_r)
7: **end for**
8: **SubBytes**(s)
9: **ShiftRows**(s)
10: **AddRoundKey**(s, W_{Nr})

Figure 2-6. AES Step-by-Step Algorithm [Vaudenay, 2006]

The block s is also called state and represented as a matrix of terms $s_{i,j}$ for $i, j \in \{0, 1, 2, 3\}$. Each term is a byte, *i.e.*, elements of a set Z of cardinality 256. *SubBytes(s)* is defined as follows:

FOR i = 0 TO 3 DO
FOR j = 0 TO 3 DO
$s_{i,j}$ = S-box($s_{i,j}$)

Where S-box is the substitution table. Mathematically, it is a permutation of {0, 1, …, 255}. *ShiftRows*(*s*) is defined as follows:

REPLACE [$s_{1,0}$, $s_{1,1}$, $s_{1,2}$, $s_{1,3}$] by [$s_{1,1}$, $s_{1,2}$, $s_{1,3}$, $s_{1,0}$]
{ROTATE row 1 BY ONE POSITION TO THE LEFT}
REPLACE [$s_{2,0}$, $s_{2,1}$, $s_{2,2}$, $s_{2,3}$] by [$s_{2,2}$, $s_{2,3}$, $s_{2,0}$, $s_{2,1}$]
{ROTATE row 2 BY TWO POSITIONS TO THE LEFT}
REPLACE [$s_{3,0}$, $s_{3,1}$, $s_{3,2}$, $s_{3,3}$] by [$s_{3,3}$, $s_{3,0}$, $s_{3,1}$, $s_{3,2}$]
{ROTATE row 3 BY THREE POSITIONS TO THE LEFT}

Defining the set Z as the set of all the 256 possible combinations

$a_0 + a_1.x + a_2.x^2 + \ldots + a_7.x^7$

where a_0, a_1, a_2, …, a_7 are either 0 or 1 and x is a formal term. Elements of Z are thus defined as polynomials of degree at most 7. *AddRoundKey*(s, k) is defined as follows:

```
FOR i = 0 TO 3 DO
    FOR j = 0 TO 3 DO
        s_{i,j} = s_{i,j} ⊕ k_{i,j}
```

Here, the $\oplus$ operation over Z is defined as an addition modulus, *i.e.*

$$\left(\sum_{i=0}^{7} a_i \,.\, x^i\right) \oplus \left(\sum_{i=0}^{7} b_i \,.\, x^i\right) = \sum_{i=0}^{7} (a_i + b_i \; mod \; 2).\, x^i$$

Given that a multiplication $\times$ in Z defined as follows:

1) Perform the regular polynomial multiplication;
2) Make the Euclidian division of the product by the $x^8 + x^4 + x^3 + x + 1$ polynomial and take the remainder;
3) Reduce all its terms modulus 2.

This provides Z with the structure of the unique finite field of 256 elements. This finite field is denoted by GF(2^8). This means that any addition, multiplication, or division by any nonzero element of Z with the same properties always results in a regular number. Matrix operations with terms in Z can be further defined. Thus, *MixColumns*(s) can be defined as:

```
FOR i = 0 TO 3 DO
    LET v BE THE 4-DIMENSIONAL VECTOR WITH COORDINATES
    s_{0,i}, s_{1,i}, s_{2,i}, s_{3,i}

    REPLACE s_{0,i}, s_{1,i}, s_{2,i}, s_{3,i} BY THE COORDINATES OF M × v
```

Where M is a 4x4 matrix over Z defined by

$$M = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix}$$

The substitution table S-box is defined by the inversion operation x^{-1} (except for $x = 0$, which is mapped to zero) in the finite field GF(2^8). This operation has good nonlinear properties [Vaudenay, 2006].

AES is considered fast and able to provide stronger encryption, compared to other well-known encryption algorithms such as DES [Nadeem and Javed, 2005]. Brute force attack (in which the attacker tries all the possible key combinations to unlock the encryption) is the only known effective attack known against it.

2.3.2. Other Encryption Techniques and Algorithms

Besides the existence of standard encryption algorithms, the data security research community has also proposed several solutions for encrypting databases.

One of the main issues in database performance due to using encryption is the inability to effectively manage useful indexing, since encryption changes data values and thus renders the traditional index building as useless. One way to deal with this is to ensure order preservation of the generated encrypted values. Based on this principle, several approaches have been proposed in order to enable direct querying against encrypted data.

Order Preserving Encryption Scheme (OPES). In [Agrawal *et al.*, 2004] an Order Preserving Encryption Scheme (OPES) for numeric data is proposed, flattening and transforming plain text distributions onto target distributions defined from value-based buckets, given the attribute's domain values. This solution allows any comparison operation to be directly applied on encrypted data, such as equality and range queries, as well as SUM, AVG, MAX, MIN and COUNT queries. The authors define a threat model with the following assumptions, given the transparent encryption setting shown in Figure 2-7:

- The storage system used by the database software is vulnerable to compromise;
- The database software (DBMS) is trusted;
- All disk-resident data (alias data-at-rest) is encrypted.

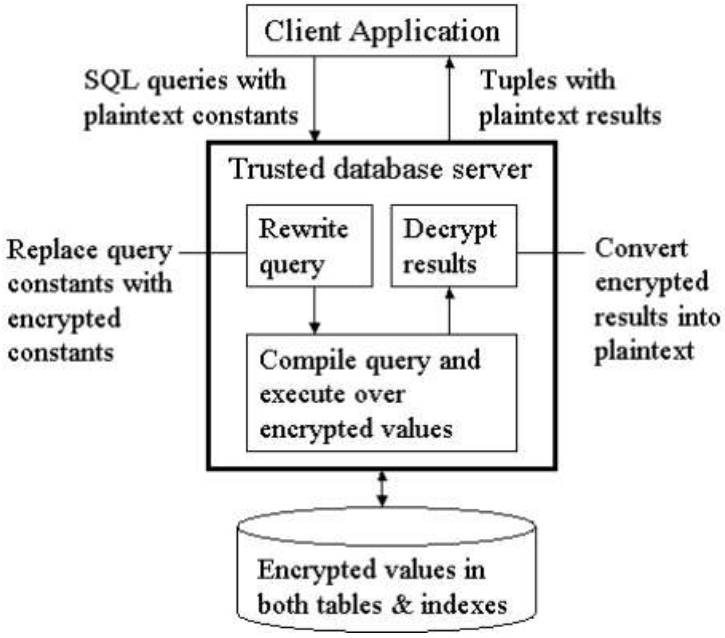


Figure 2-7. Transparent Encryption Setting for OPES [Agrawal *et al.*, 2004]

OPES works as a three stage process [Agrawal *et al.*, 2004]:

1) *Model:* The input and target distributions are modeled as piece-wise linear splines;
2) *Flatten:* The plaintext database *P* is transformed into a "flat" database *F* such that the values in *F* are uniformly distributed;
3) *Transform:* The flat database *F* is transformed into the ciphered database *C* such that the values in *C* are distributed according to the target distribution.

The results of query processing over data encrypted by OPES are exact. They neither contain any false positives nor miss any answer tuple. OPES also handles updates gracefully; a value in a column can be modified or a new value can be inserted without requiring changes in the encryption of other values. The basic idea of OPES is to take as input a provided target distribution and transform the plaintext values in such a way that the transformation preserves the order while the transformed values follow the target distribution.

Executing SQL over Encrypted Data in the Database-Service-Provider Model. A similar solution for processing queries without decrypting data was proposed earlier by [Hacigumus *et al.*, 2002], which uses the *"Database as a Service"* provider model based on Internet availability. The authors focus on assuring the owner of the data that the data stored in the service-provider site is protected against those service providers themselves, if they cannot be trusted, by keeping data always encrypted and executing SQL queries directly against the encrypted data. To accomplish this, they propose splitting the computation of the queries into two phases: the first phase computes as much as possible against the encrypted data at the service provider server without having to decrypt it, and a second phase which processes the results obtained in the first phase at the client. The data in the service provider is protected because the decryption only occurs at the client side. The service-provider architecture for this solution is shown in Figure 2-8.

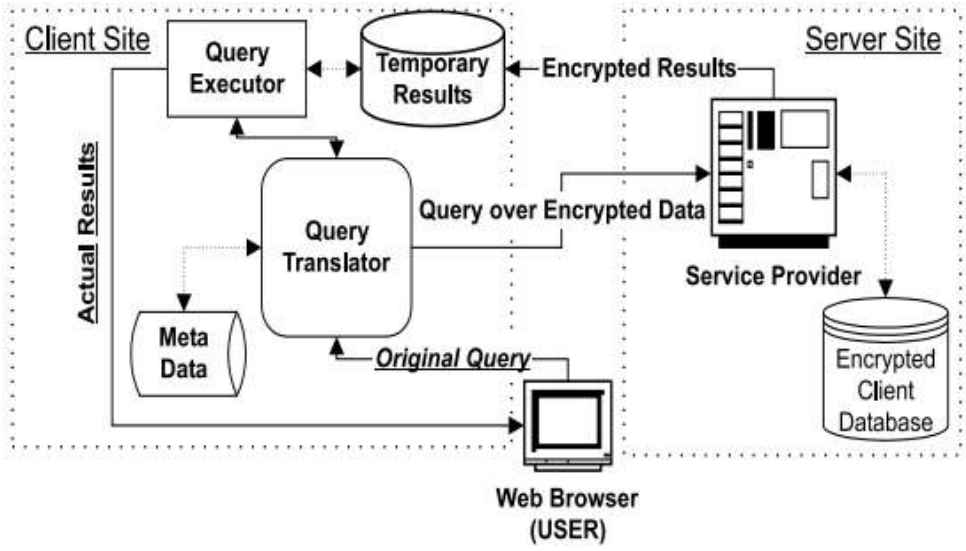


Figure 2-8. Encryption-as-a-Service Service-Provider Model [Hacigumus *et al.*, 2002]

The encrypted data is stored at the service-provider according to the following:

- For each relation $R(A_1, A_2, \ldots, A_n)$ of the original plaintext data, an encrypted relation $R^s(etuple, A_1^s, A_2^s, \ldots, A_n^s)$ is stored on the service-provider server;
- The attribute *etuple* stores an encrypted string that corresponds to a tuple in relation *R*;
- Each attribute A_i^s corresponds to the index for attribute A_i that will be used for query processing at the server.

Thus, each original unencrypted table is mapped to an encrypted table at the service-provider server. To accomplish this, they define a series of partitions on that server for each attribute, given the domain values of attributes $R.A_i$, define an identification function to assign an identifier to each partition of each attribute, and finally define a mapping function to those partitions which ensures order-preservation of the attribute's original values. A practical evolution of the initial proposal was published in [Hacigumus *et al.*, 2004], based on the same model. In this work, the authors focus on improving their transformation and mapping functions, by exploring homomorphism techniques to support aggregation in relational databases against encrypted data without decryption in the presence of logical predicates.

Encryption in Column-Oriented DBMS. The authors of [Ge and Zdonik, 2007] propose a lightweight database encryption scheme for column-stores in DWs with trusted servers, named FCE. This technique introduces low decryption overhead to enable making comparisons of ciphertexts and hence makes indexing operations fast. The authors also propose a relaxed measure of security to demonstrate FCE's security strength based on information theoretic concepts. Using this same measure, they also show that order-preserving encryption techniques are insecure under straightforward attack scenarios.

Tiny Encryption Algorithm (TEA). In an effort to trying to simplify encryption algorithms, the Tiny Encryption Algorithm (TEA) [Wheeler and Needham, 1994] was proposed in 1994. This simple algorithm uses a larger number of rounds against a small number of data transformations than, rather than a more complex set of transformations with few rounds.

The main objective of the authors of the TEA was to provide a very simple encryption algorithm instead of a complicated one. The authors claim that

it uses little setup time and does a weak non-linear iteration a sufficient number of rounds that makes it secure enough. There are no preset tables or long setup times. It assumes 32 bit words and the authors suggest executing 32 rounds. The TEA schema is shown in Figure 2-9.

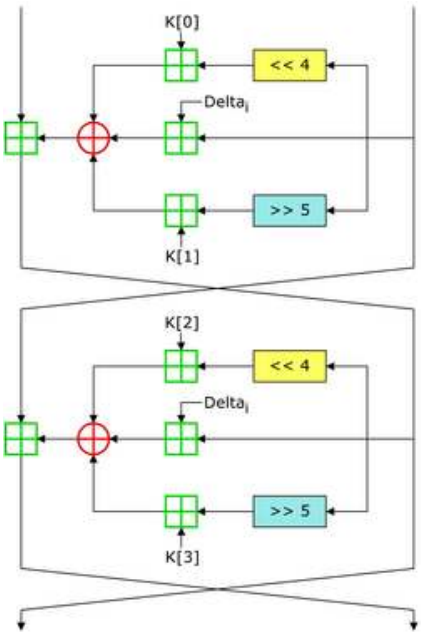


Figure 2-9. TEA Schema

The proposed encoding routine, written in C, using four 32 bit keys *k*[0] to *k*[3] (making up a 128 bit key), executing 32 rounds to encrypt 64 bits of data in *v*[0] and *v*[1], is [Wheeler and Needham, 1994]:

```
void code(long* v, long* k) {
unsigned long y = v[0],z = v[1], sum = 0, /* set up */
         delta = 0x9e3779b9, /* a key schedule constant */
         n=32;
while (n-->0) { /* basic cycle start */
   sum += delta ;
   y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
   z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
} /* end cycle */
v[0]=y ; v[1]=z ;
}
```

TEA is a Feistel type routine although addition and subtraction are used as the reversible operators rather than XOR. The routine relies on the alternate use of XOR and ADD to provide nonlinearity. A dual shift causes all bits of the key and data to be repeatedly mixed. The top five and bottom four bits are probably slightly weaker than the middle bits. These bits are generated from only two versions of z (or y) instead of three, plus the other y or z. Thus, the convergence rate to even diffusion is slower. However, the shifting evens this out with a possible delay of one or two extra cycles [Wheeler and Needham, 1994].

Blowfish Encryption Algorithm. The Blowfish encryption algorithm [Schneier, 2013] is one of the most common public domain encryption algorithms. Blowfish is a variable length key, 64 bit symmetric block cipher. This algorithm was first introduced in 1993. Each round of the algorithm consists of a key-dependent permutation and a key-and-data-dependent substitution. All operations are based on XORs and additions on 32-bit words. The key has a variable length (with a maximum length of 448 bits) and is used to generate several subkey arrays. It has been extensively analyzed and deemed "reasonably secure" by the cryptographic community. Though it suffers from weak keys problem, no attack is known to be successful against it [Nadeem and Javed, 2005]. A graphical representation of the Blowfish algorithm can be seen in Figure 2-10.

As shown in Figure 2-10, a 64-bit plaintext message is first divided into 32 bits. The "left" 32 bits are XORed with the first element of a *P-array* to create a new value named as P', run through a transformation function called F, then XORed with the "right" 32 bits of the message to produce a new value named as F'. F' then replaces the "left" half of the message and P' replaces the "right" half, and the process is repeated 15 more times with successive members of the *P-array*. The resulting P' and F' are then XORed with the last two entries in the *P-array* (entries 17 and 18), and recombined to produce the 64-bit ciphertext.

A graphical representation of the F transformation function is shown in Figure 2-11. The function divides a 32-bit input into four bytes and uses those as indices into an *S-array*. The lookup results are then added and XORed together to produce the output.

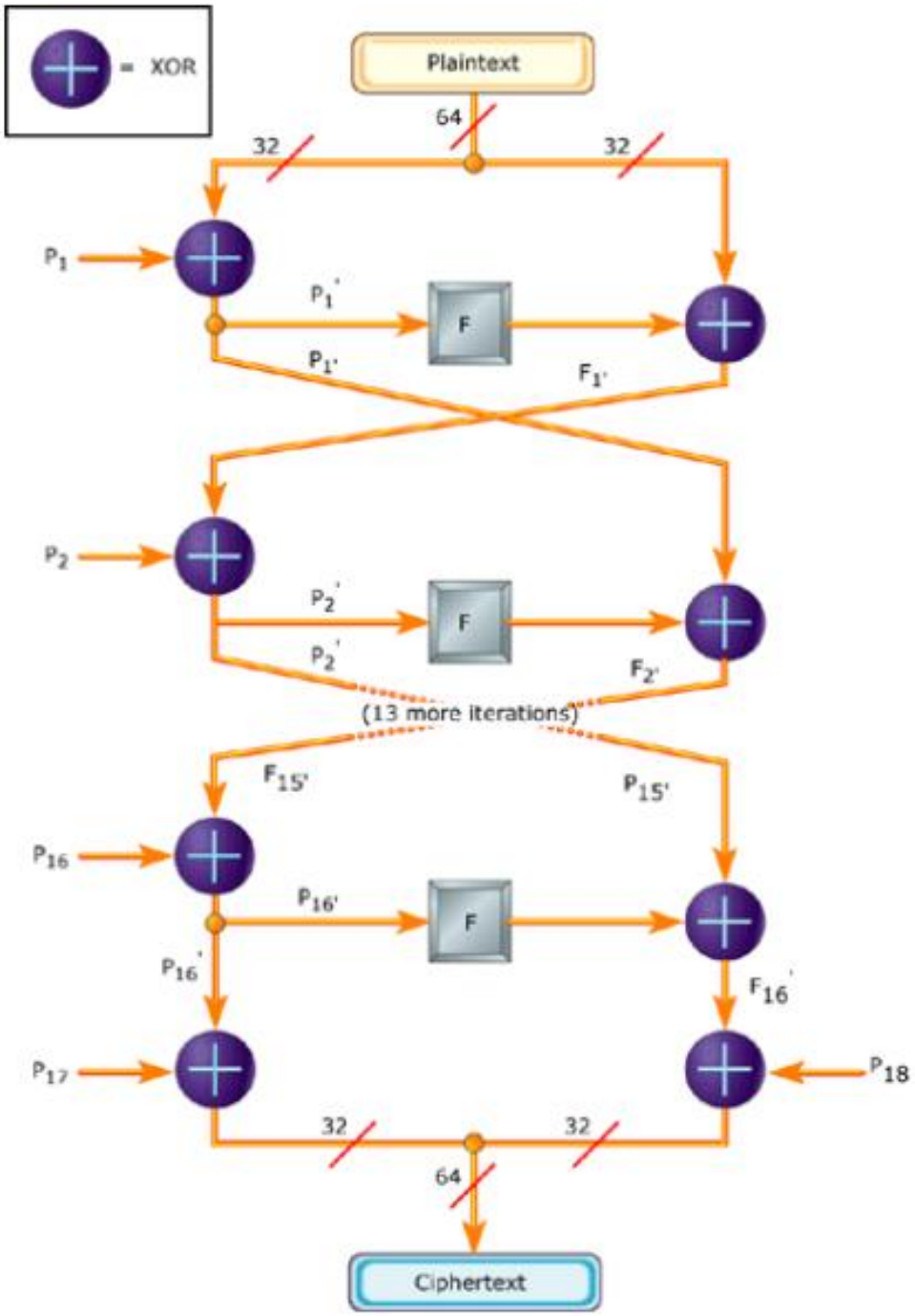


Figure 2-10. The Blowfish Algorithm

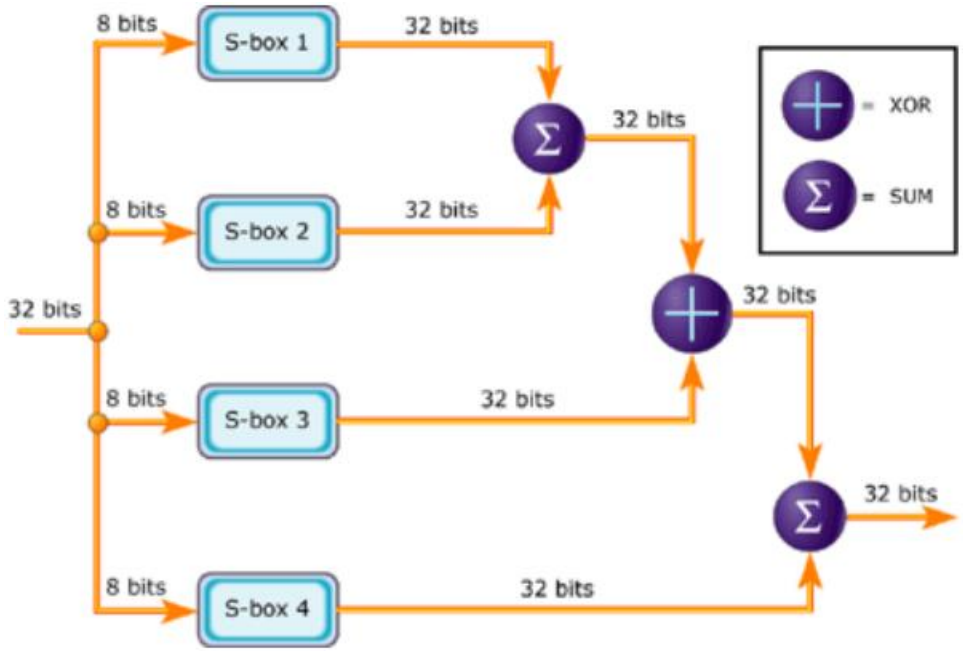


Figure 2-11. The Blowfish Transformation Function (F)

The *P-array* and *S-array* values used by Blowfish are precomputed based on the user's key. In effect, the user's key is transformed into the *P-array* and *S-array*; the key itself may be discarded after the transformation. The *P-array* and *S-array* need not be recomputed (as long as the key doesn't change), but must remain secret. The *P* and *S-arrays* are summarized as follows (according to [Schneier, 2013]):

- *P* is an array of eighteen 32-bit integers;
- *S* is a two-dimensional array of 32-bit integer of dimension 4x256;
- Both arrays are initialized with constants, which happen to be the hexadecimal digits of π (a pretty decent random number source);
- The key is divided up into 32-bit blocks and XORed with the initial elements of the *P* and *S* arrays. The results are written back into the array. A message of all zeros is encrypted; the results of the encryption are written back to the P and S arrays. The P and S arrays are now ready for use.

Snuffle (alias Salsa20) Encryption Algorithm. Recently, the Snuffle 2005 encryption algorithm (also known as Salsa20) was proposed [Bernstein, 2005; Bernstein, 2008]. The goal of Salsa20 is to produce a 64-byte block given a key, nonce[5] and block counter. The author recommends executing a number of 20 rounds, although 8 or 12 rounds are acceptable when required to gain speed against sacrificing some security. This solution can be seen as a 256-bit stream cipher and is based on a hash function with a long chain of simple operations, instead of a short chain of more complex operations (typical in standard encryption algorithms), on 32-bit words:

- 32-bit additions, producing the sum $a + b \bmod 2^{32}$ of two 32-bit words a, b (which breaks linearity over $Z/2$);
- 32-bit exclusive-or (XOR), producing $a \oplus b$ of two 32-bit words a, b (which breaks linearity over $Z/2^{32}$); and
- Constant-distance 32-bit rotation, producing the rotation $a <<< b$ of a 32-bit word a by b bits to the left, where b is constant (diffusing changes from high bits to low bits).

The author of Salsa20 states that although these operations may be considered too simplistic, they can easily emulate any circuit and are therefore capable of reaching the same security level as any other selection of operations. The real question for the cipher designer is whether a different mix of operations could achieve the same security level at higher speed.

Salsa20 expands a 256-bit key and a 64-bit nonce (unique message number) into a 2^{70}-byte stream. It encrypts a b-byte plaintext by XORing the plaintext with the first b bytes of the stream and discarding the rest of the stream. It decrypts a b-byte ciphertext by XORing the ciphertext with the first b bytes of the stream. There is no feedback from the plaintext or ciphertext into the stream.

Salsa20 generates the stream in 64-byte (512-bit) blocks. Each block is an independent hash of the key, the nonce, and a 64-bit block number; there

[5] In cryptography, a nonce is an arbitrary number used only once in a cryptographic communication. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replaying attacks. [Vaudenay, 2006]

is no chaining from one block to the next. The Salsa20 output stream can therefore be accessed randomly, and any number of blocks can be computed in parallel.

There are no hidden preprocessing costs in Salsa20. In particular, Salsa20 does not preprocess the key before generating a block; each block uses the key directly as input. Salsa20 also does not preprocess the nonce before generating a block; each block uses the nonce directly as input.

This solution is relatively simple when compared with other standard encryption algorithms such as AES and has been recognized by the cryptology research community as an interesting alternative to those algorithms in contexts where speed is more important than confidence [Tsunoo *et al.*, 2007; Bernstein, 2008].

2.3.3. DBMS Data Encryption Packages

Many DBMS vendors such as Microsoft SQL Server and Oracle TDE provide built-in standard encryption packages. These routines run in the DBMS kernel and are optimized to work against their data structures and across a large diversity of platforms.

Oracle has developed TDE (Transparent Data Encryption) [Oracle, 2005; Oracle, 2010a] incorporating both AES and 3DES, providing column and tablespace encryption. These routines can be used transparently without requiring user application source code modifications. As Oracle, Microsoft SQL Server also provides column and datafile 3DES and AES encryption routines.

When using Oracle TDE tablespace encryption, all data in the tablespace's physical datafiles is encrypted and almost no storage space overhead is generated. 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 each column is done by using an optional feature (SALT) in the encryption, which implies adding 16 bytes of the storage space per encrypted column to each row. If the NO SALT option is 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 its security strength. Tablespace encryption uses only the database master

key and the tablespace's encryption key, which makes its security level lower than column encryption.

Encryption in Oracle TDE is transparently handled, including index operations and table joins, even if the columns for the join condition are encrypted. In TDE column encryption, the index needs to be a normal B-Tree index. With TDE column encryption, the data remains encrypted in the RAM (in the database cache), but with TDE tablespace encryption the Oracle database will automatically decrypt data before it arrives in database memory (SGA). This means that all data in the SGA is always decrypted, which must be considered a weakness in security for this type of encryption.

2.3.4. Using Data Encryption in Data Warehouses

One of our main objectives in this thesis is to discuss if the commonly used data encryption algorithms are too slow for DWs. We are not interested in discussing in detail each step of each algorithm focusing on their security, but rather to compare and analyze the generic guidelines of the different types of encryption algorithms and how their performance is affected as well as how it affects DW performance.

When processing SQL on encrypted data, there are many database performance issues that arise. For example, certain basic queries are not supported, *i.e.*, they cannot be executed because they cannot be handled by the encryption/decryption schemas, or their execution is too inefficient (especially joins and ordering operations), resulting in the introduction of large response time overhead. Regarding this last issue, if no order preserving scheme is ensured by the encryption solution indexing becomes mostly useless, with its corresponding impact in database performance.

Many decision support workloads are based on actions in which the end-user interacts with the system, like performing an OLAP analysis through *ad hoc* querying or performing drill-down or roll-up reporting. When performing this type of analysis, the end user is typically in front of a computer waiting for the system to answer the query; therefore, if the DBMS is slow the end-user can lose interest in the business analysis, leave the query running and forget the business question s/he originally wanted or feel exasperated by having to wait for a long time to get the answer

[Castro, 2009]. This may compromise the acceptability and credibility of the DW system among its users and ultimately, jeopardize its usefulness.

As we have mentioned earlier, encryption algorithms typically execute a significant number of bit management operations using one or more encryption keys. In what concerns performance issues, the quality of each set of operations in achieving the intended "data mix" affects how fast the algorithm can execute. When comparing encryption algorithms referring to what, how and how many operations they execute, most encryption algorithms such as AES carry out considerably short chains of complex operations, while other hash-based solutions such as Salsa20 execute longer chains of simpler operations.

The argument in favor of using complicated operations such as the use of S-boxes is that they provide a large amount of mixing at reasonable speed on many CPUs, and thus achieve many desired security levels more quickly than simple operations on those CPUs; a single table lookup can mangle its input quite thoroughly – more thoroughly than a chain of simple integer operations – in fewer rounds. This provides a large amount of mixing at reasonable speed on many CPUs, reaching many desired security levels more quickly than simple operations. The counterargument is that potential speedup is fairly small and is accompanied by huge slowdowns on other CPUs.

On the other hand, simple operations such as bit additions and XORs are consistently fast, independently from the CPU. It is also not obvious that a series of S-box lookups (even with rather large S-boxes, as in AES, increasing L1 cache pressure on large CPUs and forcing different implementation techniques for small CPUs) is generally faster than a comparably complex series of simpler integer operations.

In what concerns the use of packaged encryption routines in DBMS', Oracle recommends the use of tablespace encryption when there is no way of determining which columns are sensitive and which are not, or when the majority of the data in the tablespace is sensitive [Oracle, 2010a]. They state that column encryption should be preferred when a small number of well defined columns are sensitive. This last scenario is typical in data warehousing environments, which makes column encryption the recommended solution according to Oracle. However, as we have shown

in [Santos *et al.*, 2012a], when applying column encryption in DWs the storage overhead will be very significant.

On the other hand, since DWs are business knowledge data sources by nature, we can assume that most of its data is sensitive. In this sense, we may also state that TDE tablespace encryption should also be highly considered. Nevertheless, data coming from tablespace encryption is made immediately transparent once that data is loaded into the SGA (located in RAM), making decryption straightforward and minimizing resource consumption, but also allowing third party access to the real data, lowering the level of security. Although we are focused on performance, we believe this is a very relevant drawback in data security and that it should not be considered a good solution, given the risk of data exposure.

There are also many situations where certain users or applications may require querying data that is less sensitive or not sensitive at all to the business. Since tablespace encryption encrypts the entire content of the tablespace, in these scenarios using tablespace encryption would require giving those users or applications the encryption keys or passwords that allow them to access the data. Using column encryption would enable to keep the columns that store less sensitive data unencrypted is this desirable, avoiding the disclosure of security keys or passwords to ensure the access to that data. Furthermore, tablespace encryption adds computational overhead to decrypt less sensitive or non-sensitive columns for query processing, that wouldn't be selected for encryption when using column encryption.

Other encryption solutions proposed by research work such as [Agrawal *et al.*, 2004] distribute data in well-defined groups to allow direct operations on encrypted data. However, the impact in performance produced by these solutions, in response time and storage space overhead, depends on the skew in the target distributions, which can be a very serious problem in DWs. There is no easy way around this. The proposal from [Hacigumus *et al.*, 2002] also suffers from the same problem.

The lightweight encryption in column-oriented DBMS proposed in [Ge and Zdonik, 2007] aims on providing low decryption overheads. However, their experiments show at least 50% of response time overhead to retrieve the encrypted tuples, which is still extremely high for many DW scenarios,

such as long running queries. The fact that is aimed at column-DWs also narrows its applications.

Topologies involving middleware solutions such as [Radha and Kumar, 2005] typically request the encrypted data from the database *a priori* and execute the decrypting actions themselves locally. The proposal in [Radha and Kumar, 2005] aims to ensure efficient query execution over encrypted databases, by evaluating most queries at the application server and retrieving only the necessary records from the database server. Only one query (Q6) of the TPC-H benchmark is used in their experimental evaluation, against a very small data subset (ranging from 10MB to 50MB, where query execution time rises up to 5 times for the last).

This dataset size cannot be considered realistic for DWs, given its typical very large sized databases. 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.

After considering the referred issues that influence performance (and security tradeoffs) of the described encryption solutions and to finish this discussion, we have come to the following conclusions:

- Both standard encryption algorithms and specific research database encryption solutions show large performance overheads;
- The type and number of operations for producing the "data mix" output in each round of the algorithm, the length of the used encryption keys, the size of the input and output blocks, and the number of rounds to execute, are all variables that affect both security and performance;
- In many software implementations of the security techniques, the CPU architecture also varies the performance outcome;

- Typically, most secure encryption algorithms will execute between 8 and 20 rounds against 64, 128 bit (or more) sized blocks, using a 128 or 256 bit key;
- Encryption algorithms which make use of chains of simple operations such as bit additions and XORs scale better and have reduced CPU dependency than solutions that make use of more complex operations such as S-box lookups;
- Salsa20 seems to provide consistent speed in a wide variety of applications across a wide variety of platforms. It is faster and simpler than the complex-operations approach of the standard algorithms 3DES and AES, while granting significant security strength. However, most commercial vendors just include AES and 3DES routines. The AES became a standard only after a five-year long standardization process that included extensive benchmarking on a variety of platforms ranging from smart cards to high end parallel machines. Thus, the adoption of encryption standards is probably only due to legal impositions and public reliability issues, given that only AES and 3DES are the current well-accepted encryption standards.
- All major DBMS provide encryption to be used transparently by user applications;
- When using tablespace encryption, the requested data is decrypted and loaded into RAM memory (in the database cache) as clear text, while column encryption does not and is thus more secure;
- Tablespace encryption does not create significant storage space overhead, while column encryption does;
- Despite the well-known pros and cons, the best choice between tablespace encryption and column encryption isn't obvious;
- Leading DBMS use standard encryption algorithms AES and 3DES, producing alphanumeric or binary values as a result of the encryption process, even for numerical-typed attributes;
- In DWs, transporting encrypted data to third party decrypting agents would create unbearable communication bandwidth consumption and compromise throughput.

2.4. Database Intrusion Detection Systems

Generically, intrusion detection (ID) is defined as the process of monitoring the events occurring in a computer system and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable user policies, or standard security practices [Scarfone and Mell, 2007]. ID systems are typically classified in two main types, depending on the environment in which they operate:

1) *Network-based ID systems*, which perform surveillance using network traffic or other network-based data;
2) *Host-based ID systems*, which are located at the host that is aimed to be protected, analyzing the activity that happens there.

In this thesis, we specifically focus on *Database Intrusion Detection Systems* (*DIDS*), which are host-based ID systems that analyze user actions occurring at the database level in order to detect (and eventually stop or prevent) intrusion actions. This section characterizes the way a typical ID system operates and presents a descriptive analysis of selected samples from each different type of approach and/or technique that can be applied in DIDS, in order to characterize the broad scope of existing solutions.

2.4.1. How Intrusion Detection Systems Operate

The main requirements that ID systems are required to cope with are:

- The quest for adequately defining and building profiles that accurately represent "normal"/"intrusion-free" behavior or workloads, as well as identifying attack signatures;
- Given those profiles and/or attack signatures, define which behavioral features as well as techniques and models that maximize the performance and accuracy of the intrusion detection processes;
- Reporting system status to security staff and notifying them about generated alerts;
- Promote a way of stopping or preventing the attack whenever an intrusion alert is raised (this feature may or not be present in the ID system; if it is the case, literature often refers to the ID system as an Intrusion Detection and Response System, or an Intrusion Detection and Prevention System).

The typical components of an ID system according to [Scarfone and Mell, 2007] are shown in Figure 2-12 and are described as:

- A *Sensor* or *Agent*, which are responsible for capturing both the information relating to the ID features that is necessary for building the "normal"/"intrusion-free" profiles and/or attack signatures, as well as the required information to execute the ID processes;
- A *Management Server*, which is a centralized device that receives the information from the Sensor or Agent and manages the profile building processes and the intrusion detection and response processes of the ID system;
- A *Repository*, for storing the behavior profiles and/or attack signatures, activity logs, generated alert information and other relevant data that is useful to the ID system; and
- A *Console*, which is the interface responsible for the interaction between security managers/staff and the ID system, *i.e.*, it enables a mean for configuring the ID system and displays the required information concerning the behavior profiles, system status, generated alerts, etc.

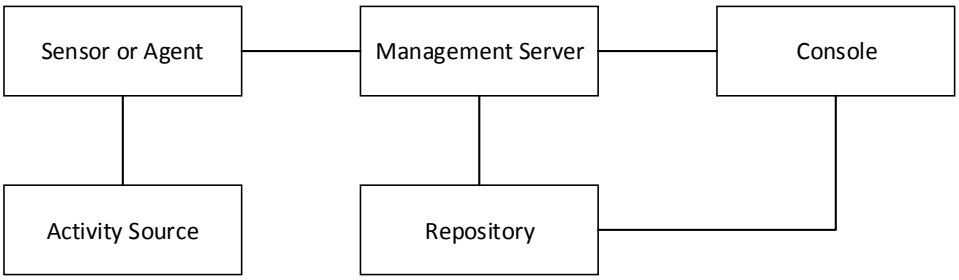


Figure 2-12. Typical ID System Architecture (adapted from [Scarfone and Mell, 2007])

Referring to Figure 2-12, the *Activity Source* is where the activity that should be analyzed is generated; in a DIDS, it can represent a user or an applications that generates SQL workloads to execute against the DW. This activity is then captured by the *Sensor* or *Agent* and sent on to the *Management Server* either to build behavior profiles and previously define attack signatures (if the activity is considered "intrusion-free" and the ID system is in the learning phase) or to perform intrusion detection and consequent alert generation (and/or response actions if this is required). The *Management Server* will both read and write data from the *Repository* in order to retrieve or store all relevant information accordingly with what it needs to do. The *Console* allows the security managers/staff to configure the ID system and retrieve all relevant information for assessing system status, user behavior and alert notifications.

In DIDS systems there is typically a learning or training phase (*i.e.*, previous to intrusion detection), in which database and/or user logs assumed as having "normal" or intrusion-free activity are used in order to build the user behavior profiles and/or define attack signatures[Newman, 2011]. After this learning phase, the intrusion detectors match user actions against those profiles and/or attack signatures to find significant deviations which are signaled as potential intrusions.

From the intrusion perspective, an intruder in a data warehousing environment can be one of the following [Treinen and Thurimella, 2006]:

- An *authorized user*, which is someone belonging to the enterprise that has regular access to authorized database interfaces and acts with malicious intent (also commonly referred to as the *insider threat*);
- A *masqueraded user*, which is someone that obtains the credentials of an authorized user and impersonating that user takes control of an authorized interface (which refers to the *insider threat* when the attacker is someone from within the enterprise but without regular authorized database access, and refers to an *outsider threat* when it someone from outside the enterprise that manages to obtain the credentials);
- An *external attacker* (commonly referred to as the *outsider threat*), which is someone from outside the enterprise that is able to bypass the

database security and gain direct database access using SQL injection[6] or other exploiting techniques;

- Any combination of the above.

Considering the possible intruders' intentions, there are mainly three types of attacks mobilized against DWs [Douligeris and Mitrokotsa, 2004]:

- *Attacks aiming at corrupting data (integrity attacks).* In these types of attack, the intruder seeks access to the database for executing actions that compromise its integrity, such as corrupting or deleting the data in a given database object (*e.g.* such as a table or view);
- *Attacks aiming at stealing information (confidentiality attacks).* In these attacks, the intruder is focused on breaking confidentiality issues, such as stealing business information, rather than damaging data;
- *Attacks aiming at making the DW unavailable (availability attacks).* These attacks aim on making database services unavailable to users, *i.e.*, they are mainly Denial of Service (DoS) attacks (*e.g.* flooding database services and bandwidth with a large number of requests, halting or crashing database server instances, deleting database objects, etc).

The way how ID processes are designed to operate is mainly based on two approaches, depending on what they intend to search for:

1) *Misuse or signature-based detection*, which searches for well-known attack patterns and signatures defined *a priori* to the attack itself; and

[6] SQL injection is a type of attack executed through means of a third party interface (*e.g.* a web application) in which the attacker appends malicious code to an authorized command that will be executed on behalf of that interface. SQL injection is often considered as a particular form of attack on its own, following very well-defined guidelines. Although the actions performed through SQL injection can also be detected by DIDS, the forms of detecting SQL injection attacks have been extensively studied and belong to a category of security mechanisms that are differentiated appart from those that we intend to focus on in this thesis. As a reference, the work in [Halfond *et al.*, 2006; Kim, 2011; Kindy and Pathan, 2012] presents detailed surveys and countermeasures on SQL injection.

2) *Anomaly detection*, which searches for deviations from typical user behavior by matching their actions against assumed "intrusion-free" profiles that significantly represent that typical user behavior.

The first approach is mainly efficient against previously well-known and expected intrusion actions. However, they are mostly incapable of acting against intrusions that reveal new forms of attack or malicious actions that seem "normal" (which, in many cases, refer to the insider threat), opening a much wider spectrum of analysis possibilities that results in a threat that is much harder to tackle and mitigate. Given the published work that refers trends indicating an increase of attacks referring to the insider threat [Jabbour and Menasce, 2009], to overcome those issues anomaly detection techniques have been proposed in the most recent DIDS.

In the past, several types of intrusion detection techniques and methods have been proposed to build behavior profiles and perform intrusion detection processes that may be used in DIDS, which we shall describe and discuss in the following subsections.

2.4.2. Intrusion Detection Techniques

The most common way to distinguish between distinct ID techniques is to classify the way they select and analyze the features used for building user profiles and execute the intrusion detection processes. In this subsection, we distinguish and describe a set of main types/classes of analysis techniques, referring prominent research work in each of these classes.

Temporal Analysis. These techniques focus on temporal features such as the time span between user actions and the duration of those actions.

The approach in [Lee *et al.*, 2000] uses a mean and standard deviation model built from time signatures to check for outliers within a predefined range in real-time database systems. This solution considers a transaction as a set of read or write actions for each data object which is executed in predefined update time periods. For example, the update of a temporal data object (event) can trigger a rule such that the update time is checked against the expected update time (condition) and rejects the update (action) if the predicate returns false, considering it an intrusion.

The training period occurs until a significant mean with 99% confidence level of a normal distribution is obtained for each object/update pair.

Database behavior is monitored by sensors at the transaction level, which are assumed to be small in size and have predefined semantics such as write-only operations and well-defined data access patterns. If a transaction tries to update a temporal data object that has already been updated in that period, an alarm is raised.

Dependency and Relation Analysis. Intrusion detection techniques based on dependency and relation analysis determine dependencies and/or relations among the distinct sets of user actions and/or accessed data in order to determine which columns, rows, tables, etc. and/or which commands are usually issued or processed together.

For example, the DEMIDS system [Chung *et al.*, 1999] builds user profiles based on their activity by determining frequent itemsets from feature/value pairs and computes distance measures of user activity against the learnt frequent itemsets to detect intrusions, given a threshold. The features are typically based on the syntactical analysis of user commands, where the itemset domains are the sets of attributes issued together.

Another approach using frequent itemset mining is presented in [Zhong and Qin, 2004]. This approach summarizes each user command into a tuple <Op, F, T, C> where Op is the type of SQL command (insert, select, etc), F is the set of attributes, T is the set of tables, C is the constrained condition set. An algorithm mines user query profiles using these tuples, based on the pattern of the submitted queries at the transaction level. The algorithm adapts the support and confidence of association rule mining by adding query structure and attribute relations to the computation.

The Role-Based Access Control (RBAC) DIDS proposed in [Kamra *et al.*, 2008] improves a previous approach [Bertino *et al.*, 2005b] using features named quiplets for summarizing each user command. Considering a generic command:

```
SELECT {Target-List}
  FROM {Relation-List}
 WHERE {Qualification}
```

A quiplet is defined as (C, PR, PA, SR, SA) where C is the SQL main command (insert, select, etc.), PR is the Projection-Relation information, PA is the Projection-Attribute information, SR is the Selection-Relation

information, and `SA` is the Selection-Attribute information. The authors define three types of quiplets with different granularities: given a relation (alias table) `R1` with attributes `A1`, `B1`, `C1`, `D1` and a relation `R2` with attributes `A2`, `B2`, `C2`, `D2` and given the user command `SELECT R1.A1, R1.C1, R2.B2, R2.D2 FROM R1, R2 WHERE R1.B1 = R2.B2`, will generate, as shown in Figure 2-13:

1) The coarse c-quiplet (`select, <2>, <4>, <2>, <2>`)
2) The medium m-quiplet (`select, <1,1>, <2,2>, <1,1>, <1,1>`)
3) The fine f-quiplet (`select, <1,1>, <[1,0,1,0], [0,1,0,1]>, <1,1>, <[0,1,0,0], [0,1,0,0]>`)

SQL Command	c-quiplet	m-quiplet	f-quiplet
SELECT $R_1.A_1, R_1.C_1, R_2.B_2, R_2.D_2$ FROM R_1, R_2 WHERE $R_1.B_1 = R_2.B_2$	select$< 2 >< 4 >< 2 >< 2 >$	select $< 1,1 >< 2,2 >$ $< 1,1 >< 1,1 >$	select $< 1,1 >$ $< [1,0,1,0], [0,1,0,1] >$ $< 1,1 > [0,1,0,0], [0,1,0,0]$

Figure 2-13. The quiplet construction process [Kamra *et al.*, 2008])

For anomaly detection when the database has role-based users (*i.e.*, it is possible to link each user action to a given role), a Naïve Bayes Classifier (NBC) is used as follows:

- For all queries in the audit logs, and for each role, the classifier for each type of quiplet is built (training phase);
- For each submitted query, if any of its classifiers is different from the ones in its roles, the action is considered an intrusion and an alert is generated (testing phase).

If role-based access policies are not implemented in the database, they propose unsupervised anomaly detection. In this case, positional and distance functions are defined for the quiplets and clustering techniques (*k*-centers and *k*-means) map every user to its representative cluster, which is the cluster with the highest number of training records for that user after the clustering phase (training phase).

For each new query to test, two approaches can be used:

1) Given the determination of its representative cluster, use the NBC as in the Role-Based anomaly detection to perform a similar test; or

2) Verify if the new query is a statistical outlier using the MAD (Median of Absolute Deviations) test [Pham-Gia and Hung, 2001], which if true considers the action as an intrusion and generates an alert.

Sequence Alignment Analysis. Sequence alignment mainly consists in determining common sequences of events (such as commands, data attributes, accessed values, etc). DIDS using this type of techniques typically learn and identify the repeatable series of events with significant length and eventually break them into smaller-sized subsets to label or classify those sequences and their subsets as normal user behavior. In the detection phase, each sequence of new events is matched against the learnt user sequences and their subsets for measuring how they differ in order to evaluate its probability of being an intrusion.

The solution presented in [Kundu *et al.*, 2010] identifies sequences of accessed attributes, commands and tables for building user profiles. The proposed features are the command types (insert, select, etc.), designed sensitive attributes, all attributes, operations on attributes, and mixes of all features. This work also defines criteria for choosing among user-based, role-based or organization-based profiles, given the working context of the database.

In the learning phase, it builds sequence models given a threshold for determining the maximum number of differences. In the detection phase, it also uses a threshold for computing the highest number of differences allowed between the tested sequences and those retained in the learning phase, to consider the sequences as normal or abnormal.

Integrating Dependency with Sequence Alignment Analysis. An approach for finding dependency relationships among transaction-level attributes with high support and confidence rules is proposed in [Hu and Panda, 2004]. These authors observed that in real-world applications, although the transaction application can often change, the whole database structure and essential data correlations rarely change. They assume that whenever an attribute is updated, this action is linked to a sequence of other events logged in the database (*e.g.* due to an update of a given attribute, other attributes are also read or written). Thus, each update is defined by three sets: the read set, a set of attributes that have been read because of the update; the pre-write set, a set of attributes that have been written before the update and because of it; and the post-write set, a set of

attributes that have been written after the update as a consequence of it. Transactions that do not follow any of the mined data dependency rules are marked as malicious.

The work in [Srivastava *et al.*, 2006a; Srivastava *et al.*, 2006b] improves that of [Hu and Panda, 2004] by considering attribute sensitivity, *i.e.*, giving a measure of importance to each attribute. They propose three levels of attribute sensitivity, considering its support in the analyzed transactions: high, medium and low. A weighted data mining algorithm is used to mine the dependencies between database attributes and generate rules that reflect that dependency, given the measured sequences of operations (read, write) and the sensitivity of each attribute. Any transaction that does not follow these rules is identified as malicious. The authors also present an extension to the Entity-Relationship model to syntactically capture the sensitivity level of the attributes.

In [Fonseca *et al.*, 2008], a generic learning algorithm for representing transactions by directed graphs describing execution paths is proposed. New profiles that deviate from the ones learnt from those execution paths are seen as unauthorized sequences of SQL commands. The features used to build the execution paths are the command type (select, insert, delete, etc.), target objects (tables) and selected columns, and restriction attributes, all of which are obtained from typical DBMS audit entries [Newman, 2011] storing information on the UserID, SessionID, CommandID, TransactionID, user command, object owner, and a timestamp of its execution.

Statistical Analysis. Statistical analysis is used in several DIDS for computing user activity and/or data statistics ID features.

The approach presented in [Spalka and Lehnhardt, 2005] makes use of statistical functions on reference values obtained from the data in relations (alias tables) and Δ-relations (changes of the values of the monitored objects/attributes for all reference values, per attribute, between two runs of the DIDS) for anomaly detection.

An extension is defined as the set of all rows of an insertion/modification of data and a relation refers to a table or view. The reference values include count, minimum, maximum, average, standard deviation, ranges, computed ratios, zero length checking and bit counting. A misuse

detection method is also included, which works by examining database objects (Database, Default, Function, Index, Privilege, Procedure, Rule, Schema, Statistics, Table, Trigger, and View) and all operations on them. This is done by previously defining if each pair <Database object, operation> is dangerous or not.

The work proposed in [Mathew *et al.*, 2010] is based on computing summarized statistics such as counting, maximum, minimum, mean, median, standard deviation and cardinality values of each attribute from the dataset resulting or affected by the execution of each user command. These statistics are stored in a vector with fixed dimension named as an S-Vector, regardless of how large the command's result dataset may be. When the dataset for obtaining the S-Vector is large, the authors propose sampling the dataset by fetching the first initial k tuples or a subset of randomly picked k tuples, for maintaining performance and scalability. The set of each user's S-Vectors is then used for applying techniques such as clustering, naïve Bayes, support vector machines or decision trees in order to obtain models that represent the user's normal behavior given the information in those S-Vectors. In the intrusion detection phase, statistical deviation and outlier verification is applied to inspect each user command and classify it as normal or abnormal.

Information-Theoretic Analysis. Approaches using information-theoretic analysis compute measures like entropy and information gain for characterizing user profiles and compare them with those of subsequent user actions to see how they differ from the original ones.

The work in [Lee and Xiang, 2001] describes such a solution. Features are composed by a tuple of audit data with n variables for each data object (*e.g.* IP address, message size, etc). Entropy is used as a measure of regularity of audit data (*e.g.* event types such as a list of commands), where each record represents a class; the smaller the entropy, the fewer the number of distinct records (*i.e.*, the higher the redundancies), the more regular the audit dataset. The fact that many events are repeated (or redundant) in a dataset suggests that they are likely to appear in the future. Anomaly detection models constructed using datasets with small entropy will likely be simpler and have better detection performance.

Conditional entropy is used to define temporal sequences of audit data. $H(X|Y)$ shows how much uncertainty remains for the rest of the audit

events in a sequence X after seeing Y. For anomaly detection, it is used as a measure of regularity of sequential dependencies. If the audit trail is a sequence of events of the same type, then the conditional entropy is 0 and the event sequences are deterministic. Conversely, large conditional entropy indicates that the sequences are not as deterministic and hence much harder to model.

Relative conditional entropy between distributions is used for measuring regularities (distance) between two audit datasets, where the training dataset is a validated audit dataset and the tested dataset is the one that needs to be inspected. Once again, the best solution is the one with smaller relative conditional entropy. Information gain is introduced to aid the feature selection and construction process to improve the detection performance because of its direct connection with conditional entropy. The higher information gain owned by the feature, the smaller conditional entropy, and hence the better detection performance.

Command Template Analysis. Command modeling DIDS use a command database log to analyze all the regular user commands and build some sort of summarized templates that are able to generically represent the typical user workloads.

In [Lee *et al.*, 2002], an algorithm summarizes a set of supposed "legitimate" queries into SQL templates that represent the models of all those queries. Each conditional filtering variables in the WHERE clause of similar commands are considered as parameters. To see if an unbounded variable should be used for each parameter or a finite list of values, a Kolmogorov-Smirnov test is done at a 90% confidence level. The algorithm also tabulates the frequency of each learnt fingerprint, *i.e.*, how often it occurs in the set of SQL statements.

Taking a new fingerprint `F` and a previously defined fingerprint `F'`, `F` is considered legitimate if `F` differs from `F'` only by: 1) any extra conditions in the WHERE clause of `F` that are missing from `F'` are joined with the AND operator; and 2) `F` selects an equal or fewer number of columns than `F'`. They also propose a method for deducing missing fingerprints (*i.e.*, ranges of queries that are similar to the database log queries used in the learning phase), based on mixing the possible combination of conditions in the WHERE clause from the previously acquired fingerprints. In the testing

phase, each command significantly differing from the computed fingerprints is considered abnormal.

In [Bockermann *et al.*, 2009] the authors propose applying a grammar-based analysis using machine-learning techniques instead of commonly used vector-based data. This approach applies tree-kernel based learning, which has become popular in natural language processing, using the parse-tree structure of SQL for correlating commands with applications and to differentiate between benign and malicious ones by inspecting changes in command syntax trees.

They derive a distance measure induced by a tree-kernel function to measure the similarity of SQL commands using their parse-trees. Support vector machines are used in the learning phase and clustering is applied for distinguishing benign from malicious commands by outlier detection. This method promotes a context sensitive similarity that enables locating the nearest non-intrusive command for a malicious statement, which helps in root cause analysis.

Table 2-2 summarizes the approaches previously described, mentioning each type of technique along with the actions and user action elements that can be analyzed. It also shows if each approach allows implementing intrusion prevention, *i.e.*, if it enables stopping the intrusion action *a priori* to its execution.

In what concerns intrusion prevention, which is the capability of stopping the intrusion action when it occurs or even before it occurs, it can be seen that several solutions enable full intrusion prevention, while others can only partially accomplish this. In [Lee *et al.*, 2000], the temporal analysis technique detects any queries that request execution outside a predefined time schedule and may therefore deny their execution and prevent the intrusion action. The sequence analysis technique used in [Kundu *et al.*, 2010] may enable intrusion prevention by avoiding subsequent user actions when it detects a suspicious sequence of actions. However, it needs to wait for a significant amount of actions that make up that sequence, meaning that it will probably only detect the intrusion after some of those actions have finished their execution, which makes it only capable of partial intrusion prevention.

Table 2-2. Database intrusion detection techniques and their coverage

Technique	Reference	Elements that can be analyzed				Intrusion Prevention Capability
		Command Syntax	Accessed Columns	Processed Rows	Result Dataset	
Temporal Analysis	[Lee *et al.*, 2000]	X				Yes
Dependency and Relation Analysis	[Chung *et al.*, 1999]	X	X			Yes
	[Zhong and Qin, 2004]	X	X	X		Yes
	[Bertino et al., 2005b]	X	X			Yes
	[Kamra *et al.*, 2008]	X	X			Yes
Sequence Analysis	[Kundu *et al.*, 2010]	X				Partial
Integrated Dependency and Sequence Analysis	[Hu and Panda, 2004]	X	X			Partial
	[Srivastava *et al.*, 2006]	X	X			Partial
	[Fonseca *et al.*, 2008]	X	X			Partial
Statistical Analysis	[Spalka and Lehnhardt, 2005]	X	X	X		Partial
	[Mathew *et al.*, 2010]	X	X		X	No
Information-Theory Analysis	[Lee and Xiang, 2001]	X				Partial
Command Template Analysis	[Lee, 2002]	X	X			Yes
	[Bockermann *et al.*, 2009]	X	X			Yes

All the solutions based on dependency and relation analysis that were described [Bertino *et al.*, 2005; Kamra *et al.*, 2008; Zhong and Qin, 2004] are fully capable of enabling intrusion prevention, since they may check each individual user command syntax and if they find those commands suspicious their execution can be stopped before their execution occurs. The solutions integrating a mix of dependency and sequence analysis such as [Fonseca *et al.*, 2008; Hu and Panda, 2004; Srivastava *et al.*, 2006a; Srivastava *et al.*, 2006b] are capable of performing only partial intrusion prevention, for the same reasons pointed out in the previous paragraph concerning the solution proposed in [Kundu *et al.*, 2010].

The solutions presented in [Mathew *et al.*, 2010; Spalka and Lehnhardt, 2005], which are based on statistical analysis, are mostly incapable of intrusion prevention, as they mostly rely on analyzing the changes in data or execution results *after* they have been processed. This means they can only detect the intrusion *a posteriori* to the attack. However, the approach in [Spalka and Lehnhardt, 2005] can be adapted to check *a priori* statistical data concerning the rows requested to be processed by the user action, enabling it to have partial intrusion prevention capabilities. For this same

reason, the information-theory analysis approach presented in [Lee and Xiang, 2001] may also accomplish partial intrusion prevention.

The solutions based on command template analysis proposed in [Bockermann *et al.*, 2009; Lee *et al.*, 2002] can fully enable intrusion prevention due to same reason as those previously mentioned that use dependency and relational analysis [Bertino *et al.*, 2005; Chung *et al.*, 1999; Kamra *et al.*, 2008; Kamra, 2010; Zhong and Qin, 2004].

Besides the previously described specific ID techniques and approaches that can be used in databases, other research works have been published that can also contribute to this intrusion detection field. For example, although it does not present itself as a DIDS, the work in [Motwani *et al.*, 2008] describes a method for auditing SQL queries to measure their suspiciousness from a privacy and confidentiality perspective that may be useful for intrusion detection purposes. A generic survey on how data mining techniques can be applied to intrusion detection is shown in [Pei *et al.*, 2004].

2.4.3. Using Database Intrusion Detection Systems in Data Warehousing Environments

By observing Table 2-2 it can be seen that most DIDS focus on analyzing user command syntax (*i.e.*, parsing the SQL-expression syntax of queries to construct user profiles). As pointed out in [Mathew *et al.*, 2010], the most common problems with this type of approach is:

- Regular user queries may differ widely in syntax yet produce "normal" (*i.e.*, good non-intrusive) output, which generates false positives (*i.e.*, false alarms);
- Queries may be crafted by the attacker to differ slightly in syntax from the "normal" user behavior profiles yet produce "abnormal" (*i.e.*, malicious and intrusive) output, which generates false negatives (*i.e.*, attacks that pass undetected).

Given the expressiveness of the SQL language and the need to determine query equivalence or similarity, it is evident that syntax analysis is complex and very difficult to perform correctly. In fact, query containment and equivalence is NP-complete for conjunctive queries and uncertain for queries involving negation [Mathew *et al.*, 2010].

In databases where typical user workloads have a well-defined number of distinct commands that are issued repetitively, relying on command syntax analysis may be feasible to achieve high ID efficiency. This is typically what occurs in transactional systems. However, in analytical systems such as DW's many actions are *ad hoc* and have variable execution times with variable data access patterns and dimension-size frequencies and thus, are mostly unpredictable and broad-scoped. This makes distinguishing between normal and abnormal commands in DWs an extremely difficult task. In such analytical databases, limiting ID to command syntax analysis by simply modeling SQL command templates or static frequent data access patterns (*e.g.* which tables or columns are accessed) is unreliable or, at least, minimalist.

Regarding the previously presented characteristics of DW user workloads, the ID solutions relying on temporal analysis such as presented in [Lee *et al.*, 2000] are inadequate and mostly produce very poor ID results due to the unpredictable rate and execution time of those workloads. Due to the *ad hoc* nature of most of those workloads, ID solutions such as [Bockermann *et al.*, 2009; Lee *et al.*, 2002] that are based on command template analysis lack the necessary dynamics to efficiently perform the ID processes and therefore also produce poor ID results.

Although the approach proposed in [Mathew *et al.*, 2010] adds a data-centric analysis of each user command execution's resulting dataset, the analysis is performed *a posteriori* to that execution. Given the time span between the start of the intrusion and its detection, together with resource consumption and sensitivity of the targeted data, many enterprises can suffer huge losses if their DIDS either takes too long to alert a malicious intrusion or is unable to prevent or stop its execution. In this sense, these approaches alone are not efficient solutions for intrusion detection in DWs.

Conclusively, the unpredictable execution frequency and *ad hoc* nature of the user workloads make time-based and SQL templating ID approaches such as [Bockermann et al., 2009; Lee *et al.*, 2002; Lee *et al.*, 2000] mostly inadequate. On the other hand, DIDS performing ID at a coarse-grained basis such as database sessions or transaction command sets, instead of a fine-grained basis such as analyzing each SQL command, risk that a series of malicious commands may be executed before the intrusion can be dealt with. Therefore, data dependency and sequence alignment approaches

such as [Chung *et al.*, 1999] that are able to inspect each user command *a priori* to its execution, but only after a considerable amount of actions have been executed, should be used carefully according to each DW context.

Data-centric techniques such as [Mathew *et al.*, 2010; Spalka and Lehnhardt, 2005] are capable of bringing added value to *a priori* ID techniques by executing an *a posteriori* analysis of the data affected by the user action. Combining these techniques with data access pattern analysis techniques such as [Bertino *et al.*, 2005; Kamra *et al.*, 2008], that deem the processed data, seem *a priori* the most feasible and efficient DIDS for DWs.

2.5. Summary

This chapter presents the background and related work concerning the data security domains focused by the research work in this thesis, namely data masking, encryption and database intrusion detection.

The concepts concerning DWs are described and data warehousing environments are characterized. The differences and characteristics that distinguish operational systems from DWs have also been detailed.

We have also enumerated and described the standard and state-of-the-art techniques and methods in data masking, encryption and database intrusion detection systems, and discussed the issues concerning their applicability in data warehousing environments.

Chapter 3

Data Warehouse Security Framework

Despite the fact that published research and best practice guides from many DBMS vendors state that the best way to protect data in databases is to use encryption solutions together with intrusion detection systems, to the best of our knowledge there has been no proposal regarding a conceptual framework for integrating these distinct solutions together. In this chapter, we propose a framework that enables integrating together the proposed masking, encryption and intrusion detection solutions, which are presented in the following chapters.

The proposed framework can be seen as a middle tier between the user interfaces and the DBMS, working as an extension of the DBMS itself. We define the sequence of steps within the scope of the framework, that occur from the moment a user statement arrives at the data warehouse to be processed, and describe the information flow and each of its components. We also define a series of principles that drive the development of the masking, encryption and DIDS solution proposed in this thesis. These guidelines deal with the issues of data security and provide a body of knowledge for the development of specific solutions for data warehousing environments.

The chapter is organized as follows. Section 3.1 details the middle tier and how it enables integrating data masking, encryption and intrusion detection to deal with user actions in a single pass-through overall process. Section 3.2 presents the guidelines for enhancing data masking and encryption in data warehouses and Section 3.3 presents the guidelines for enhancing intrusion detection in data warehouses. Finally, Section 3.4 concludes the chapter.

3.1. Overview of the Data Warehouse Security Middle Tier

The typical information flow of data warehouse user actions between the interface used by the user and the DW database(s) is shown in Figure 3-1. In practice, the user interface typically issues a SQL statement and sends it to the DBMS, which then processes it against the respective database(s), receive the processed results, and finally send it back to the user interface that requested its execution.

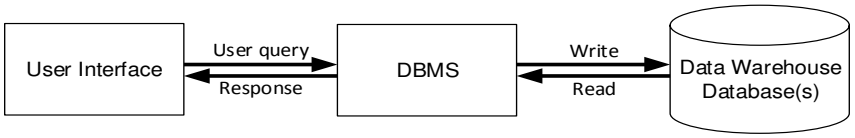


Figure 3-1. Typical DW user action information flow

In the context of our work, each SQL statement is parsed and analyzed once it arrives at the DBMS. Whenever required, data masking, encryption and intrusion detection are applied given the command itself and its targeted data, immediately before the command is executed. Intrusion detection is also applied to the processed data and results after its execution finishes and before disclosing the results back to the users. The sequence of steps given a request to process a SQL statement issued by the user is shown in Figure 3-2.

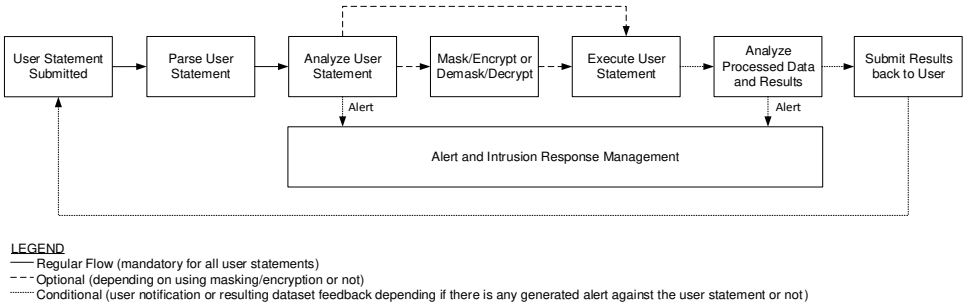


Figure 3-2. Step sequence of the submittance of a SQL user statement

As shown in the figure, each user statement is parsed and then analyzed before it is executed by the *DBMS*, to make an *a priori* verification of its suspiciousness. If it is considered an intrusion, then an alert should be raised against this user action and its execution can be stopped at this step.

Contrarily, if it is not considered an intrusion, then the user statement can be processed by the *DBMS* against the *Data Warehouse Database(s)* with or without use of the data masking or encryption processes, according to the security measures defined for the targeted data. After the user statement finishes being processed by the *DBMS*, the processed data and resulting dataset are also be analyzed for suspiciousness. If it is considered an intrusion, then an alert is also raised against the user action and disclosure of the results can be stopped at this step, otherwise the results are sent back to the user.

To accomplish the aimed functionality according to this sequence of steps, the framework includes intrusion detection, masking and encryption components, defining an information flow as shown in Figure 3-3.

The middle tier includes mandatory and optional components, considering that the intrusion detection processes are mandatory and the masking and encryption processes are optional, given the functionalities defined by the security administrators. For example, parts of the database may require encryption or masking due to security requirements, while other parts of the database may not require encryption or masking. This means that a user command is always subjected to the intrusion detection components, but might not require going through the masking or encryption components.

The main elements of the information flow of the middle tier and each of its components are described in the following subsections.

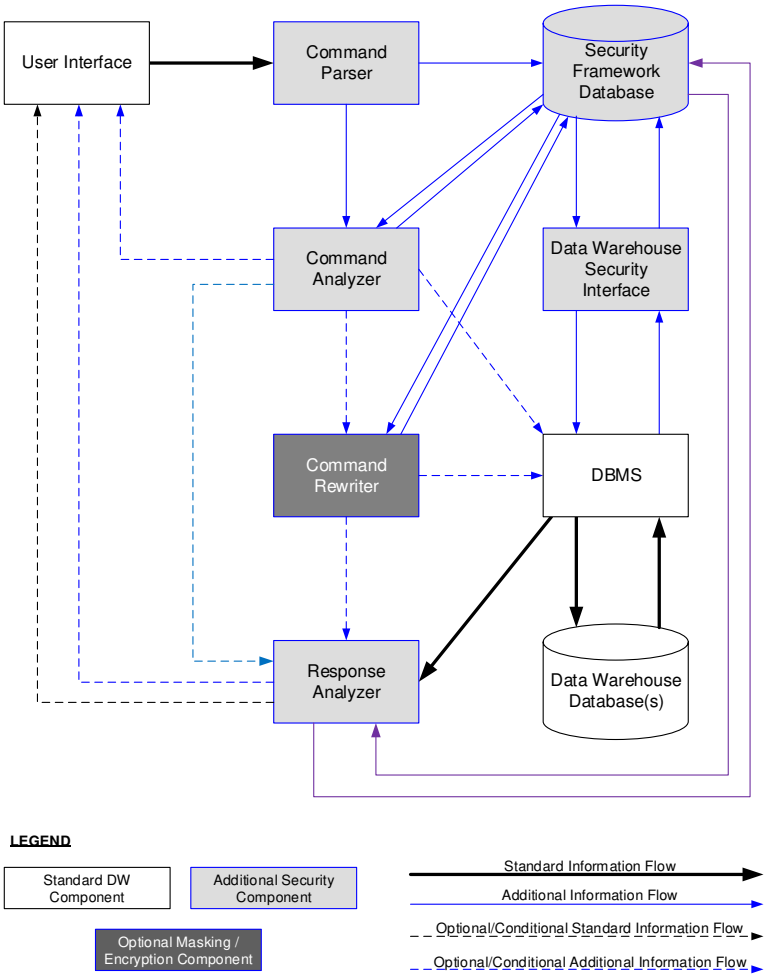


Figure 3-3. Integrated Data Warehouse Security Framework

3.1.1. The Security Framework Database

The *Security Framework Database* is a database that stores all the user data that enables identifying each DW user (name and password) and his/her data access policies (attributed role(s) and SQL grant privileges) and a historical command log that stores all the issued user commands against the data warehouse database(s), together with the information required for each component of the masking, encryption and intrusion detection processes.

For masking and encryption, the *Security Framework Database* stores all the necessary masking and encryption keys for each DW database that needs to be masked or encrypted. On the other hand, for intrusion detection purposes, the *Security Framework Database* stores all the DW user behavior profiles that will be used to assess the incoming user statements. It also contains the complete history of all the generated alerts in an alert log that identifies the user command to which each alert refers and attributes for enabling the *Data Warehouse Security Administrator* to confirm if that alert concerns a true intrusion action or a false alarm. The rulebase for the risk exposure method and the risk exposure measure computed for each alert is also stored in the database.

3.1.2. The Data Warehouse Security Interface

The *Data Warehouse Security Interface* is used by the *Data Warehouse Security Administrator* for managing the *Security Framework Database* and all the masking, encryption and intrusion detection components. Whenever the *Data Warehouse Security Administrator* wants to protect a data warehouse database by applying the framework, the following actions should be performed:

- After entering the DBA login and database connection data, the *Data Warehouse Security Interface* scans all the data access policies defined in the *Data Warehouse Database(s)* for identifying authorized users and respective permissions;
- A user command log is created in the *Security Framework Database* for recording all future user actions requested to execute against the *Data Warehouse Database(s)*;
- All user behavior profiles are then built using the *Data Warehouse Database(s)* command log and the existing data.

The interface allows the *Data Warehouse Security Administrator* to define the rules to be used by the intrusion detection risk exposure method. It also displays the information concerning all the generated intrusion alerts and allows the *Data Warehouse Security Administrator* to confirm the authenticity of each alert, *i.e.*, if it refers to a true intrusion or a false alarm.

The *Data Warehouse Security Administrator* may use the *Data Warehouse Security Interface* to define, at any time, which attributes should be masked

or encrypted. Each time this type of action is required, all the data concerning such attributes is immediately masked or encrypted by replacing the original values with the new masked or encrypted ones. Whenever the *Data Warehouse Database(s)* needs to be updated, this must always be done through the middle tier instead of directly through the DBMS.

3.1.3. Analyzing the User Statement a Priori

Before the user statement can be processed by the DBMS, it must be analyzed to verify its suspiciousness and assess if it is an intrusion or not. The information flow referring to this initial process is shown in Figure 3-4.

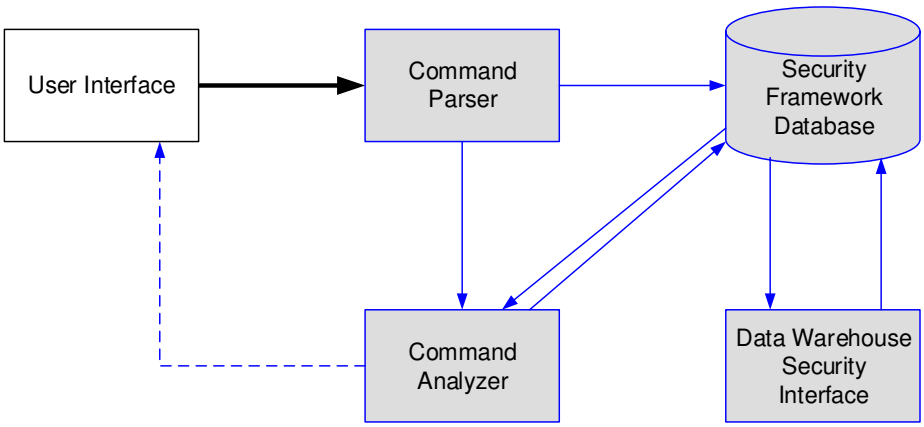


Figure 3-4. Information flow concerning the *a priori* analysis of the user statement

First, the user statement must go through the *Command Parser* component. The *Command Parser* component is responsible for parsing the SQL statement, splitting it into its individual sub-queries (if it has any sub-query) and extracting the relevant intrusion detection features (defined by the DIDS – the DIDS proposed in this thesis is explained in Chapter 6, including its respective features), which are finally passed to the *Command Analyzer* component. The command itself and the information that traces it back to the user that requested its execution, as well as the moment when that execution was requested, are stored in the *Security Framework Database*.

Afterwards, the query (and sub-queries' set) is passed on to the *Command Analyzer* component. An important aspect is that the DBMS should be configured to only process SQL statements that have gone through the *Command Analyzer* component. All SQL statements that avoid going through the *Command Analyzer* should be rejected by the DBMS. The *Command Analyzer* retrieves the information regarding the user behavior profile to which each command concerns from the *Security Framework Database*, and performs the respective intrusion detection tests on each command to verify if it should be considered an intrusion. If the user command is considered an intrusion, the *Security Framework Database* is updated by flagging the command as a potential intrusion and an alert is generated, which is passed on to the *Data Warehouse Security Interface* in order to be communicated to the *Data Warehouse Security Administrator*, and the user action may be stopped. If the user action is not considered an intrusion, it can then be executed by the DBMS against the *Data Warehouse Database(s)*, which is the next step.

3.1.4. Executing the User Statement

A user statement that has not been considered an intrusion by the *Command Analyzer* component may be executed by the DBMS. There are two possibilities:

1) If the user statement does not contain any reference to masked or encrypted columns, then it is immediately executed by the DBMS;
2) If the user statement contains any reference to masked or encrypted columns, then it is passed on to the *Command Rewriter* component to be modified in order to correctly execute against the masked and/or encrypted data, and then it is executed by the DBMS.

The information flow referring to this process of executing the user statement is shown in Figure 3-5. In practice, for each user statement deemed as a non-intrusion, the *Command Analyzer* component notifies the *Response Analyzer* component to wait for a response so the targeted processed data and the statement's execution results can also be analyzed.

As we explain further in chapters 4 and 5, the proposed data masking and encryption algorithms only use operators and transformations that are native to standard SQL. This allows them to simply rely on SQL rewriting to accomplish their masking/unmasking and encryption/decryption

purposes. After receiving a user statement from the *Command Analyzer*, the *Command Rewriter* queries the *Security Framework Database(s)* to retrieve the necessary data masking and encryption keys for that user statement and applies the required SQL rewriting to the user statement and sends it to be executed by the DBMS. When a user statement completes its execution, the results are sent to the *Response Analyzer* component to perform an *a posteriori* verification.

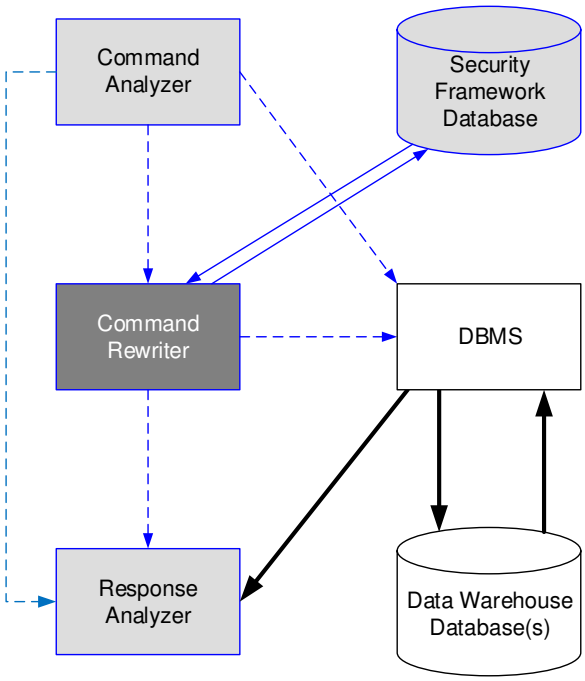


Figure 3-5. Information flow concerning the execution of the user statement

3.1.5 Analyzing the Processed Data and Dataset Result a Posteriori

After the user statement has been processed by the *DBMS* against the *Data Warehouse Database(s)*, the results are sent to the *Response Analyzer* to check if the processed data and the results themselves are suspicious, given the behavior profile of the typically accessed data and resulting datasets of the user to which the statement belongs. The information flow referring to this process is shown in Figure 3-6.

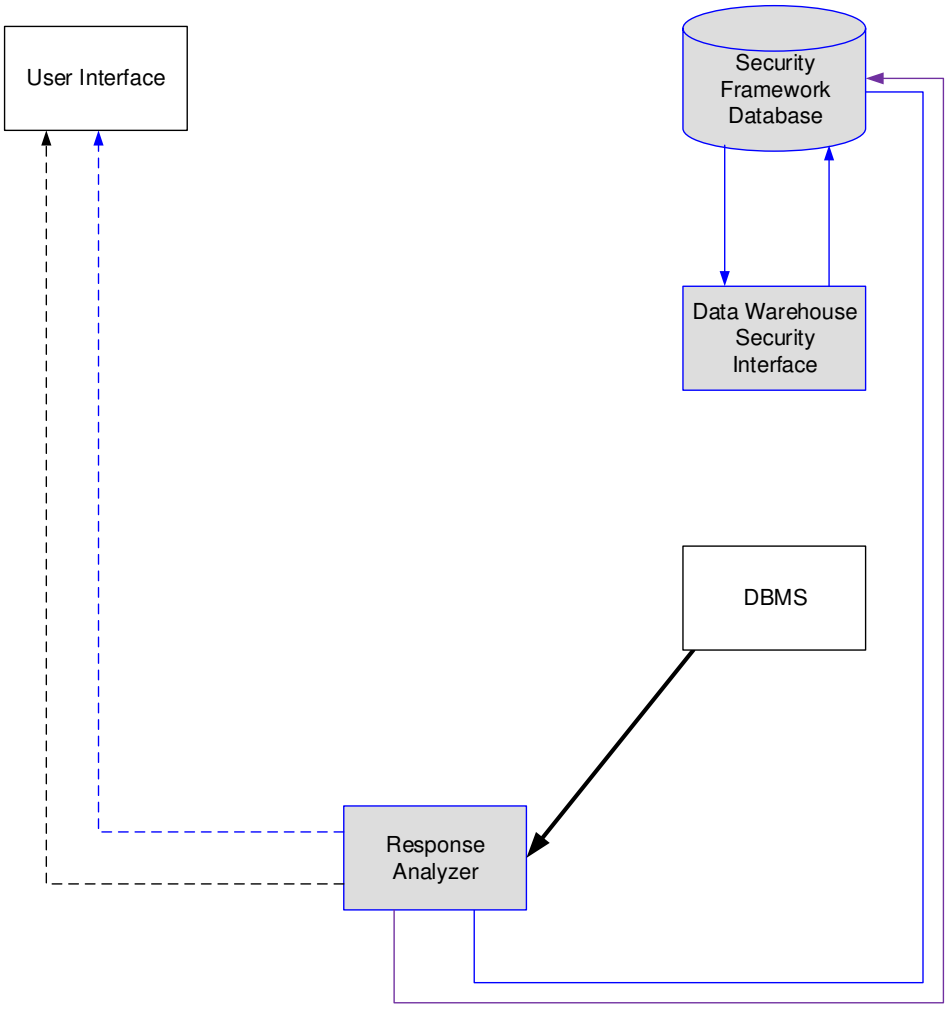


Figure 3-6. Information flow of the *a posteriori* analysis of the user statement

The *Response Analyzer* retrieves the information from the *Security Framework Database* regarding the features belonging to the behavior profile of the typically accessed data and resulting datasets of the user to which the statement belongs, and performs the respective intrusion detection tests against the values of the processed data and resulting dataset to verify if it should be considered an intrusion. If it is considered an intrusion, then the *Security Framework Database* is updated by flagging the command as a potential intrusion and an alert is generated, which is then passed on to the *Data Warehouse Security Interface* in order to

communicate the event to the Data Warehouse Security Administrator, and the user action can be stopped. If the user action is not considered an intrusion, the results are simply sent back to the user that requested the execution and the *Security Framework Database* is updated by flagging the action as a non-intrusion that has completed its execution.

3.2. Guidelines for Enhancing Data Masking and Encryption Performance in Data Warehousing

In this section, we present the guidelines that drived the development of the data masking and encryption solutions proposed in Chapters 4 and 5. These generic principles intend to deal with the data masking and encryption issues pointed out in Chapter 2, and establish the foundations for each proposed solution in the context of the middle tier presented in the previous section.

3.2.1. Numerical vs Textual Masked or Ciphered Input and Output

As mentioned in Chapter 2, standard encryption algorithms were conceived for encrypting general-purpose data and therefore, receive and output textual or binary data, while data warehouse data is mostly composed by numerical datatype fact table columns that typically take up 90% or more of the total storage space [Kimball and Ross, 2013]. Most data warehouse user workloads request processing arithmetic functions such as sums, averages, etc., which implies that those textual or binary values need to be converted back into their numerical format.

Since working with text values is much more computationally expensive than working with numerical values, standard ciphers are much slower than ciphers specifically designed for receiving numerical inputs and producing numerical outputs.

Therefore, to avoid the overhead processing time concerning the referred datatype conversions, the masking and encryption solutions proposed in this thesis were specifically designed to receive numerical input and produce numerical output.

3.2.2. Preserving Column Datatypes

Considering that numerical datatype sizes usually range from 1 to 8 bytes, while standard encryption outputs have lengths of 8 to 32 bytes [Natan,

2005] and that data warehouses have a huge amount of rows that typically take up many gigabytes or terabytes of space, even a small increase of any column size required by changing numeric datatypes to textual or binary (in order to store encryption outputs) introduces very large storage space overhead. This consequently increases the amount of data to process, as well as the required storage and processing resources, which also degrades database performance.

While the importance of encrypting text values might be significant or not for data warehouses (depending on its context), efficiently encrypting numerical values is critical, as these represent the business facts. The masking and encryption solutions proposed in this thesis allow preserving the original datatype and length of each encrypted column, which allows maintaining their original data storage space.

3.2.3. Using Only Native SQL Operations to Mask/Encrypt Data

Another issue previously pointed out concerns the data roundtrips between the database and the encryption and decryption mechanisms. Topologies involving middleware solutions such as the one proposed in [Radha and Kumar, 2005] typically request all the encrypted data from the database and execute decrypting actions themselves locally, finally sending the decrypted results back to the user that requested them. Given the typically large amount of data accessed for processing DW queries, previously acquiring all the data from the database for encrypting or decrypting in a middle tier is unfeasible. This strangles the database server and/or network with communication costs due to bandwidth consumption and I/O bottlenecks, jeopardizing throughput and consequently, response time.

As our approach is based on operators supported by native SQL, it requires only query rewriting for masking/encrypting and unmasking/decrypting actions. In fact, using only native SQL operators and functions brings several major benefits:

- It allows building the sequence of steps for all masking/encrypting and unmasking/decryption processes as a unique SQL statement, and no external languages or resources need to be instantiated;

- Computing the masking/encrypting and unmasking/decrypting operations as a SQL statement enables them to run directly against the data, avoiding data roundtrips between the database and the masking and encrypting mechanisms and thus, avoiding I/O and network overhead from the critical path;
- Contrarily to what happens with standard encryption algorithm implementations, which are typically OS platform and CPU dependent, using only native SQL makes our solutions DBMS platform independent, making them usable in any data warehouse running on any CPU model, without depending on any programming language or external OS resource;
- Since the SQL statements can run directly against the masked or encrypted data, it means that the data can remain masked or encrypted at all times, only disclosing the computed results back to the user which requested the statement's execution.

3.2.4. Masking and Encryption Algorithm Design

As discussed in Chapter 2, the complexity of each transformation round in masking and encryption algorithms is directly linked with the security strength achieved by the algorithm, as is the number of rounds it executes and the size of the used encryption key(s). It is assumed by the security community as a general rule that, as the number of complex operations, encryption key lengths, and/or number of encryption rounds increase, the algorithms security strength also increases or, at least, remains the same [Vaudenay, 2006]. However, increasing the complexity of the "data mix", the number of rounds or the encryption key length also introduces a performance drawback, since it requires more machine resources and processing time.

In what concerns the design of "data mixing" for each masking or encryption round, we discard bit shifting and permutations, commonly used by most ciphers [Vaudenay, 2006], since there is no standard SQL support for these actions. We also discard the use of substitution boxes (*e.g.* AES uses several 1024-byte S-boxes, each of which converts 8-bit inputs to 32-bit outputs), because of their complexity and resource consumption.

Our masking and encryption approaches are based on the widely used and well known XOR and MOD operators, which are available to be implemented in native SQL. In practice, we propose the use of a set of arithmetic operators combined with XOR and MOD operators to transform numerical data.

The XOR operator is widely used in most encryption algorithms. In fact, it is the baseline for achieving perfect secrecy in the most basic encryption transformation, the Vernam Cipher[7] [Vaudenay, 2006]. Its properties in achieving perfect secrecy given certain conditions and its ease in mixing up the input values makes the XOR operator an excellent candidate for building data transformation functions for masking or encryption purposes.

The modulus (MOD) remainder operator is another good candidate for data transformation functions with masking or encryption purposes, because it enables building non-invertible functions. For a function to be directly 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 [Bartle, 1976]. An injective function is a function that preserves distinctness: it never maps distinct elements of its domain to the same element of its codomain. From an information theory perspective, this means that for an injective function, each input-output pair has intrinsically the exact same probability of occurrence. This provides information to break the cipher's key if the attacker has access to

[7] The Vernam Cipher was published in 1926 by Gilbert Vernam from AT&T. It is based on an encryption key with the same bit length as the input plaintext and applies a XOR operation against both values to get the encrypted output. Shannon proved that this cipher achieved perfect secrecy if the keys are generated in a randomly uniform distribution and the same key is only used once to encrypt one input value. In this case, there is no information leakage because the same key is never used twice and the attacker needs to test all possible encryption key values in each case to guarantee absolute success in the attack, requiring on average half of that number in order to succeed. Statistically, perfect secrecy means that the *a posteriori* distribution of the plaintext X after the encrypted ciphertext Y is known is equal to the a priori distribution of the plaintext: the conditional distribution of X given Y is equal to the original distribution. Formally, for all x and y such that $\Pr[Y = y] \neq 0$, we have $\Pr[X = x \mid Y = y] = \Pr[X = x]$.

its algorithm and set of outputs. Therefore, the main objective of a cipher should be to assure a maximum of non-injective transformations in order to introduce uncertainty over which inputs generate the output, thus avoiding information disclosure to break the cipher.

The MOD 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 that the transformation functions that use MOD are also non-injective. Given that injectivity is a required property for having invertibility, masking or encryption algorithms that use MOD transformations are therefore, non-invertible.

3.3. Guidelines for Enhancing Intrusion Detection in Data Warehousing

This section presents the guidelines that drove the development of the intrusion detection solution proposed in Chapter 6. These principles intend to deal with the data warehouse intrusion detection issues pointed out in Chapter 2 in the context of the middle tier presented in Subsection 3.1.

3.3.1. Using Individual User Profiles

In typical transactional systems, it is normal to have a very high number of predefined queries that are issued in a repetitive manner by each user, making most queries extremely predictable. For example, each teller in a supermarket store is always repeating queries to retrieve individual product prices. Furthermore, independently from the number of tellers, all of them mostly repeat the same type of query. Considering a generalization of this typical operational business environment, it is easy to understand that user profiling in transactional systems is relatively simple and user role profiles can be built, instead of building an individual profile per each user.

Decision support systems do not have the same user characteristics as those of operational transactional systems. As previously mentioned in Chapter 2, distinguishing normal from abnormal user behavior in data warehouses is a very difficult task, given the typical high amount of *ad hoc* queries issued by the users. On the other hand, given that each user has its

own data query demands that are closely linked to his/her business role, the portion of *ad hoc* queries inherent to each user should typically contribute to reveal a unique profile that distinguishes each user from the remaining. Therefore, in this work we claim that user profiles in DWs should be built with the highest detail, *i.e.*, individual profiles should be built for each user in order to obtain high intrusion detection rates, against role-based profiling as suggested in other approaches such as [Kamra *et al.*, 2008].

3.3.2. Analyzing the Targeted Tables and Columns, Processed Data and Resulting Datasets

None of the intrusion detection techniques proposed in the past is capable of analyzing all the aspects directly linked with user behavior in what concerns database usage in an integrated manner. For instance, the RBAC intrusion detection approach proposed by [Kamra *et al.*, 2008] profiles the columns and tables accessed by the users that belong to a given role. In our opinion, reducing the analysis of user behavior merely to this type of approach is too simplistic.

Most intrusion detection techniques focus on features that enable the analysis of which tables and columns are being targeted by the user actions. Few techniques focus on the data processed by the user actions or on the resulting datasets themselves, which are a consequence of processing those user actions. We argue that such distinct approaches should be integrated so the features can reflect the impact produced by the user actions for all the previously referred aspects or dimensions.

Therefore, the DIDS proposed in this thesis uses features that enable analyzing the targeted tables and columns included in the user actions, the data processed by those actions and its resulting datasets, in an integrated manner, which never occurs in current DIDS.

3.3.3. Intrusion Detection and Prevention a Priori and a Posteriori

In the past, each DIDS approach for analyzing user actions from a timely perspective could be divided into two main groups: 1) analyzing the user action *a priori* to its execution; or 2) analyzing the user action *a posteriori*, *i.e.*, after it finished its execution. Of course, the second type of analysis would not be able to provide intrusion prevention, which we consider

critical for data warehouses. In this work we consider that both types of analysis should be used, before and after the user actions are executed and before its results are disclosed.

The DIDS approach proposed in this thesis focuses not only on building user profiles regarding features holding information on the issued SQL commands, but also includes features that infer information on the processed data and resulting datasets. This enables our solution to perform intrusion detection and prevention both *a priori* and *a posteriori* to the execution of user actions, before the results are disclosed back.

3.3.4. Using Risk Exposure for Alert Management

When analyzing user actions, most DIDS output numerical measures that require defining thresholds to determine if those values imply considering the respective user actions as intrusions or non-intrusions. While defining high thresholds could potentially produce less false alarms and give higher assurance that a generated alert would in fact refer to a true intrusion, this could also potentiate the number of false negatives, *i.e.*, the number of true intrusions that pass by undetected. Given the value and sensitivity of data warehouse data, it is preferable to define low thresholds for the intrusion detection processes. However, this typically generates an extremely high number of alerts that mostly turn out to be false alarms, wasting time and resources. There can typically be a significant amount of alerts with low probability of referring to an intrusion, but those alerts however may produce a very high negative impact on the business, given that DIDS typically do not assess the damage that those intrusions can produce on the business. Furthermore, not all intrusions represent the same potential amount of danger to the enterprise.

In this work we propose a risk exposure method that evaluates the risk to the enterprise represented by each alert without excluding any of them, given the probability that it really refers to an intrusion and the potential impact that the action may produce on the business. This allows considering all generated alerts instead of excluding any of them just because they have low probability thresholds. Ranking the alerts using a measure of risk exposure enables checking them by their order of importance, which means that security staff will spend time and resources more efficiently, by quickly dealing with intrusions that can produce

greater damage rather than wasting time checking for intrusions that represent a lower risk of damage. Considering that none of the generated alerts are discarded and that ranking them by the risk they present to the enterprise, makes the proposed risk exposure method a much more reliable and efficient alert management approach than those using correlation techniques.

3.3.5. Fine-Tuning Intrusion Detection Features

In the proposed DIDS approach, each individual feature can generate intrusion alerts. The diversity of user behavior characteristics caught by each feature in each data warehouse environment depends on heterogeneous (and sometimes unpredictable) events such as the business context itself and the role played by each user, for example. This means that the same feature can produce very different false positive (*i.e.*, false alarm), true positive (*i.e.*, real intrusions detection), true negative (*i.e.*, true normal user behavior) and false negative (*i.e.*, intrusions that pass undetected) rates in different data warehousing environments.

Although in most data warehouses it may be very difficult to define *a priori* which features should be deemed as more efficient to the intrusion detection processes, the DIDS should be able to fine tune its sensitivity over time. Considering that the features that produce the best intrusion detection results are the most reliable for the intrusion detection processes, these processes should be able to reflect the relative individual efficiency between the complete set of feature to improve the overall results.

The DIDS proposed in this thesis uses a calibration technique that computes a measure to assure that the features that show a higher efficiency in intrusion detection are those who's alerts have higher probability of referring true intrusions. This is made effective in our approach by using this measure in the risk exposure method to assess the probability of each alert, given the feature that generated it, *i.e.*, the feature's efficiency measure is directly linked with the probability that the generated alert refers to a true intrusion. Through time, the system is self-adaptive by fine-tuning each feature's measure according to its intrusion detection efficiency, given its true positive and false positive rates.

3.4. Summary

In this chapter we presented the middle tier that enables the integration of the proposed data masking, encryption and intrusion detection for data warehousing environments, and described each of its components.

We also described the information flow and how each individual component works within the execution path of each individual user action to form an overall security solution that deals with those actions in real-time.

The guidelines that drove the development of each data masking, encryption and intrusion detection solution proposed in this thesis were also presented. The following chapters will explain in detail how each of these solutions operate and demonstrate their efficiency.

Chapter 4

MOBAT: A Data Masking Solution for Data Warehouses

The irreversibility and lack of proven security strength attributed to data masking routines have made them an unacceptable choice when it comes to securing sensitive data in live production and reporting databases [Natan, 2005; Ravikumar *et al.*, 2011]. On the other hand, data masking is the main choice for generating test databases for software development environments or when there is a need to publish data that has values with privacy issues. However, we argue that it may be worth considering the usage of a reversible data masking solution in a data warehousing context, as it can effectively provide an alternative solution for protecting data with some level of security strength while introducing low overheads in database storage space and response time performance.

In this chapter, we propose MOBAT (MOdulus BAsed data masking Technique), a low cost and straightforward data masking technique for numerical values that aims at balancing the tradeoff between data security and database performance. The data masking function uses the MOD-modular operator (which returns the remainder of a division expression) and simple arithmetic operations to mask data. Storage space overhead is avoided by preserving each masked column's datatype and by simply using SQL rewriting to mask and unmask values. This also allows avoiding I/O and network bandwidth bottlenecks by discarding data roundtrips between the database and the masking and unmasking mechanisms.

Note that this proposal does not intend to replace any standard encryption algorithms currently available as built-in packages in most DBMS, but rather should be viewed as an alternative solution for protecting the confidentiality of DW data. The main objective is to provide a significant level of security while introducing very small overheads in storage space and database performance, *i.e.*, acceptable tradeoffs between security and

performance, which is a critical issue in order to assure the feasibility of these solutions in DWs.

To evaluate our proposal, we include experiments using two leading commercial DBMS, Oracle 11g and Microsoft SQL Server 2008, and one open-source DBMS, MySQL Server 5.5. The experiments allow to compare the proposed data masking solution against the built-in AES (with 128 bit and 256 bit security) and 3DES168 encryption algorithms provided in the referred DBMS, as well as research state-of-the-art proposals such as Order-Preserving Encryption (OPES) and Salsa20 (alias Snuffle), using the TPC-H decision support benchmark and a real-world sales DW.

The remainder of this chapter is organized as follows. In Section 4.1 we present and describe our masking technique and point out the main issues regarding its use, while Section 4.2 describes its functional architecture. In Section 4.3 we discuss our solution's security and performance issues. Section 4.4 presents the experimental evaluations that were conducted using the well-known TPC-H decision support benchmark and a real-world DW to assess the proposed data masking technique's performance and compare it against standard and state-of-the-art encryption algorithms. Section 4.5 includes a discussion on the proposed data masking solution and on the results obtained in the experiments. Finally, Section 4.6 presents our conclusions.

4.1 MOBAT Masking Expression

Generally, most facts in DWs are columns with numerical values [Kimball and Ross, 2013]. Since fact tables usually represent more than 90% of the DW's total size [Kimball and Ross, 2013], it is fair to assume that numeric type columns also represent the largest portion of business data. The solution proposed in this chapter aims at masking the DW's numerical values while introducing small overheads in the computational efforts for query processing.

Our MOdulus-BAsed data masking Technique (MOBAT), which allows replacing sensitive data with realistic (but not real) data without heavily impacting database performance, is based on a quite simple masking expression. Assume 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 expression depends on the following predefinitions:

- K_1 is a 128 bit random generated value, constant for table 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 the column's datatype. 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 in [1; 2^{128}], represented by $K_{3,j}$.

Assume each value to be masked represented 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:

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

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):

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

Given that an independent value of $K_{3,j}$ is required for each row, 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, which should be avoided at all cost due to the typical enormous number of rows in fact tables. In order to avoid table joins in query processing when using MOBAT, the values of $K_{3,j}$ must be stored along with each row j in table T. To accomplish this, there are two possible solutions:

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

The second option implies additional efforts and amount of 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 physically included with the original data in each row from the start; the second option may make it to be physically stored apart from the remaining original data in the table because it is added *a posteriori* to its

creation. The impact on database performance can be compared be observing the results in Section 4.4.

A third option for defining $K_{3,j}$ values which speeds 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 security 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 Section 4.4.

As a simple example on how MOBAT is applied, consider the following: assume a table T that requires two masked columns, *Column1* and *Column2*. Suppose that the generated values for masking keys $K_1 = 9264$ for table T and $K_{2,1} = 12$ and $K_{2,2} = 78254$ for each respective column. Table 4-1 shows the original data for T on the left and its resulting masked content on the right, represented as T'.

Table 4-1. Example of original dataset and resulting MOBAT masked dataset

T – Original dataset		
Column1	*Column2*	$K_{3,j}$
11	91873	7537
2	38824	1808
18	71624	29636
19	38824	50877
15	84624	34997
12	46926	41395

T' – MOBAT Masked dataset		
Column1'	*Column2'*	$K_{3,j}$
22	162590	7537
6	115270	1808
22	148034	29636
22	112521	50877
22	155673	34997
17	120841	41395

It can be seen in Table 4-1 that the same original values of *Column2* result in different masked values and that the same masked values in *Column1'* also correspond to different original true values in *Column1*, achieving apparent randomness. Of course, this is a very small dataset used only to illustrate these features. We discuss MOBAT's security issues further on in Section 4.3. In the next section we explain how to query the masked database.

4.2 Functional Architecture

The functional architecture for using MOBAT in practice is shown in Figure 4-1, and comprises three key entities:

- The masked database and its DBMS;
- The MOBAT security middleware interface;
- User/client interfaces to query the masked database.

The *MOBAT middleware interface* acts as a broker between the masked database DBMS and the user interfaces, using the MOBAT masking and unmasking methods, ensuring that the queried data is securely processed and proper results are returned to those interfaces. All communications are executed through SSL/TLS secure connections, to protect SQL instructions and returned results between the system's entities. In the *Black Box*, the middleware will store all the generated masking keys and predefined data access policies for the database to which it concerns.

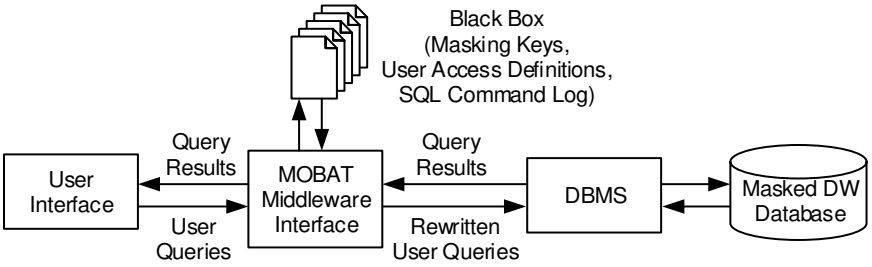


Figure 4-1. The MOBAT Data Security Architecture

The *Black Box* is stored in the *Security Framework Database* database server, as described in Chapter 3, and there is one *Black Box* created for each masked DW database. 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; Oracle, 2010a]. However, contrarily to what happens in Oracle, where the DBA is free to access the Oracle Wallet whenever s/he wishes, in our solution only the MOBAT middleware itself can access the *Black Box*, *i.e.*, absolutely no user has direct access to its content because it is encrypted using the AES standard encryption algorithm [AES, 2001] with a 256 bit key only known by MOBAT.

The MOBAT middleware also creates a historical command log for recording all the instructions and actions executed against the database, for auditing and control purposes. 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 or restoring a replica that has been previously backed up.

Masking keys' privacy depends on where the keys are stored and who has access to them. Our solution uses three masking keys (K_1, K_2 and K_3): two are private and one is public. The private masking keys are generated by the MOBAT middleware, 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 the MOBAT middleware, which will store a copy of those instructions in the *Black Box* command history log.

Each time a user requests the execution of a query or any other action, the MOBAT middleware 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 user interface that issued the request. Thus, MOBAT is transparently used, since SQL command rewriting is transparently managed by the middleware. The only change required to user applications is that commands should be sent to the middleware, instead of directly to the DBMS.

To mask a database, a DBA must require this action through the MOBAT middleware. After inputting the DBA login and database connection information, the MOBAT middleware will attempt to log on to that database. If it succeeds, it then scans all the data access policies defined in the database for identifying authorized users and respective permissions. The *Black Box* is then created for that database 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 is also created, as explained earlier. Afterwards, the tables and columns to be masked are chosen by the DBA. All the required private masking keys for each table and column are then generated, encrypted by an AES256 algorithm and stored in the respective *Black Box*.

Finally, the MOBAT middleware applies the data masking formula on all rows of all columns to be masked, replacing the original values with the

new masked values. Inserting new data or modifying or deleting existing data must always be done through the MOBAT middleware, which applies the masking routine to any value referring to any masked column, and stores the masked value directly in place for update and insert actions. Contrarily to most standard commercial data masking solutions, MOBAT also allows reversing the masked database back to its original data, if masking is no longer needed.

Whenever user applications wish to execute a query, they submit it to the MOBAT middleware instead of directly querying the database. The middleware 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 middleware 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 each of those tables and columns, as well as the additional $K_{3,j}$ key columns used by MOBAT in those tables.

As an example, suppose the *LineItem* table of the TPC-H benchmark [TPC-H] has three numerical fact columns (i = 3) (*L_Quantity*, *L_ExtendedPrice*, 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_Discount) 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 middleware and submitted to the DBMS is as follows:

```
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 shown in the example, query parsing and rewriting is a straightforward operation, replacing each masked column with their respective unmasking Formula (2). This is valid for any type of query, including equality and range queries, as well as built in functions. These changes to the queries are handled transparently by the middleware and kept hidden from the users. Only the query results are returned to the user interface.

4.3 Security Issues

In this section we discuss the security issues concerning the use of the proposed data masking technique. We present the threat model, explain why we use the MOD operator as the base operation for the masking expression and highlight the advantages of having data-at-rest masked at all times, and describe the attack costs for breaking MOBAT's security.

4.3.1 Threat Model

All user queries and instructions that come through are managed by the MOBAT middleware, which transparently parses and rewrites them to query the DBMS and retrieve the intended results. The stored copy of those commands can never be changed or erased, and users never see the rewritten instructions. For security purposes, any historical logging on the DBMS should be shut off or made secure (*e.g.* via encryption) before requesting the execution of the rewritten instructions, so that they are not stored in the DBMS as plain text, since this would disclose the masking keys. Note that for security auditing and to be able to comply with legal auditing regulations, the MOBAT command log always stores a copy of all the issued user commands. All communications between user applications, the MOBAT middleware and the DBMS are performed through encrypted SSL/TLS connections. In what concerns the *Black Box*, all contents are

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 MOBAT middleware allows any user with administration privileges to query the read-only historical command log, so anyone can watch over anyone to check for misuse. All database access is controlled by the middleware, extracting the predefined data access policies in the first instantiation with the database to mask, from the data access policies previously defined using the DBMS. Subsequent changes in data access policies by DBAs must be done through the MOBAT middleware. As these changes are also stored in the *Black Box* history command log, changes in data access policies with the purpose of executing malicious actions can always be checked.

The only allowed access to the masking keys in the *Black Box* is done by the middleware, which is managed only by the middleware itself. We assume that the DBMS is a trusted server because it is expected to correctly execute the SQL commands that are sent to it. However, we consider the database as untrusted as it may be compromised by an attacker able to bypass the network and MOBAT access controls, gaining direct access to the database itself. We also assume that the MOBAT expressions are public, so the attacker can replicate the masking and unmasking mechanisms, meaning that the goal of the attacker is to obtain the private masking keys in order to break security.

4.3.2 Using Column Datatype Key Lengths and Consecutive MOD Operations

In order to minimize the impact in data storage space and query response time overheads, the private keys for each column have the same length as the defined column datatype. Although this might imply using small sized keys and make the masking expression to produce a small amount of possible distinct outputs, it should not be very significant from a practical perspective. For example, if the masked column has a *bit* datatype, there is no point in generating masked values in a range of $[0\ldots2^{128}]$, since the attacker probably knows *a priori* that it can only hold a 0 or 1 by observing the column's name. Given that the best practices in DWs suggest using meaningful names for the columns in the database tables for the sake of

readability [Kimball and Ross, 2013], this also suggests that there is not much to gain in incrementing the size of the masked output range of values because this will probably not imply an increase of the level of security strength.

As previously mentioned, the MOD operator is used as the main operation in the masking expression because it is non-injective, given that for X MOD $Y = Z$, the same output Z, considering Y as a constant, can have an undetermined number of possibilities in X as an input that will generate the same value Z. This is illustrated in Section 4.1 (Table 4-1), where the same original values originate different masked values and vice-versa. Since MOD operations are non-injective, the masked outputs are also non-injective. Given that injectivity is a required property for invertibility, the proposed masking expression is thus not directly invertible, enforced by using two consecutive MOD operations. Thus, the objective of the attacker should be focused on obtaining the private masking keys in order to break security.

4.3.3 Data-at-rest is Always Masked

Since MOBAT operates simply by rewriting SQL commands to be processed against the data, this enables running SQL directly against the masked data, which means that the data-at-rest stored within the database files is masked at all times.

This also means that even if someone gains direct access to the database, s/he will only see masked data values. As the masked values are realistic-looking and maintain their original column datatypes, if an attacker was to query the database s/he would view expected values, although they would be incorrect. This means that MOBAT would potentially be able to produce misleading effects against attackers.

4.3.4 Attack Costs on MOBAT

As known (and as we assume the attackers have access to the masking expression), the level of security of data masking or encryption solutions does not depend on its secrecy, but on its keys [Elminaam *et al.*, 2010; Nadeem and 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, because the achievement of higher complexity often implies the consumption of a higher amount of resources and processing time.

As mentioned before, there keys are used in our proposal: 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, typically varying between 1 and 128 bits. This means that our technique 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.

This is however the worst case scenario for the attacker and executing a chosen ciphertext attack would allow the attacker to reduce the key search space in the following way (considering the masking expression defined in Formula (1)):

Consider $x'_{1,i}$ and $x'_{2,i}$ as the masked values for two given rows (respectively 1 and 2) of column i and $x_{1,i}$ and $x_{2,i}$ as their respective original true values, *i.e.*, $x'_{1,i} = (R_1, C_i)'$, $x'_{2,i} = (R_2, C_i)'$, $x_{1,i} = (R_1, C_i)$, and $x_{2,i} = (R_2, C_i)$. In this case,

$$x'_{1,i} = x_{1,i} - ((K_{3,1} \text{ MOD } K_1) \text{ MOD } K_{2,i}) + K_{2,i}$$

$$x'_{2,i} = x_{2,i} - ((K_{3,2} \text{ MOD } K_1) \text{ MOD } K_{2,i}) + K_{2,i}$$

Knowing that $K_{3,j}$ is a public value key, if the attacker chooses two masked outputs where $K_{3,j}$ have very small values (close to zero), then it is highly probable that those values are smaller than the K_1 private key, *i.e.*, $K_{3,1} < K_1$ and $K_{3,2} < K_1$. In this case, the masking expression would be reduced to:

$$x'_{1,i} = x_{1,i} - (K_{3,1} \text{ MOD } K_{2,i}) + K_{2,i}$$

$$x'_{2,i} = x_{2,i} - (K_{3,2} \text{ MOD } K_{2,i}) + K_{2,i}$$

where all values are known except for the private key $K_{2,i}$.

Building up an expression with the difference between both variables, we have:

$$(x'_{1,i} - x'_{2,i}) = (x_{1,i} - (K_{3,1} \text{ MOD } K_{2,i}) + K_{2,i}) - (x_{2,i} - (K_{3,2} \text{ MOD } K_{2,i}) + K_{2,i})$$

$$= (x_{1,i} - (K_{3,1} \text{ MOD } K_{2,i})) - (x_{2,i} - (K_{3,2} \text{ MOD } K_{2,i}))$$

Finally, isolating the expressions with known values from those having unknown values:

$$(x'_{1,i} - x'_{2,i}) - (x_{1,i} - x_{2,i}) = (K_{3,1} \text{ MOD } K_{2,i} - K_{3,2} \text{ MOD } K_{2,i})$$

which would significantly reduce the search space for $K_{2,i}$. After breaking $K_{2,i}$ the attacker could then discover K_1 in a similar manner by using the original expressions of Formula (1) for the masked values.

To evaluate the database performance when using the proposed masking solution, the following section presents experimental results obtained by MOBAT against standard and state-of-the-art encryption solutions.

4.4 Experimental Evaluation

To evaluate the proposed masking technique, we used the TPC-H decision support benchmark [TPC-H] (1GB and 10GB scale sizes) and a real-world sales DW storing one year of commercial data taking up 2GB of storage space (full description of TPC-H can be found in [TPC-H Specifications], while full description of the sales DW including its description, size, data schema and query workload can be seen in Appendix A). We tested all scenarios using the Oracle 11g and Microsoft SQL Server 2008 R2 DBMS with default settings, on a Pentium IV 2.8GHz CPU with a 1.5TB SATA hard disk and 2GB of RAM, 512MB of which devoted to the database memory cache. Oracle 11g ran on Windows XP Professional, while SQL Server ran on Windows 2003 Server.

Although we include experiments from both DBMS, it is not our aim to compare the results between the DBMS, but rather to compare the performance of each standard and research solution with that of MOBAT within the same DBMS.

The columns chosen for testing the masking solution were those referring to numerical datatype columns belonging to the fact tables. 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 connected to it. In the TPC-H setups, four columns of *LineItem* were masked (*L_Quantity*, *L_ExtendedPrice*, *L_Tax* and *L_Discount*), given that 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 our solution is column-based, for fairness we compare it with column-based AES128 and 3DES168 encryption algorithms. Note that tablespace encryption has functional primitives that speedup performance, which makes it unfair to compare it with column-based techniques [Huey, 2008; Oracle, 2010a]. Moreover, best practices for encryption in the documentation from both DBMSs [Huey, 2008; Oracle, 2010a] recommend using column-based encryption when the sensitive data consists on a small number of well-defined columns. We used the AES128 and 3DES168 Transparent Data Encryption (TDE) algorithms provided by both DBMS for comparison because they are, respectively, the fastest and slowest available algorithms in those DBMS [Huey, 2008; Oracle, 2010a], and OPES [Agrawal *et al.*, 2004] and Salsa20/20 [Bernstein, 2005; Bernstein, 2008]. OPES and Salsa20 were implemented using C#.

Table 4-2 shows the experimental encryption/masking scenarios. The results for MOBAT where the new $K_{3,j}$ masking key columns are added to the fact tables are referenced as *MOBAT_AddCol*; and the results for MOBAT where the $K_{3,j}$ columns are added in the fact tables from the start and completely rebuilt are referenced as *MOBAT_CreateCol*. The results for the tests using an existing table column as $K_{3,j}$ instead of adding a new column to the fact table is referred 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.

Table 4-2. Experimental 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
OPES	Data encrypted with Order-Preserving Encryption [Agrawal et al., 2004]
Salsa20	Data encrypted with Salsa20/20 encryption [Bernstein, 2008]
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 $K_{3,j}$

All loading time and query response time results shown in this section are an average of six executions in each described setup/scenario. Given the resulting standard deviations are relatively small assures that this number of executions if sufficient enough to be representative for comparisons. The complete set of results and respective statistical measures can be seen in Appendix B.

4.4.1 Analyzing Storage Space

Figures 4-2a and 4-2b respectively show the results of total data storage space (in MB) and percentage of storage space overhead for loading the TPC-H 1GB *LineItem* fact table in Oracle, while Figures 4-3a and 4-3b show the same results in SQL Server. To execute the loading processes, all indexes were dropped on the fact tables.

As shown, the standard storage space for the TPC-H LineItem fact table without using any sort of encryption or masking solution takes up 772MB of storage space in Oracle and 1237MB of storage space in SQL Server. There is a significant difference in the standard data storage space sizes between the DBMS because they have distinct ways of storing data, in which Oracle standardly uses a type of compression algorithm while SQL Server does not.

Note that the resulting values registered for MOBAT refer to *MOBATAddCol* (adding a column to the fact table) and *MOBATCreateCol* (recreating the fact table with the addition of a column), involving the

creation of an extra public key column (referred to as $K_{3,j}$ as described in the previous sections). The *MOBAT ColKey* setup (in which the column used as the public key column is a column that originally belongs to the fact table) is not included, since it does not require changing the fact table data structure to handle the implementation of MOBAT. Thus, the overhead for *MOBATColKey* is actually inexistent, making it the best technique in what concerns avoiding storage space overhead.

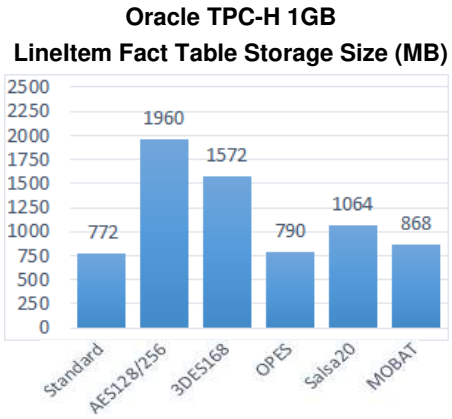


Figure 4-2a. Storage Size in Oracle for the TPC-H 1GB Fact Table per Solution

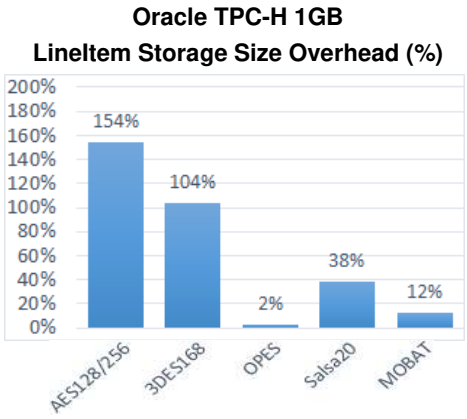


Figure 4-2b. Storage Overhead (%) in Oracle for the TPC-H 1GB Fact Table per Solution

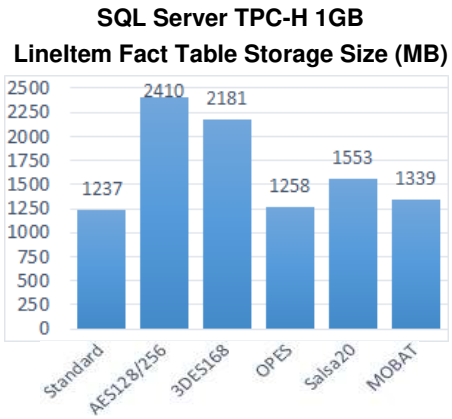


Figure 4-3a. Storage Size in SQL Server for the TPC-H 1GB Fact Table per Solution

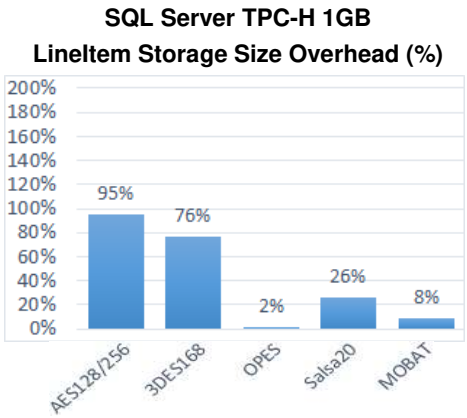


Figure 4-3b. Storage Overhead (%) in SQL Server for the TPC-H 1GB Fact Table per Solution

As shown, OPES and MOBAT produce much smaller storage space overheads than the remaining solutions. OPES shows a 2% overhead for both DBMS, corresponding to an extra 18MB of storage space in Oracle and 21MB in SQL Server, and 12% and 8% overhead for MOBAT respectively in Oracle and SQL Server, corresponding to an extra 96MB and 102MB of storage space. OPES produces a small storage space overhead because the smallest and largest gaps between the sorted values for its target distributions are mostly small in the TPC-H database. This attests what is explained in [Agrawal *et al.*, 2004], where the authors express that they would expect a small increase of the required space for the ciphertexts.

Salsa20 introduces more storage space overhead than OPES and MOBAT, namely 38% in Oracle, corresponding to adding 292MB, and 26% in SQL Server, which adds 316MB of extra storage space. The standard encryption solutions produce the highest overhead, with AES being the worst by requiring 154% in Oracle and 95% in SQL Server of storage space overhead, corresponding to respectively adding 1188MB and 1173MB and 154%, while 3DES168 produced a storage space overhead of 104% in Oracle and 76% in SQL Server, respectively corresponding to 800MB and 944MB of extra storage space.

Figures 4-4a and 4-4b respectively show the results of total data storage space (in MB) and percentage of storage space overhead for loading the TPC-H 10GB *LineItem* fact table in Oracle, while Figures 4-5a and 4-5b show the same results in SQL Server. Figures 4-4a to 4-5b show that the extra storage space added to the 10GB database by each solution is approximately proportional to those of the 1GB database, which means ten times bigger. Thus, the analysis of the results for the 10GB sized TPC-H database is similar to that of the 1GB sized TPC-H database.

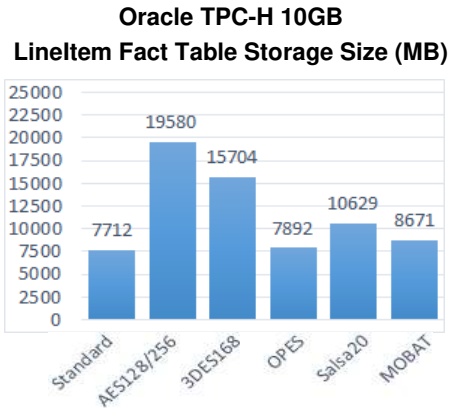


Figure 4-4a. Storage Size in Oracle for the TPC-H 10GB Fact Table per Solution

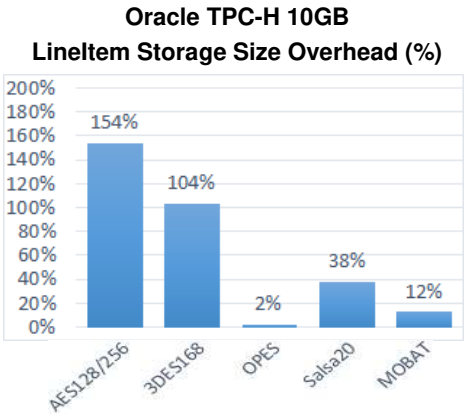


Figure 4-4b. Storage Overhead (%) in Oracle for the TPC-H 10GB Fact Table per Solution

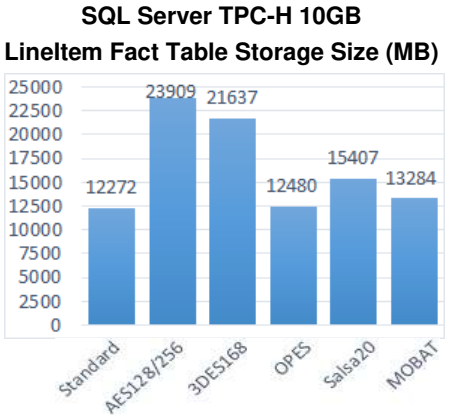


Figure 4-5a. Storage Size in SQL Server for the TPC-H 10GB Fact Table per Solution

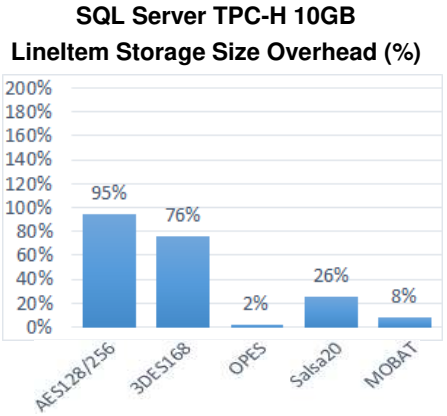


Figure 4-5b. Storage Overhead (%) in SQL Server for the TPC-H 10GB Fact Table per Solution

Figures 4-6a and 4-6b show the total data storage space (in MB) and percentage of storage space overhead for loading the Sales DW fact table in Oracle, while Figures 4-7a and 4-7b show the same results in SQL Server. It can be seen that the standard storage space for the Sales fact table without using any encryption or masking solution takes up 1664MB of storage space in Oracle and 1932MB of storage space in SQL Server.

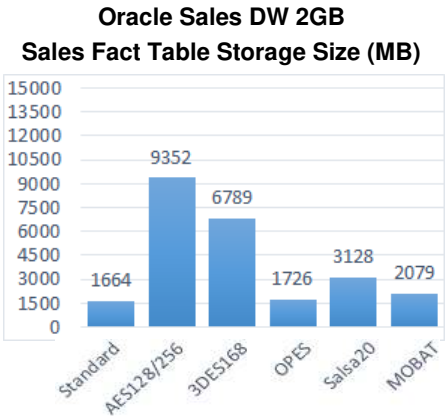


Figure 4-6a. Storage Size in Oracle for the Sales DW Fact Table per Solution

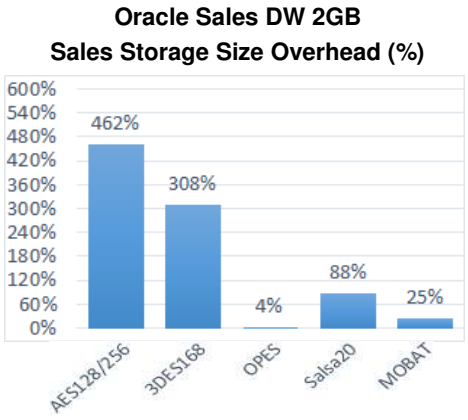


Figure 4-6b. Storage Overhead (%) in Oracle for the Sales DW Fact Table per Solution

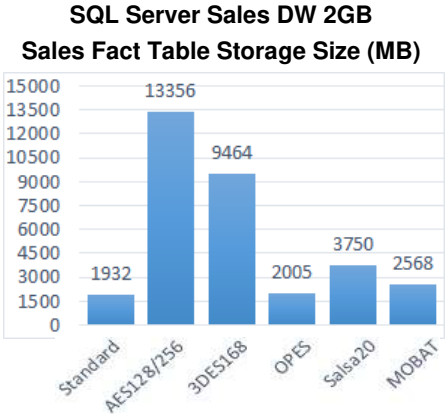


Figure 4-7a. Storage Size in SQL Server for the Sales DW Fact Table per Solution

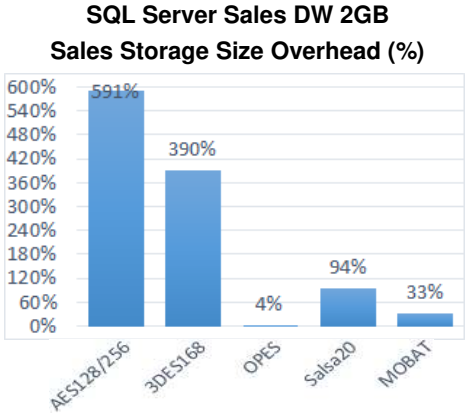


Figure 4-7b. Storage Overhead (%) in SQL Server for the Sales DW Fact Table per Solution

As shown in Figures 4-6a to 4-7b, OPES and MOBAT continue to produce much smaller storage space overheads than the remaining solutions, similarly to the occurred with TPC-H. OPES shows a 4% overhead for both DBMS, corresponding to an extra 64MB of storage space, and MOBAT presents 25% and 33% overhead respectively in Oracle and SQL Server, corresponding to an extra 415MB and 636MB of storage space. OPES continues to present the best results because of the same reasons that were previously mentioned, *i.e.*, the data values in the Sales DW allow it to

generate target distributions which do not require much additional space to store the ciphertexts.

Salsa20 also introduces more storage space overhead than OPES and MOBAT, namely 88% in Oracle, corresponding to adding 1464MB, and 94% in SQL Server, which adds 1818MB of extra storage space. The standard encryption solutions produce the highest overhead, with AES also being the worst by requiring 462% in Oracle and 591% in SQL Server of storage space overhead, corresponding to respectively adding 7688MB and 11424MB of storage space, while 3DES168 produced a storage space overhead of 308% in Oracle and 390% in SQL Server, respectively corresponding to 5125MB and 7532MB of extra storage space.

Tables 4-3, 4-4 and 4-5 summarize the fact table storage space results respectively for the TPC-H 1GB, TPC-H 10GB and Sales DW, for each DBMS, highlighting the best solutions in each case.

Table 4-3. TPC-H 1GB Lineitem Fact Table Storage Size Overhead

	Oracle TPC-H 1GB Storage Size (Overhead)	**SQL Server TPC-H 1GB Storage Size (Overhead)**
Standard	772MB	1237MB
AES128/256	1960MB (+1188MB / 154%)	2410MB (+1173MB / 95%)
3DES168	1572MB (+800MB / 104%)	2181MB (+944MB / 76%)
OPES	790MB (+18MB / 2%)	1258MB (+21MB / 2%)
Salsa20	1064MB (+292MB / 38%)	1553MB (+316MB / 26%)
MOBAT	868MB (+96MB / 12%)	1339MB (+102MB / 8%)

Table 4-4. TPC-H 10GB Lineitem Fact Table Storage Size Overhead

	Oracle TPC-H 10GB Storage Size (Overhead)	**SQL Server TPC-H 10GB Storage Size (Overhead)**
Standard	7712MB	12272MB
AES128/256	19580MB (+11868MB / 154%)	23909MB (+11637MB / 95%)
3DES168	15704MB (+7992MB / 104%)	21637MB (+9365MB / 76%)
OPES	7892MB (+180MB / 2%)	12480MB (+208MB / 2%)
Salsa20	10629MB (+2917MB / 38%)	15407MB (+3135MB / 26%)
MOBAT	8671MB (+959MB / 12%)	13284MB (+1012MB / 8%)

Table 4-5. Sales DW 2GB Fact Table Storage Size Overhead

	Oracle Sales DW 2GB Storage Size (Overhead)	SQL Server Sales DW 2GB Storage Size (Overhead)
Standard	1664MB	1932MB
AES128/256	9352MB (+7688MB / 462%)	13356MB (+11424MB / 591%)
3DES168	6789MB (+5125MB / 308%)	9464MB (+7532MB / 390%)
OPES	1726MB (+62MB / 4%)	2005MB (+73MB / 4%)
Salsa20	3128MB (+1464MB / 88%)	3750MB (+1818MB / 94%)
MOBAT	2079MB (+415MB / 25%)	2568MB (+636MB / 25%)

4.4.2. Analyzing Loading Time

In this subsection, we analyze the loading time for populating the fact table of each DW, which is affected by both the execution of the masking or encryption processes and the need to write additional data taking up extra storage space. Figures 4-8a and 4-8b respectively show the results of total loading time (in seconds) and percentage of time overhead for loading the TPC-H 1GB *LineItem* fact table in Oracle, while Figures 4-9a and 4-9b show the same results in SQL Server. It can be observed that the standard loading time for the TPC-H LineItem fact table without using any sort of encryption solution is 310 seconds in Oracle and 212 seconds in SQL Server.

As shown in the figures, MOBAT produces much smaller loading time overheads than the remaining solutions, introducing between 3% and 8% of overhead in both DBMS, respectively corresponding to adding between 6 and 25 seconds of loading time. OPES comes after MOBAT in loading time performance, showing an overhead of 49% in Oracle and 44% in SQL Server, which respectively correspond to adding 151 and 93 seconds. Salsa20 introduces more loading time overhead than OPES and MOBAT, namely 73% in Oracle, corresponding to adding 227 seconds, and 70% in SQL Server, which adds 149 seconds of extra loading time.

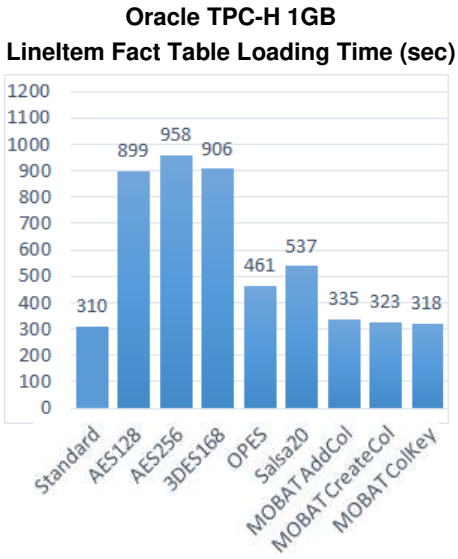


Figure 4-8a. Loading Time in Oracle for the TPC-H 1GB Fact Table per Solution

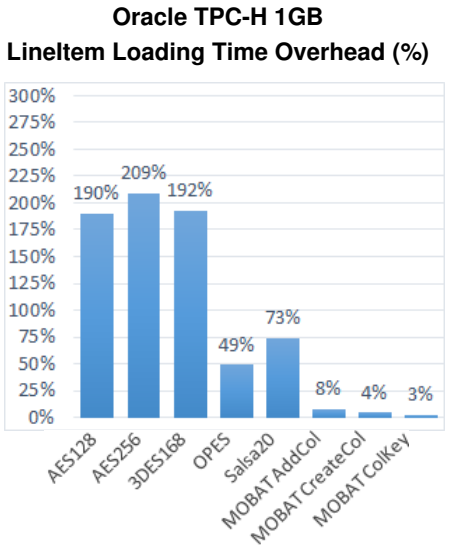


Figure 4-8b. Loading Time Overhead (%) in Oracle for the TPC-H 1GB Fact Table per Solution

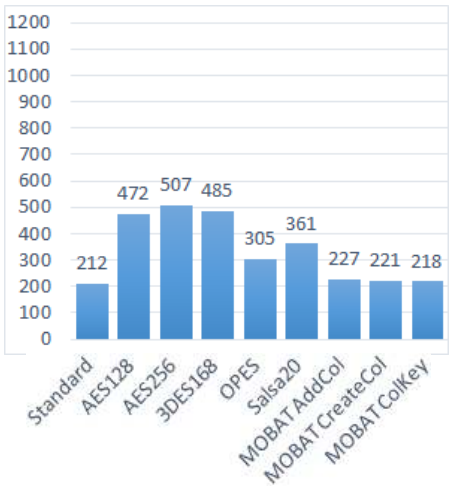


Figure 4-9a. Loading Time in SQL Server for the TPC-H 1GB Fact Table per Solution

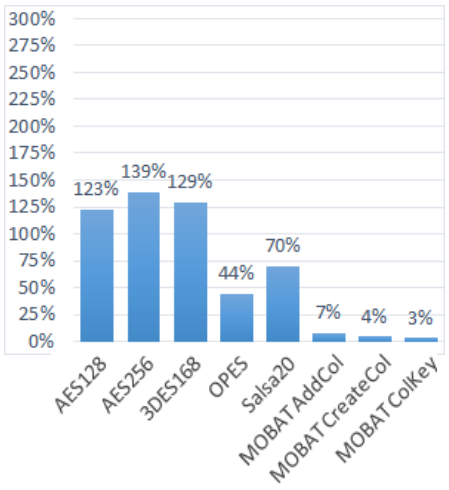


Figure 4-9b. Loading Time Overhead (%) in SQL Server for the TPC-H 1GB Fact Table per Solution

Similarly to what occurred with storage space, the standard encryption solutions produced the highest loading time overheads. AES with 128 bit security produced 190% in Oracle and 123% in SQL Server, respectively corresponding to adding 589 and 260 seconds to the standard loading time. AES with 256 bit security shows an overhead of 209% in Oracle and 139% in SQL Server, respectively corresponding to 648 and 295 seconds of extra loading time. 3DES168 introduces 192% loading time overhead in Oracle, corresponding to adding 596 seconds, and 129% in SQL Server, which adds 273 seconds of extra loading time.

Figures 4-10a and 4-10b respectively show the results of total loading time (in seconds) and percentage of loading time overhead for loading the TPC-H 10GB *LineItem* fact table in Oracle, while Figures 4-11a and 4-11b show the same results in SQL Server.

From observing the results in Figures 4-10a to 4-11b, it can be seen that the extra loading time added to the 10GB database by each encryption solution is approximately over-proportional to those of the 1GB database, as occurred with the storage space, which means slightly over ten times bigger. Thus, the analysis of the results for the 10GB sized TPC-H database is also similar to that of the 1GB sized TPC-H database.

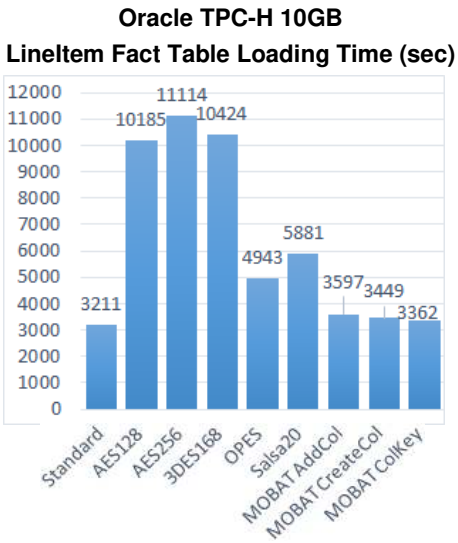


Figure 4-10a. Loading Time in Oracle for the TPC-H 10GB Fact Table per Solution

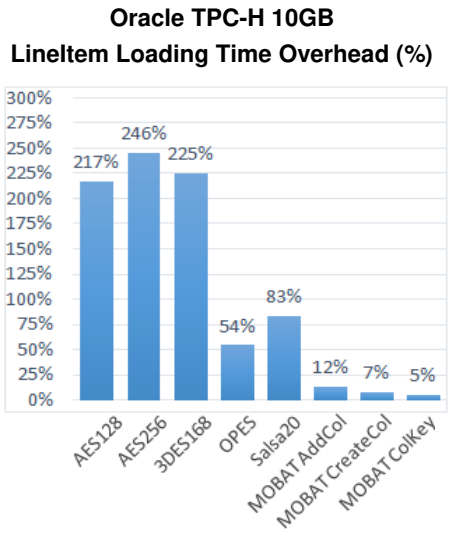


Figure 4-10b. Loading Time Overhead (%) in Oracle for the TPC-H 10GB Fact Table per Solution

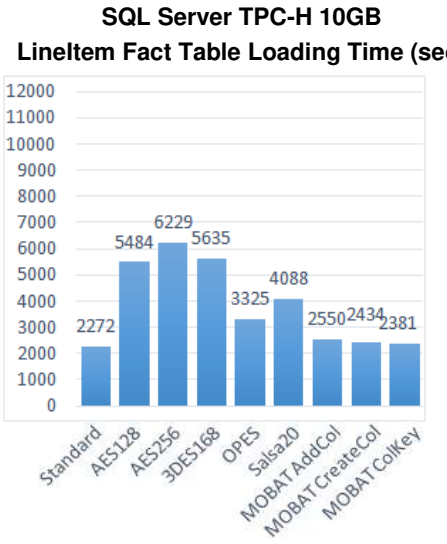


Figure 4-11a. Loading Time in SQL Server for the TPC-H 10GB Fact Table per Solution

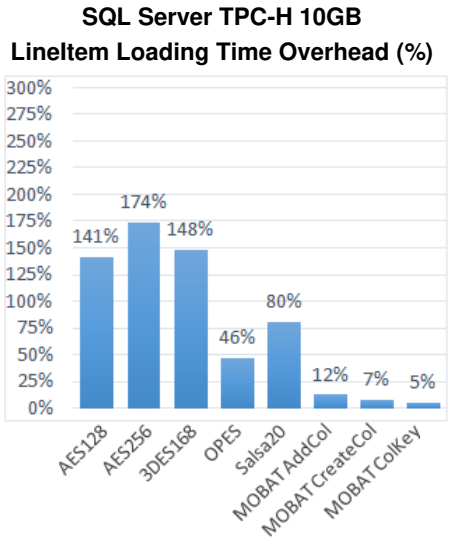


Figure 4-11b. Loading Time Overhead (%) in SQL Server for the TPC-H 10GB Fact Table per Solution

Figures 4-12a and 4-12b respectively show the results of total loading time (in seconds) and percentage of time overhead for loading the Sales DW fact table in Oracle, while Figures 4-13a and 4-13b show the same results in SQL Server. It can be seen that the standard loading time for the Sales fact table without using any encryption solution is 1195 seconds in Oracle and 1247 seconds in SQL Server.

As seen in both figures, MOBAT continues to produce much smaller loading time overheads than the remaining solutions, similarly to the occurred with TPC-H. MOBAT *AddCol* shows 15% and 16% overhead in Oracle and SQL Server, respectively corresponding to an extra 178 and 200 seconds in loading time. MOBAT *CreateCol* shows 9% and 10% in Oracle and SQL Server, corresponding to adding 113 seconds in Oracle and 120 seconds in SQL Server, and when using MOBAT ColKey the loading time overhead was 5% in Oracle and 6% in SQL Server, corresponding to 65 seconds of extra loading time in Oracle and 71 seconds of extra loading time in SQL Server.

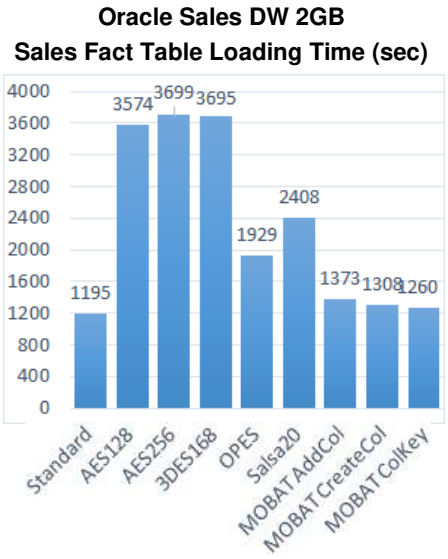


Figure 4-12a. Loading Time in Oracle for the Sales DW Fact Table per Solution

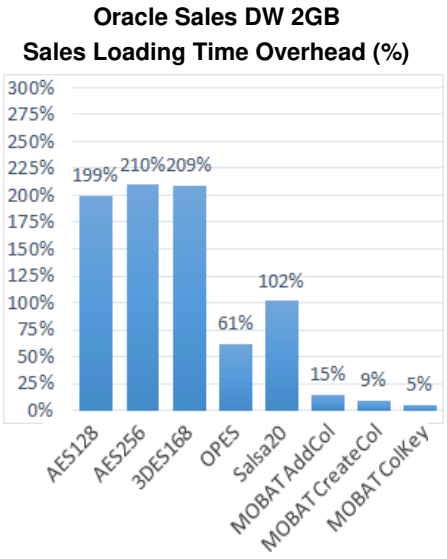


Figure 4-12b. Loading Time Overhead (%) in Oracle for the Sales DW Fact Table per Solution

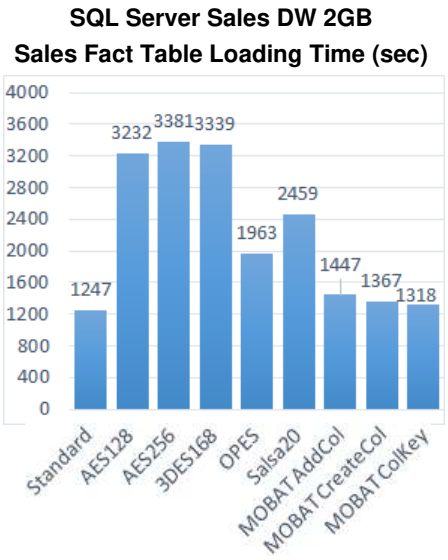


Figure 4-13a. Loading Time in SQL Server for the Sales DW Fact Table per Solution

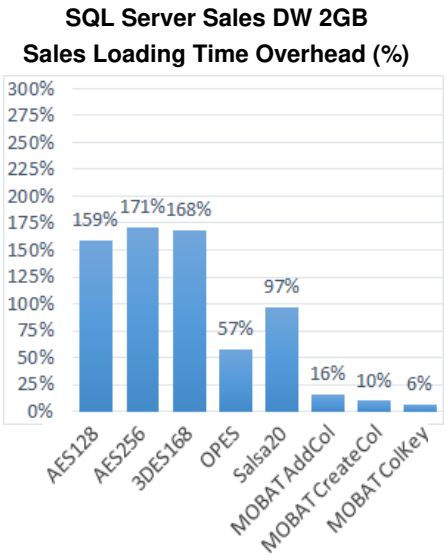


Figure 4-13b. Loading Time Overhead (%) in SQL Server for the Sales DW Fact Table per Solution

OPES comes after MOBAT in loading time performance, showing a 61% overhead in Oracle and 57% in SQL Server, respectively corresponding to an extra 734 seconds and 716 seconds of loading time, and Salsa20 presents 102% and 97% overhead for respectively in Oracle and SQL Server, corresponding to 1213 and 1212 seconds of extra loading time.

The standard encryption solutions continue to produce the highest overhead, where AES with 128 bit security produced 199% in Oracle and 159% in SQL Server, respectively corresponding to adding 2379 and 1985 seconds to the standard loading time. AES with 256 bit security shows an overhead of 210% in Oracle and 171% in SQL Server, respectively corresponding to 2504 and 2134 seconds of extra loading time. 3DES168 introduces 209% loading time overhead in Oracle, corresponding to adding 2500 seconds, and 168% in SQL Server, which adds 2092 seconds of extra loading time.

Tables 4-6, 4-7 and 4-8 summarize the fact table loading time results (in hh:mm:ss format) respectively for the TPC-H 1GB, TPC-H 10GB and Sales DW, for each DBMS, highlighting the best solutions in each case.

Table 4-6. TPC-H 1GB Lineitem Fact Table Loading Time Overhead

	Oracle TPC-H 1GB Loading Time (Overhead)	**SQL Server TPC-H 1GB Loading Time (Overhead)**
Standard Loading Time	00:05:10	00:03:32
AES128	00:14:59 (00:09:49 / 190%)	00:07:52 (00:04:20 / 123%)
AES256	00:15:58 (00:10:48 / 209%)	00:08:27 (00:04:55 / 139%)
3DES168	00:15:06 (00:09:56 / 192%)	00:08:05 (00:04:33 / 129%)
OPES	00:07:41 (00:02:31 / 49%)	00:05:05 (00:01:33 / 44%)
Salsa20	00:08:57 (00:03:47 / 73%)	00:06:01 (00:02:29 / 70%)
MOBAT AddCol	00:05:35 (00:00:25 / 8%)	00:03:47 (00:00:15 / 7%)
MOBAT CreateCol	00:05:23 (00:00:13 / 4%)	00:03:41 (00:00:09 / 4%)
MOBAT ColKey	00:05:18 (00:00:08 / 3%)	00:03:38 (00:00:06 / 3%)

Table 4-7. TPC-H 10GB Lineitem Fact Table Loading Time Overhead

	Oracle TPC-H 10GB Loading Time (Overhead)	SQL Server TPC-H 10GB Loading Time (Overhead)
Standard Loading Time	00:53:31	00:37:52
AES128	02:49:45 (01:56:14 / 217%)	01:31:24 (00:53:32 / 141%)
AES256	03:05:14 (02:11:43 / 246%)	01:43:49 (01:05:57 / 174%)
3DES168	02:53:44 (02:00:13 / 225%)	01:33:55 (00:56:03 / 148%)
OPES	01:22:23 (00:28:52 / 54%)	00:55:25 (00:17:33 / 46%)
Salsa20	01:38:01 (00:44:30 / 83%)	01:08:08 (00:30:16 / 80%)
MOBAT AddCol	00:59:57 (00:06:26 / 12%)	00:42:30 (00:04:38 / 12%)
MOBAT CreateCol	00:57:29 (00:03:58 / 7%)	00:40:34 (00:02:42 / 7%)
MOBAT ColKey	00:56:02 (00:02:31 / 5%)	00:39:41 (00:01:49 / 5%)

Table 4-8. Sales DW 2GB Fact Table Loading Time Overhead

	Oracle Sales DW 2GB Loading Time (Overhead)	SQL Server Sales DW 2GB Loading Time (Overhead)
Standard Loading Time	00:19:55	00:20:47
AES128	00:59:34 (00:39:39 / 199%)	00:53:52 (00:33:05 / 159%)
AES256	01:01:39 (00:41:44 / 210%)	00:56:21 (00:35:34 / 171%)
3DES168	01:01:35 (00:41:40 / 209%)	00:55:39 (00:34:52 / 168%)
OPES	00:32:09 (00:12:14 / 61%)	00:32:43 (00:11:56 / 57%)
Salsa20	00:40:08 (00:20:13 / 102%)	00:40:59 (00:20:12 / 97%)
MOBAT AddCol	00:22:53 (00:02:58 / 15%)	00:24:07 (00:03:20 / 16%)
MOBAT CreateCol	00:21:48 (00:01:53 / 9%)	00:22:47 (00:02:00 / 10%)
MOBAT ColKey	00:21:00 (00:01:05 / 5%)	00:21:58 (00:01:11 / 6%)

4.4.3. Analyzing Query Performance

To analyze the query performance of the masking technique and the selected encryption algorithms, we defined a query workload for each database. The TPC-H workload included the benchmark queries 1, 3, 6, 7, 8, 10, 12, 14, 15, 17, 19 and 20 (which correspond to all queries in TPC-H that access the LineItem fact table). For the Sales DW, the workload was a set of 29 queries, all processing the Sales fact table, as a set of usual decision support reports, gathering daily (9 queries), monthly (9 queries) and annual (11 queries) values, including actions such as row selection, joining,

aggregates, and ordering. These queries represent typical reporting workloads against the fact table for each database. For fairness, databases were optimized in a best practice manner (including primary keys, foreign keys, and referential integrity constraints and join indexes).

As previously mentioned, all response time results are an average obtained from six executions in each scenario on each DBMS. The standard execution time (average of the execution times of the workload against a non-encrypted database) for each scenario is 625, 6155, and 2233 seconds in Oracle 11g, and 580, 5301, and 2211 seconds in SQL Server 2008, for the 1GB, 10GB TPC-H and Sales DW, respectively.

Figures 4-14a and 4-14b respectively show the total workload execution time and its overhead in Oracle and Figures 4-15a and 4-15b show the total workload execution time and overhead in SQL Server, for the TPC-H 1GB database.

It can be seen that MOBAT executes much faster than the remaining solutions, introducing overheads between 22% and 35% of query workload execution time in Oracle, respectively corresponding to adding between 138 and 221 seconds to total execution time, and overheads between 23% and 40% in SQL Server, respectively corresponding to adding between 132 and 233 seconds to total execution time.

All the remaining encryption solutions are approximately leveled and present overheads between 176% and 203% in Oracle, corresponding to adding an extra 1102 to 1270 seconds to total execution time, and overheads between 163% and 195% in SQL Server, corresponding to adding an extra 943 to 1132 seconds to total execution time. Regarding these solutions, Salsa20 was the fastest with AES128 coming afterwards, followed by OPES and AES256, with 3DES168 as the slowest solution. This means that MOBAT produces overheads that are roughly one sixth of the encryption solutions, on average, in the chosen experimental setups.

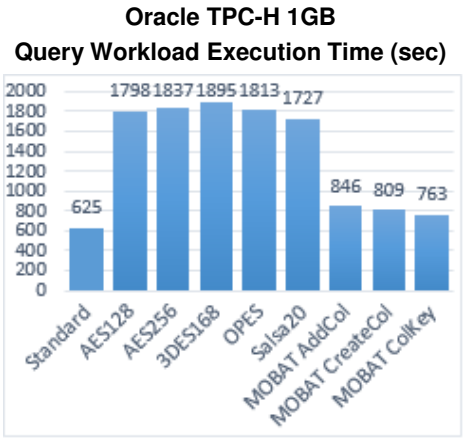


Figure 4-14a. Query Workload Execution Time per Solution in Oracle for TPC-H 1GB

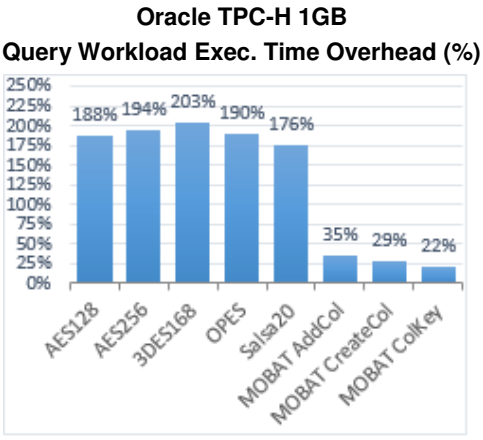


Figure 4-14b. Query Workload Execution Time Overhead (%) per Solution in Oracle for TPC-H 1GB

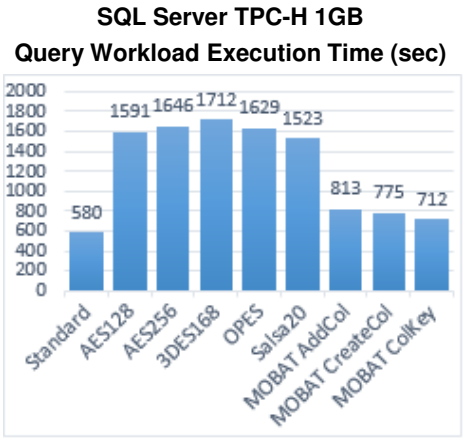


Figure 4-15a. Query Workload Execution Time per Solution in SQL Server for TPC-H 1GB

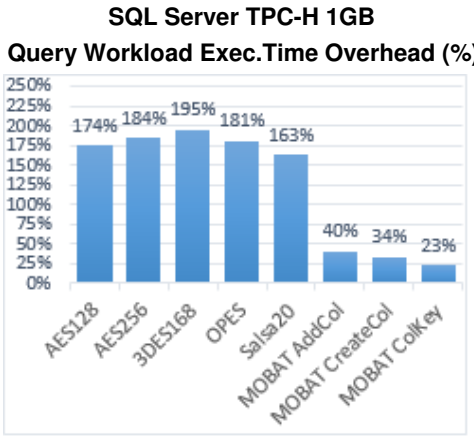


Figure 4-15b. Query Workload Execution Time Overhead (%) per Solution in SQLServer for TPC-H 1GB

Figures 4-16a and 4-16b respectively show the total workload execution time and its overhead in Oracle and Figures 4-17a and 4-17b show the total workload execution time and overhead in SQL Server, for the TPC-H 10GB database. As can be observed, the results lead to similar results as those seen in the TPC-H 1GB database, in what concerns the ranking of the tested solutions. MOBAT remains the solution having the best execution time,

with lower overhead for all scenarios in both DBMS. When compared with the results for the TPC-H 1GB database, it can be seen that the differences between the solutions are slightly enforced with the higher amount of data that need to be processed in the 10GB scale size.

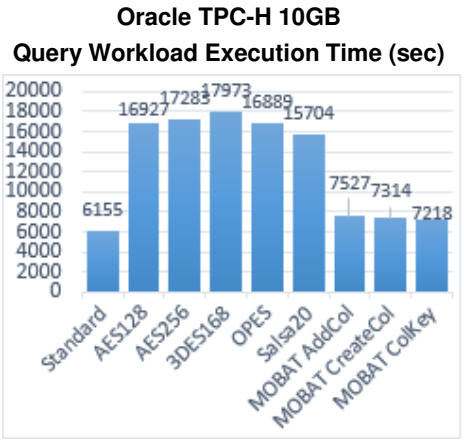


Figure 4-16a. Query Workload Execution Time per Solution in Oracle for TPC-H 10GB

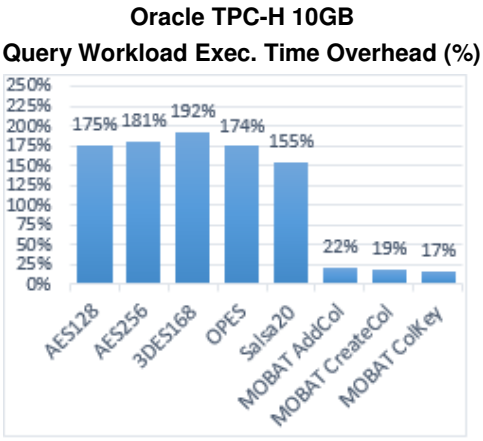


Figure 4-16b. Query Workload Execution Time Overhead (%) per Solution in Oracle for TPC-H 10GB

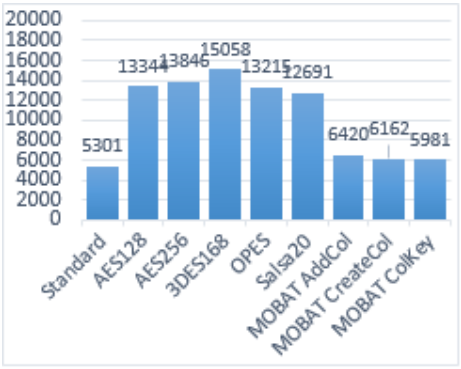


Figure 4-17a. Query Workload Execution Time per Solution in SQL Server for TPC-H 10GB

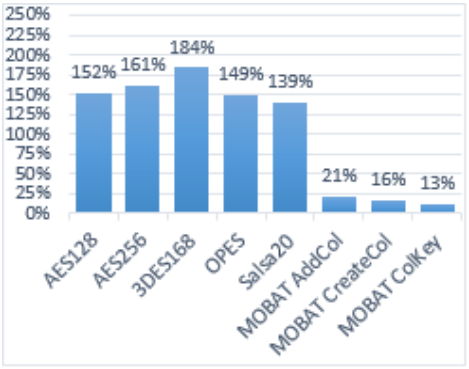


Figure 4-17b. Query Workload Exec. Time Overhead (%) per Solution in SQLServer for TPC-H 10GB

Furthermore, MOBAT executes much faster than the remaining solutions, introducing overheads between 17% and 22% of query workload execution time in Oracle, respectively corresponding to adding between 1063 and 1372 seconds to total execution time, and overheads between 13% and 21% in SQL Server, respectively corresponding to adding between 680 and 1119 seconds to total execution time.

All the remaining encryption solutions are approximately leveled and present overheads between 155% and 192% in Oracle, corresponding to adding an extra 9549 to 11818 seconds to total execution time, and overheads between 139% and 184% in SQL Server, corresponding to adding an extra 7390 to 9757 seconds to total execution time. Regarding these solutions, Salsa20 continues being the fastest, followed by OPES, AES128 and AES256, with 3DES168 as the slowest solution. This means that MOBAT continues to produce overheads that are roughly one eighth to one tenth of the encryption solutions, on average, in the chosen experimental setups, similar to what occurred in the TPC-H 1GB.

Figures 4-18a and 4-18b respectively show the total workload execution time and its overhead in Oracle and Figures 4-19a and 4-19b show the total workload execution time and overhead in SQL Server, for the Sales DW database.

As shown, MOBAT also executes much faster than the remaining solutions in the Sales DW, introducing overheads between 78% and 128% of query workload execution time in Oracle, respectively corresponding to adding between 1733 and 2851 seconds to total execution time, and overheads between 64% and 124% in SQL Server, which respectively correspond to adding between 1426 and 2735 seconds to total execution time.

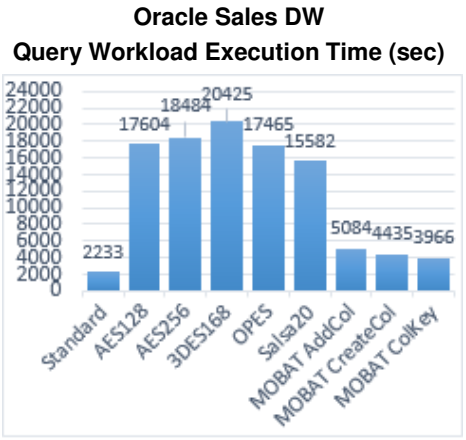


Figure 4-18a. Query Workload Execution Time per Solution in Oracle for the Sales DW

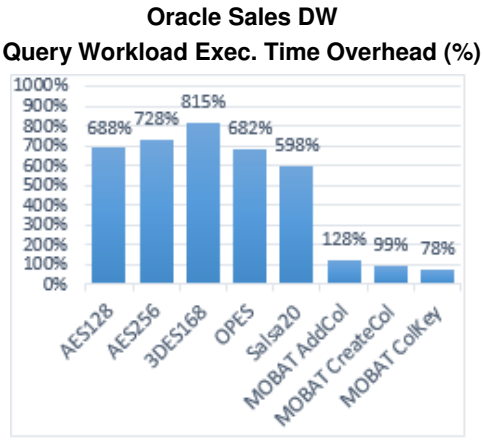


Figure 4-18b. Query Workload Execution Time Overhead (%) per Solution in Oracle for the Sales DW

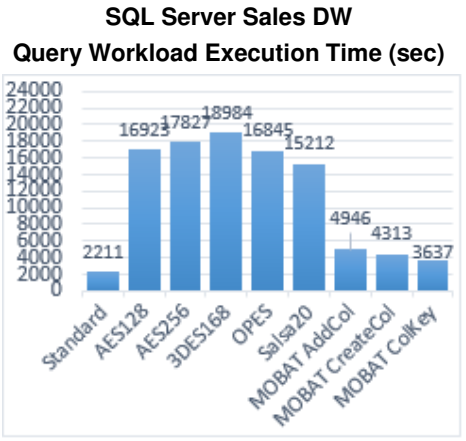


Figure 4-19a. Query Workload Execution Time per Solution in SQL Server for the Sales DW

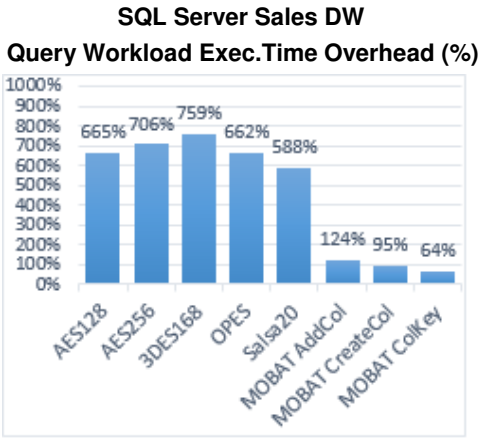


Figure 4-19b. Query Workload Exec. Time Overhead (%) per Solution in SQLServer for the Sales DW

All the remaining encryption solutions continue approximately leveled and present overheads between 598% and 815% in Oracle, corresponding to adding an extra 13349 to 18192 seconds to total execution time, and overheads between 588% and 759% in SQL Server, corresponding to adding 13001 to 16773 seconds to total execution time. Regarding these solutions, Salsa20 continues to be the fastest with OPES and AES128

coming afterwards, followed by AES256 and 3DES168 as the slowest solution. This means that MOBAT produces overheads that are roughly one sixth to one eighth of the encryption solutions, on average, in the chosen experimental setups.

Tables 4-9, 4-10 and 4-11 summarize the query workload execution time results respectively for the TPC-H 1GB, TPC-H 10GB and Sales DW, for each DBMS, highlighting the best solutions in each case.

Table 4-9. TPC-H 1GB Query Workload Execution Time Overhead

	Oracle TPC-H 1GB Execution Time (Overhead)	SQL Server TPC-H 1GB Execution Time (Overhead)
Standard Loading Time	00:10:25	00:09:40
AES128	00:29:58 (00:19:33 / 188%)	00:26:31 (00:16:51 / 174%)
AES256	00:30:37 (00:20:12 / 194%)	00:27:26 (00:17:46 / 184%)
3DES168	00:31:35 (00:21:10 / 203%)	00:28:32 (00:18:52 / 195%)
OPES	00:30:13 (00:19:48 / 190%)	00:27:09 (00:17:29 / 181%)
Salsa20	00:28:47 (00:18:22 / 176%)	00:25:23 (00:15:43 / 163%)
MOBAT AddCol	00:14:06 (00:03:41 / 35%)	00:13:33 (00:03:53 / 40%)
MOBAT CreateCol	00:13:29 (00:03:04 / 29%)	00:12:55 (00:03:15 / 34%)
MOBAT ColKey	00:12:43 (00:02:18 / 22%)	00:11:52 (00:02:12 / 23%)

Table 4-10. TPC-H 10GB Query Workload Execution Time Overhead

	Oracle TPC-H 10GB Execution Time (Overhead)	SQL Server TPC-H 10GB Execution Time (Overhead)
Standard Loading Time	01:42:35	01:28:21
AES128	04:42:07 (02:59:32 / 175%)	03:42:24 (02:14:03 / 152%)
AES256	04:48:03 (03:05:28 / 181%)	03:50:46 (02:22:25 / 161%)
3DES168	04:59:33 (03:16:58 / 192%)	04:10:58 (02:42:37 / 184%)
OPES	04:41:29 (02:58:54 / 174%)	03:40:15 (02:11:54 / 149%)
Salsa20	04:21:44 (02:39:09 / 155%)	03:31:31 (02:03:10 / 139%)
MOBAT AddCol	02:05:27 (00:22:52 / 22%)	01:47:00 (00:18:39 / 21%)
MOBAT CreateCol	02:01:54 (00:19:19 / 19%)	01:42:42 (00:14:21 / 16%)
MOBAT ColKey	02:00:18 (00:17:43 / 17%)	01:39:41 (00:11:20 / 13%)

Table 4-11. Sales DW 2GB Query Workload Execution Time Overhead

	Oracle Sales DW 2GB Execution Time (Overhead)	SQL Server Sales DW 2GB Execution Time (Overhead)
Standard Loading Time	00:37:13	00:36:51
AES128	04:53:24 (04:16:11 / 688%)	04:42:03 (04:05:12 / 665%)
AES256	05:08:04 (04:30:51 / 728%)	04:57:07 (04:20:16 / 706%)
3DES168	05:40:25 (05:03:12 / 815%)	05:16:24 (04:39:33 / 759%)
OPES	04:51:05 (04:13:52 / 682%)	04:40:45 (04:03:54 / 662%)
Salsa20	04:19:42 (03:42:29 / 598%)	04:13:32 (03:36:41 / 588%)
MOBAT AddCol	01:24:44 (00:47:31 / 128%)	01:22:26 (00:45:35 / 124%)
MOBAT CreateCol	01:13:55 (00:36:42 / 99%)	01:11:53 (00:35:02 / 95%)
MOBAT ColKey	01:06:06 (00:28:53 / 78%)	01:00:37 (00:23:46 / 64%)

To demonstrate the effects of using masking and encryption on each individual query, Figure 4-20 shows the results for individual query execution time in Oracle for the TPC-H 10GB scenarios, with a logarithmic scale. These results show that all queries have similar proportional overhead to those of the complete workload. This is also true for all the other scenarios, making it redundant to include all of them. Query Q1 presents the most significant results because it processes more than 90% of the fact table data, while the other process less than 10%. It can be seen that mostly all queries processed using the encryption solutions have introduced overheads of several orders of magnitude higher than MOBAT, individually matching what has been shown in the total query workload results through Figures 4-14 to 4-19.

The individual query execution times for the Sales DW are not included, given that this set of queries can produce a certain amount of insight as a whole (and shown in the total query workload execution graphs in Figures 4-18a to 4-19b), but should mainly not be considered as appropriate for individual analysis, since this DW is a specific real-world database and it is not a standard nor a benchmark.

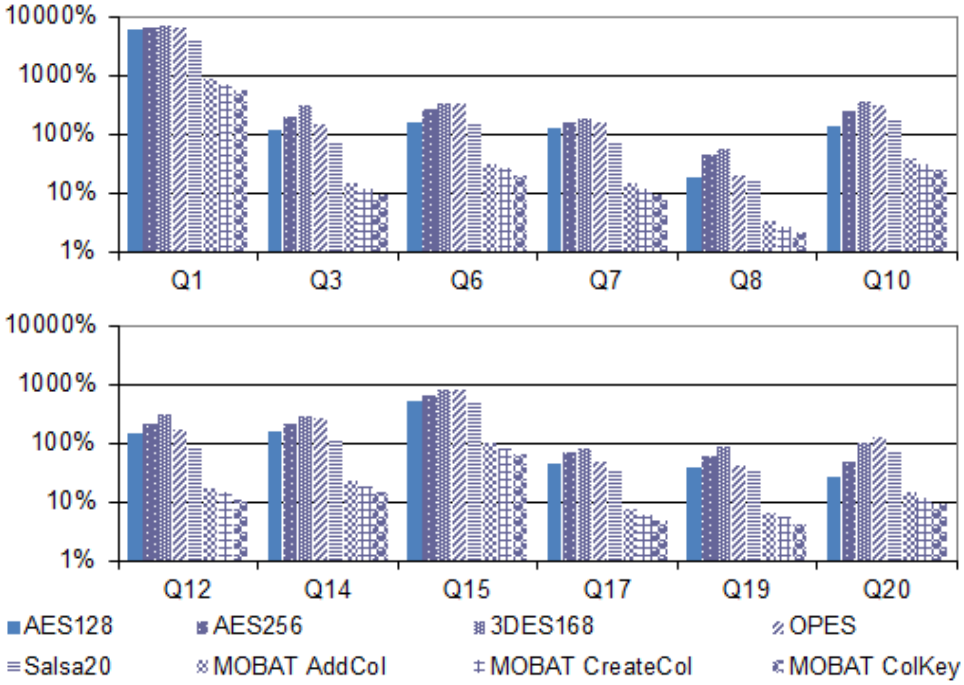


Figure 4-20. TPC-H 10GB Individual Query Execution Time Overhead per Query per Solution in Oracle 11g

4.5 Discussion on MOBAT

Contrarily to typical commercial data masking tools which provide data masking routines that, once applied, do not allow reversing the operations to retrieve the original data, the technique proposed in this chapter manages full masking and unmasking processes. MOBAT accomplishes continuous data protection similarly to commercial masking tools, since it maintains data-at-rest masked at all times, and adds the advantage of enabling its usage in live databases.

Basing the masking and unmasking processes simply on SQL rewriting enables executing direct queries against masked/unmasked data without having that data transferred between the database and the masking/unmasking mechanisms, thus avoiding the I/O and network bandwidth congestion that other solutions introduce due to those data roundtrips.

High-level SQL rewriting also makes MOBAT a straightforward portable technique to be universally used in any DBMS regardless of the CPU and operating system, contrarily to what occurs with most standard encryption packages supplied by DBMS. Most of these packages are CPU optimized, *i.e.*, designed and programmed for specific processor models and therefore depending on those CPUs, meaning that they may fail to execute on other machines. MOBAT is completely processor-independent, since all CPUs support basic modular and arithmetic operations.

As we discussed before, while DW data is mainly composed by numerical values, standard encryption algorithms are designed to output generic textual values. In the encryption packages supplied by commercial DBMS, the output they produce is textual or varbinary type values (char, varchar, varbinary, etc). Given that most sensitive columns in DW fact tables store numerical values, using these packages to encrypt data requires converting those values 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. MOBAT is specifically designed for masking numerical values, and in this sense, it is much more performance efficient for protecting DW facts. The data loading and query execution response time results shown in the experimental evaluations demonstrate this, and show that using encryption does in fact introduce extremely high storage space, loading time and query response time overhead.

In what concerns storage space, OPES and MOBAT introduce much smaller storage space overheads than the remaining solutions (less than 25% of extra storage space), followed by Salsa20 at a considerable difference (adding approximately 30% of storage space in TPC-H and almost 100% in the Sales DW), while the standard encryption solutions produce the highest storage space overheads by far. The standard encryption solutions introduce the highest overheads, roughly requiring duplicating the original database storage space for the TPC-H scenarios tested and between 308% and 591% of extra storage space in the Sales DW scenarios.

Note that in the best case scenarios for the standard encryption algorithms in TPC-H 10GB, an overhead of 104% in Oracle implies using more 8GB of storage space, and for an overhead of 308% in the Sales DW implies using more 5GB of storage space. OPES only requires a storage space overhead of 2% for TPC-H, which means that the worst case scenarios would imply using more 208MB of storage space in TPC-H 10GB and 62MB in the Sales DW. MOBAT would require almost 1GB of extra storage space for the TPC-H 10GB worst case scenario, and 73MB of extra storage space for the Sales DW. Salsa20 requires approximately three times more storage space overhead than MOBAT, and ten to twenty times more than OPES. These results show that Salsa20 and the standard encryption solutions effectively introduce a much higher increase of extra storage space than OPES and MOBAT.

In what concerns loading time, MOBAT is much faster than all the remaining solutions, introducing 3% to 16% of extra loading time in the tested scenarios. OPES has the second best performance, introducing 46% to 71% of extra loading time, more than four times worse than MOBAT on average. Salsa20 presents loading time overheads from 72% to 114%, on average roughly nine or ten times worse than MOBAT, while the standard encryption solutions introduce overheads of more than 100%, reaching more than 200% in several scenarios. In practice, while MOBAT introduces an extra 6 minutes of loading time in the worst case scenario, the standard algorithms introduce at least almost one hour of extra loading time.

Considering the results obtained in the query workload executions, MOBAT is also much faster than the remaining solutions. By observing the results, it can be seen that the relative differences between the solutions are approximately proportional throughout the different scenarios, with MOBAT always as the fastest solution and therefore introducing the smallest execution time overheads by several orders of magnitude, roughly one sixth, on average, of the remaining solutions. In practice, MOBAT adds less than 12 minutes of extra execution time in all TPC-H 10GB and Sales DW scenarios, and the remaining solutions introduce at least 30 more minutes up to more than 2 hours.

All the results in all scenarios and databases for both DBMS also show that the performance of *CreateCol Masking* is better than *AddCol Masking*, which was expected. The performance results of *ColKey Masking* are the best,

given the absence of changes in the original fact table data structure and size.

Given that decision support environments typically execute long running queries (*i.e.*, queries that can run for many minutes up to hours), the response time overheads introduced due to the use of encryption solutions represent high absolute values that can easily make query responses overdue and jeopardize the usefulness of the DW itself. Considering the magnitude of the results shown in the experimental evaluations, even a minimum gain in response/CPU time can be considered as an important achievement.

Although not nearly as secure as standard or state-of-the-art encryption algorithms, the proposed data masking technique is able to provide at least acceptable security while requiring a small amount of computational resources, introducing small response time and storage space overhead. Moreover, it keeps the data-at-rest always masked. Assuming that implementing a minimum amount of security strength concerning data confidentiality is better than not implementing any security at all, this makes the proposed masking technique a feasible and valid alternative for data warehousing contexts in which minimizing response time is so critical that using encryption to protect the DW is not acceptable.

Given that the proposed masking technique is straightforward and nearly effortless to implement, the masking keys may be periodically refreshed by rebuilding the masked table values, frequently switching the values of all or any one of the K_1, K_2, and K_3 keys before refreshing masked data in order to ensure that data is properly protected. Although it is not possible to absolutely prove that a particular algorithm is absolutely secure [Elminaam *et al.*, 2010; Ge and Zdonik, 2007; Kim *et al.*, 2010; Mattson, 2004; Nadeem and Javed, 2005; Natan, 2005], we believe that our technique is secure enough to be acceptable for use and that the small overheads introduced in both data loading and query execution performance are also acceptable, allowing us to state that it may be considered as a valid alternative for enhancing data confidentiality in DWs.

4.6 Summary

In this chapter we proposed a data masking solution specifically designed for enhancing data confidentiality in DWs. 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 easily implemented in any DBMS. The proposed solution is transparently used and to query the database the user interfaces only need to send their queries to a middleware instead of to the DBMS. Data at rest is always masked and only the processed results are returned to the authorized user interfaces that requested them. All SQL commands and actions are encrypted and stored in a log by the middleware security broker, which can be audited by security staff. If an attacker bypasses the broker and gains direct access to stored data, s/he just views masked "realistic-looking" but not real values.

Since it basically works by transparently rewriting user queries, the approach minimizes the required changes to user applications, and does not jeopardize network bandwidth. The masked database can be directly used for production purposes, while applications under development may directly query the database without passing through the MOBAT application (*e.g.* for software testing purposes), therefore retrieving realistic data, but never the real data. 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, we have compared it with the AES and 3DES encryption algorithms provided by leading commercial DBMS, as well as two state-of-the-art encryption proposals. The experimental results show that the storage space increase and the degradation of database performance in response time introduced by these standard and research solutions is very significant from the DW perspective. This enforces stating that those techniques are in fact too complex to be used in DW scenarios.

Given that most DW data consists on numerical values, our masking technique is tailored for this kind of data, thus showing better database performance than the remaining encryption solutions, while managing to maintain a significant level of security strength. Thus, it is an efficient overall solution and a valid alternative for balancing performance and security issues from the DW perspective. In the next chapter, we propose an encryption solution based on the masking solution that enhances its security while maintaining low performance overhead.

Chapter 5

SES-DW: A Specific Encryption Solution for Data Warehouses

As we discussed in Chapter 2 and demonstrated in Chapter 4, database storage size and response time overheads introduced by using encryption in very large databases such as DWs may jeopardize their feasibility. However, given the value of DW data, it is not advisable to avoid using encryption to secure that data just because of those overheads. This arises the need for encryption solutions that are capable of maintaining database performance as high as possible while providing significant security strength. Although the data masking solution proposed in the previous chapter provides some security strength, it is far from being a full-poof solution. Therefore, in this chapter we propose an encryption algorithm that computes a series of data transformations based on the data masking solution proposed in the previous chapter, which improves its security strength while maintaining low performance overhead.

The proposed Specific Encryption Solution tailored for Data Warehouses (SES-DW) consists on a lightweight encryption cipher for numerical values, which uses only mixes of standard SQL operators such as eXclusive OR (XOR) and modulus (MOD, that return the remainder of a division expression), together with additions and subtractions, similarly to the data masking solution proposed in the previous chapter. Storage space overhead is also avoided by preserving each encrypted column's datatype, while using only standard SQL operators enables the transparent use of SQL rewriting in order to avoid I/O and network bandwidth bottlenecks by discarding data roundtrips between the database and the encryption and decryption mechanisms (similarly to the masking solution presented in Chapter 4).

Also similarly to what we mentioned in the previous chapter it is important to note that it is not our aim to propose an encryption solution as strong in

security as any state-of-the-art encryption algorithm, but rather a technique that provides a considerable level of overall security strength while introducing small performance overhead, *i.e.*, that presents better security-performance balancing. Nevertheless, we include a thorough security analysis of the proposed cipher. As the data masking technique proposed in the previous chapter, this encryption technique fits into the middleware layer of the security framework described in Chapter 3, working transparently between user interfaces and the DBMS.

Experiments are included in order to compare the proposed solution with the standard encryption algorithms available in current DBMS, namely AES and 3DES, and also with state-of-the-art proposals such as Order-Preserving Encryption (OPES) and Salsa20 (alias Snuffle), using the TPC-H decision support benchmark and a real-world DW running on top of the Oracle 11g and Microsoft SQL Server 2008 DBMS.

The remainder of this chapter is organized as follows. Section 5.1 presents the encryption cipher and Section 5.2 describes its functional architecture. Section 5.3 presents a security analysis on the proposed cipher. Section 5.4 presents the experimental evaluation. Section 5.5 includes a discussion on the proposed encryption solution and on the results obtained in the experiments. Section 5.6 presents our conclusions.

5.1 SES-DW Encryption Cipher

Given x as the plaintext value to cipher and y as the encrypted ciphertext, the external view for encrypting x using the SES-DW cipher is shown in Figure 5-1, and considers the following assumptions:

- *NR* is the number of rounds executed by the cipher;
- *RowK* is a 2^{128} bit random encryption key (in a database table *T*, each row *j* has its own *RowK*, meaning each encrypted table *T* has a vector *RowK[j]* where *j* = [1…number of rows in *T*);
- *Operation*[*t*] is a random binary vector with *NR* elements (*i.e.*, each element is randomly 1 or 0), where *t* represents each encryption round's number (*i.e.*, *t* = 1...*NR*);
- *XorK*[*t*] and *ModK*[*t*] are vectors where each element is a random value encryption subkey with the same bit length as the plaintext *x*, (where *t* = 1...*NR*);

- $F(t)$ is a MOD/XOR mix function (explained further), where t represents each encryption round's number (*i.e.*, $t = 1...NR$).

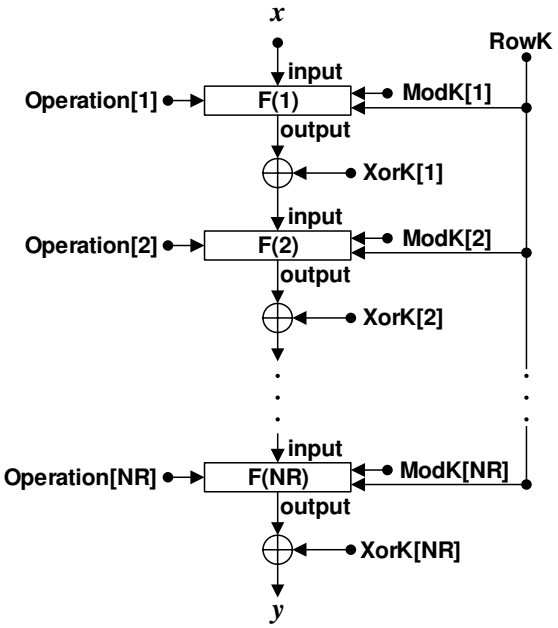


Figure 5-1. The SES-DW Data cipher for encryption

The MOD/XOR mix function $F(t)$ for encryption, considering *input* as the function's input and *output* as its output, is defined as:

```
IF Operation[t] = 1 THEN
   output = input+(RowK MOD ModK[t])-ModK[t]
ELSE
   output = input
END_IF
```

Given this, the SES-DW cipher encryption function for encrypting x by executing NR rounds is as shown:

```
FUNCTION Encrypt(x, NR)
   EncrOutput = x
   FOR t = 1 TO NR
      IF Operation[t] = 1 THEN
         EncrOutput = EncrOutput+(RowK MOD ModK[t])-ModK[t]
      END_IF
      EncrOutput=EncrOutput XOR XorK[t]
   END_FOR
RETURN EncrOutput
```

As illustrated, SES-DW randomly mixes MOD with XOR throughout the encryption rounds, given a random distribution of 1 and 0 values of the vector *Operation*. In the rounds where *Operation*[*t*] = 0, only XOR is used with the respective *XorK*[*t*]; in rounds where *Operation*[*t*] = 1, SES-DW first performs MOD with addition and subtraction using the respective *ModK*[*t*] and *RowK*[*j*], and afterwards XOR with the respective *XorK*[*t*]. To avoid generating a ciphertext that may overflow the bit length of *x* it must be assured that the bit length of the term using MOD (*EncrOutput* + (*RowK*[*j*] MOD *ModK*[*t*]) - *ModK*[*t*]) is smaller or equal to the bit length of *x*.

As a practical example of encrypting with SES-DW, consider the encryption of an 8 bit numerical value (*x* = 126) executing 4 rounds (*NR*= 4), for a row that has *RowK* = 15467801, given the following assumptions for *Operation*, *XorK* and *ModK*:

```
Operation = [0, 1, 0, 1]
XorK = [31, 2, 28, 112]
ModK = [87, 36, 123, 19]
```

Then for *t* = 1 (round 1), `EncrOutput = 126 XOR 31 = 97`

For *t* = 2 (round 2), `EncrOutput = (97+(15467801 MOD 36)-36) XOR 2 = 64`

For t=3 (round 3), `EncrOutput = 64 XOR 28 = 92`

For t=4 (round 4), `EncrOutput = (92+(15467801 MOD 19)-19) XOR 112 = 40`

Thus, *Encrypt*(126, 4) = 40. To decrypt, SES-DW inverts the cipher. Figure 5-2 shows the external view of the SES-DW decryption cipher steps, in which $F^{-1}(x)$ also represents the reverse MOD/XOR mix function for decryption. Given this, the SES-DW cipher decryption function for decrypting *y* with *NR* rounds is:

```
FUNCTION Decrypt(y, NR)
   DecrOutput = y
   FOR t = NR DOWNTO 1
      DecrOutput = DecrOutput XOR XorK[t]
      IF Operation[t] = 1 THEN
         DecrOutput = DecrOutput-(RowK MOD ModK[t])+ModK[t]
      END_IF
   END_FOR
RETURN DecrOutput
```

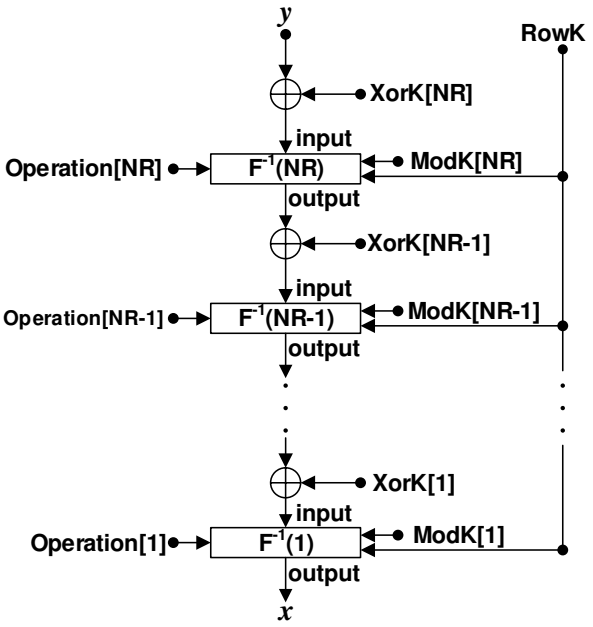


Figure 5-2. The SES-DW Data cipher for decryption

Considering the encryption example previously shown, we now demonstrate the decryption process for y = 40, given the same *Operation*, *RowK*, *XorK* and *ModK*:

For t=4 (round 1), `DecrOutput=(40 XOR 112)-(15467801 MOD 19)+19 =92`

For t=3 (round 2), `DecrOutput=92 XOR 28=64`

For t=2 (round 3), `DecrOutput=(64 XOR 2)-(15467801 MOD 36)+36 =97`

For t=1 (round 4), `DecrOutput=97 XOR 31=126`

Thus, *Decrypt*(40, 4) = 126, which is the original x plaintext value. Although the SES-DW cipher aims to work only with numerical values, we maintain the designation of *plaintext* and *ciphertext* respectively for the true original input value and ciphered output value.

5.2 Functional Architecture

The functional architecture for using SES-DW in practice is shown in Figure 5-3, which is similar to what was presented for MOBAT in the previous chapter. The architecture is made up by three entities:

- The encrypted database and its DBMS;
- The SES-DW security middleware interface;
- User/client interfaces to query the encrypted database.

The *SES-DW middleware interface* acts as a broker between the DBMS and the user interfaces, using the SES-DW encryption and decryption methods and ensuring the queried data is securely processed and the proper results are returned to those interfaces. All communications are executed through SSL/TLS secure connections, to protect SQL instructions and returned results between the system's entities.

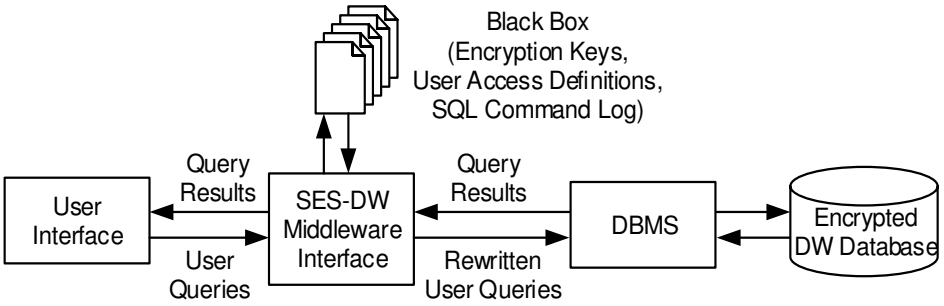


Figure 5-3. The SES-DW Data Security Functional Architecture

The *Black Box* is stored in the *Security Framework Database* database server, and there is one *Black Box* created for each encrypted DW database. Only the SES-DW middleware itself can access the *Black Box*, where all encryption keys and predefined data access policies for the database are stored.

As in MOBAT, the SES-DW middleware also creates a history command log that can also be used for intrusion detection purposes, when integrated with the DIDS proposed in the following chapter. All *Black Box* contents are encrypted using AES with a 256 bit key, and there is no way to restore its true data, except by cracking the encryption keys. These keys are generated by the SES-DW middleware and are never shown or known by the DBA or any other user.

To obtain true results, all user queries or actions must pass through the SES-DW middleware, which will store a copy of those instructions in the *Black Box* command history log. Each time a user requests any action, the

middleware will receive and parse the instructions, fetch the encryption keys, rewrite the command, send it to be processed by the DBMS and retrieve the results, and finally send those results back to the user interface that issued the request. Thus, SES-DW is transparently used, since SQL command rewriting is transparently managed by the middleware. Obviously, user applications should send the commands to the middleware, instead of querying the DBMS directly.

To encrypt a database, a DBA requires it through the SES-DW middleware. After entering login and database connection information, the middleware connects to the database and creates the corresponding *Black Box*, as explained earlier. Afterwards, the middleware allows the DBA to define which tables and columns to encrypt. All the required encryption keys (*RowK*, *XorK*, *ModK*) for each table and column are generated, encrypted by an AES256 algorithm and stored in the Black Box. Finally, the middleware encrypts all values in each column that were marked for encryption. Subsequent updates on the database data must always be done through the middleware, which will apply the cipher to the values and store them directly in the database.

In order to implement SES-DW encryption in a given table T, consider the following: suppose table T with a set of N numerical columns $C_i = \{C_1, C_2, \ldots, C_N\}$ to be encrypted and a total set of M rows $R_j = \{R_1, R_2, \ldots, R_M\}$. Each value to encrypt in the table is identified as a pair (R_j, C_i), where Rj and Ci respectively represent the row and column to which the value refers ($j = \{1..M\}$ and $i = \{1..N\}$). To use the SES-DW cipher, we generate the following encryption keys and requirements:

- An encryption key *TabK*, a 128 bit random generated value, constant for table T;
- Vector *RowK*[j], with $j = \{1..M\}$, for each row j in table T. Each element holds a random 128 bit value;
- Define NR_i with $i = \{1..N\}$, which gives the number of encryption rounds to execute for each column C_i. We define $NR_i = SBL_i/\text{BitLength}(C_i)$, where SBL_i is the desired security bit strength for the *XorK* and *ModK* encryption keys of column C_i and BitLength(C_i) is the datatype bit length of column C_i (*e.g.* if we want

to secure a 16 bit column C_i with a security strength of 256 bits, then the number of encryption rounds would be 256/16 = 16);

- Vectors $XorK_i[t]$ and $ModK_i[t]$, with t = {1..NR_i}, for each C_i, filled with randomly generated unique values. The bit length of each key is equal to the bit length of each C_i's datatype;
- A vector $Operation_i[t]$, with t = {1..NR_i}, for each column C_i, filled randomly with 1 and 0 values, so that the count of elements equal to 1 is the same as the count of elements equal to 0 (e.g. $Operation_i$ = [0, 1, 0, 0, 1, 1, 0, 1], with NR_i = 8). This makes Prob(Operation[t]=0) ⇔ Prob(Operation[t]=1), *i.e.*, the probability of executing or not MOD operations in each cipher round is uniformly distributed, in order to avoid information leakage in attempting to break security.

Since the number of rows in a DW fact table is often very big, the need to store a $RowK[j]$ encryption key for each row j poses a challenge. If these values were stored in a lookup table separate from table T, a heavy join operation between those tables would be required to decrypt data. Given the typically huge number of rows in fact tables, this must be avoided. For the same reasons, storing $RowK[j]$ in RAM is also impracticable. To avoid table joins, as well as oversized memory consumption, the values of $RowK[j]$ must be stored along with each row j in table T, as an extra column C_{N+1}. This is the only change needed in the DW data structure in order to use SES-DW. To secure the value of $RowK[j]$, it should be XORed with key $TabK$ before being stored. To retrieve the true value of $RowK[j]$ in order to use the SES-DW algorithms, we need to simply calculate (R_j, C_{N+1}) XOR $TabK$.

5.3 Security Issues

Most security issues and assumptions concerning the threat model, datatype preservation, having data-at-rest masked or encrypted at all times, and the use of MOD and XOR operations for SES-DW are similar to those described in the previous chapter for MOBAT. In this section we present the security proof specifically concerning the SES-DW algorithm, as well as the entropy produced by SES-DW.

5.3.1 Using Variable Key Lengths and MOD-XOR Mixes

The bit length of the encryption subkeys *XorK* and *ModK* are the same as the bit length of each encrypted column, meaning that an 8 bit sized column datatype will have 8 bit sized encryption subkeys. It is obvious that using 8 bit subkeys on their own is not secure at all. However, since all keys are distinct in each round, executing 16 rounds would be roughly equivalent to having a 16*8 = 128 bit key in the encryption process. As discussed in [Elminaam *et al.*, 2010; Kim *et al.*, 2010; Mattson, 2004; Nadeem and 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. It is up to the DW security administrator to decide how strongly secure each column should be, which defines how many rounds should be executed, considering the bit length of the column's datatype.

As previously mentioned in Chapter 3, the MOD operator is used in the cipher because it is non-injective and consequently makes our cipher not directly invertible. It is also true that the same ciphered output values are most likely to come from different original input values and have approximately the same probability for any output value within the full range of possible output values. This is formally demonstrated in Subsection 5.3.3, where the cipher's entropy is computed, showing a nearly uniform probability distribution.

Randomly using the XOR and MOD operators as the two possible operators for each round also increases the number of possibilities an attacker needs to test in exhaustive searches for the output values of each encryption round, since the attacker does not know the rounds in which MOD is used with XOR and needs to test both hypothesis (XOR and MOD-XOR). Furthermore, if the attacker does not know the security strength chosen for encrypting each column, s/he does not know how many encryption rounds were executed for each ciphered value.

By making the values of $XorK_i$ and $ModK_i$ distinct between columns, we also make encrypted values independent from each other between columns. Even if the attacker breaks the security of one column in one table row, the information obtained from discovering the remaining encryption keys is limited. Thus, the attacker cannot infer information enough to break overall security; in order to succeed, s/he must perform recover all the keys for all columns.

5.3.2 Attack Costs on SES-DW

To break security by key search in a given column C_i, the attacker needs to have at least one pair (plaintext, ciphertext) for a row j of C_i, as well as the security bit strength involved, as explained in Section 5.2, because it will indicate the number of rounds that were executed. In this case, taking that known plaintext, its respective ciphertext, and the C_{N+1} value (storing $RowK_j$ XOR *TabK*, as explained in Section 5.2), s/he may then execute an exhaustive key search.

The number of cipher rounds for a column C_i is given by *NRi*, and β is the bit-length of C_i's datatype. Since half the values of vector *Operation* are zeros and the other half are ones, the probability of occurrences of 1 and 0 is equal, *i.e.*, *Prob*(*Operation*[*t*]=0) = ½ = *Prob*(*Operation*[*t*]=1), where the number of possible values for *Operation*[*t*] is 2^{NRi}.

Considering β, each *XorK* and *ModK* subkey also has a length of β bits and thus, each *XorK* and *ModK* subkeys have a search space with 2^{β} possible values. *TabK* is a 128 bit value, thus with a search space of 2^{128} possible values. Considering the cipher's algorithm and given the probability of {0, 1} values in *Operation*, a XOR is executed in all rounds (*NRi*), while a MOD is executed before the XOR in half the rounds (*NRi*/2). Given this, the key search space dimension considering the combination of XOR and MOD/XOR rounds is given by $G(x)$:

$$G(x) = \sum_{x=1}^{NRi+\frac{NRi}{2}} F(x) \,.\, 2^{(\beta x)+128}$$

$$F(x) = \begin{cases} \binom{NRi-x}{\frac{NRi}{2}-x} & , x = 1 \\ F(x-1) + (-1)^x \binom{NRi-x}{\frac{NRi}{2}-x} & , 2 <= x <= NRi/2 \\ F(x-1) & , NRi/2+1 <= x <= NRi \\ F(x-1) + (-1)^{(x-\frac{NRi}{2})} \binom{x-\frac{NRi}{2}-1}{x-NRi-1} & , NRi+1 <= x <= NRi + NRi/2 - 1 \\ \binom{NRi}{\frac{NRi}{2}} & , x = NRi + NRi/2 \end{cases}$$

Considering Y as the number of attempts to discover the keys, Y is a discrete random variable with support $S = \{1 \ldots N\}$, where N represents the search space's dimension. For one attempt, considering a random variable B, it has only two possibilities:

$$B = \begin{cases} 0, & given\ the\ attempt\ is\ not\ successful \\ 1, & given\ the\ attempt\ is\ successful \end{cases}$$

Therefore, B follows a Bernoulli distribution with probability $p = Prob(B{=}1) = 1/N$. Since the number of attempts is limited, given the search space is finite, variable Y also has a finite support $S = \{1 \ldots N\}$. The probability of being successful after k attempts is given by:

$$Prob(Y = k) = Prob(\bar{A} \cap \bar{A} \cap \ldots \cap \bar{A} \cap A) = \left(1 - \frac{1}{N}\right)^{k-1} . \frac{1}{N}\ , k{=}1 \ldots N$$

Note that the probability of needing more than m attempts is given by:

$$Prob(Y > m) = \sum_{k=m+1}^{N} Prob(Y = k) = \sum_{k=m+1}^{N} \left(1 - \frac{1}{N}\right)^{k-1} . \frac{1}{N} = (1 - 1/N)^m . \left[\left(1 - \left(1 - \frac{1}{N}\right)^{N-m}\right)\right]$$

The probability of needing n more attempts, given m initial unsuccessful attempts (for $m > 1$ and $n > 1$) is defined by $Prob(Y > m{+}n \mid Y > m) = Prob(Y > m{+}n) / Prob(Y > m)$, since the event $\{Y > m{+}n\}$ is contained in $\{Y > m\}$, which means that after having m unsuccessful attempts, being successful after n more attempts only depends on those n additional attempts and not on the initial m attempts, *i.e.*, it does not depend on the past. For the complete search space, the average number of attempts is then given by:

$$\sum_{k=1}^{N} k . Prob(Y = k) = \frac{1}{N} \sum_{k=1}^{N} k \left(1 - \frac{1}{N}\right)^{k-1} = (*)$$

From the series theory it is known that

$$\sum_{k=0}^{+\infty} x^k = \frac{1}{1-x}\ , \text{ if } |x| < 1$$

Which is the case in

$(*)$ for $\left(1 - \frac{1}{N}\right)$.

Thus,

$$\left(\sum_{k=1}^{+\infty} x^k\right)' = \left(\frac{1}{1-x}\right)' \Leftrightarrow \sum_{k=1}^{+\infty} k . x^{k-1} = \frac{1}{(1-x)^2}\ , \ |x| < 1$$

Thus, the average number of attempts for finding the keys is

$$(*) = \frac{1}{N} \cdot \frac{1}{\left(1 - \left(1 - \frac{1}{N}\right)\right)^2} = N$$

which is equal to the dimension of the key search space (N). Note however, that this is the worst case complexity. It is possible for the attacker to reduce the key search space by chosen plaintext attacks. Since the same *TabK* key is used for encrypting all *RowK*, as explained in section 5.2 (C_{N+1}(row j) = $RowK[j] \oplus TabK$), the information leakage is given by

$$y_1 \oplus y_2 = (x_1 \oplus TabK) \oplus (x_2 \oplus TabK) \Leftrightarrow$$
$$\Leftrightarrow y_1 \oplus y_2 = (x_1 \oplus x_2) \oplus (TabK \oplus TabK)$$
$$\Leftrightarrow y_1 \oplus y_2 = x_1 \oplus x_2$$

This implies that C_{N+1}(row j) $\oplus$ C_{N+1}(row j+1) = $RowK[j] \oplus RowK[j+1]$, reducing the possible search space for *RowK* to 2^{64} instead of 2^{128} in each row. If the attacker manages to use very low *RowK* values, which are most probably smaller than the value of the *ModK* encryption keys (*i.e.* $RowK < ModK[t]$), then the ($RowK$ MOD $ModK[t]$) – $ModK[t]$ operation in the cipher will be reduced to $RowK - ModK[t]$, thus further reducing complexity. In this case, for example, taking more than one (*plaintext*, *ciphertext*) pair y_1 = *Encrypt*(x_1,2) and y_2 = *Encrypt*(x_2,2) for 2 encryption rounds on the same row, where *Operation*=[0,1]:

$$y_1 \oplus y_2 = (x_1 \oplus XorK[1] + RowK - ModK[2]) \oplus (x_2 \oplus XorK[1] + RowK - ModK[2])$$

Considering that each x_i has a length of β bits, given the encryption key *RowK* has a reduced search space of 2^{64} (as previously mentioned) and each *XorK* and *ModK* have a search space of 2^{β}, the key search space in this example is given by $2^{2\beta+64}$. Since *XorK*[1] and *ModK*[2] are just half the keys for the 2 round SES-DW, to obtain the remaining *XorK*[2] and *ModK*[1] keys, the search space is incremented by $2^{2\beta}$.

As the number of *XorK* and *ModK* encryption keys is the same as the number of rounds, the generic expression for the reduced key search space in this type of attack is given by $G(x) = 2^{NRi*\beta+64} + 2^{NRi*\beta}$. Note that for an 8 bit value (β = 8) encrypted by 16 rounds (*NRi* = 16), using 16 *XorK* and *ModK* subkeys with 8 bits each (each total key length for *XorK* and *ModK* is 16*8

= 128 bits), the key search space complexity is $2^{192} + 2^{128} \cong 6.3x10^{57}$, which remains a considerable measure of security strength.

5.3.3 SES-DW Entropy

In information theory, entropy is a measure of randomness or uncertainty [Vaudenay, 2006]. In this context, the term usually refers to Shannon's entropy, which quantifies the randomness of a variable based upon the knowledge of the information contained in its message. The entropy of a discrete variable X with n bits in length is given by the following expression, where $Prob(x_i)$ is the probability of occurrence of each x_i within the probability distribution of all possible integer values [1…2^n]:

$$Entropy(X) = -\sum_{i=1}^{2^n}\big(Prob(X = x_i).log_2 Prob(X = x_i)\big)$$

Since numeric datatype storage sizes are typically 8, 16, 32, 64 or 128 bits, each of our cipher's input/output values (as well as the encryption keys) respectively have a number of 2^8, 2^{16}, 2^{32}, 2^{64}, or 2^{128} possible combinations. While it is computationally fast to obtain the probability distribution in the first case by combining all possible input and encryption key values (with all 8 bit values = [1...2^8]) using two cipher rounds (the minimum number of rounds), for the remaining (2^{16}, 2^{32}, 2^{64} and 2^{128}) the task gets exponentially time-expensive.

Therefore, after a series of statistical regression experiments using the calculated 8 bit probability distribution for SES-DW, we found that the logarithmic regression ($y = a + b.ln(x)$) generated the most adjusted statistical model for representing the cipher's probability distribution (with correlation $R^2 >= 0.98$ and a standard error of 0.001). Knowing that the accumulated probability for n bits must be equal to 1, using the logarithmic regression function we must ensure that:

$$\int_1^{2^n} a + b.ln(x)\, dx = 1$$

This expression leads to $Prob(x_i) = \hat{a} + \hat{b}.ln(x_i)$, which represents the estimated probability distribution function for n bits SES-DW, where:

$$\hat{a} = \frac{1 - n.b.2^n.ln(2)}{2^n - 1} + b \quad \wedge \quad \hat{b} = \frac{\bar{X} - \left(2^{n-1} + \frac{1}{2}\right)}{2^{2n-2} - \frac{1}{4} - n\,.\,2^{n-1}\,.\,\ln(2)}$$

Given $Prob(x)$, the entropy of SES-DW for n = 8, 16, 32, 64 and 128 bits is shown in Table 5-1. As can be seen, the entropy produced for n bits is nearly n, thus meaning the generated ciphertexts are very close to a uniformly random n bit value and therefore, have very little information leakage because very little can be inferred from the output generated by the cipher.

Table 5-1. Estimated SES-DW entropy values

Number of bits (n)	SES-DW Entropy
8	7.967144
16	15.972308
32	31.979863
64	63.986246
128	127.989741

5.4 Experimental Evaluation

In these experiments we used the TPC-H benchmark [TPC-H] implemented in its 1GB and 10GB scale sizes, and a real-world sales DW storing one year of commercial data taking up 2GB of storage space (as we previously mentioned, full description of the sales DW can be seen in Appendix A). We tested all scenarios using the Oracle 11g and Microsoft SQL Server 2008 DBMS with default settings, on a Pentium Core2Duo 3GHz CPU with a 1.5TB SATA hard disk and 2GB RAM (512MB of devoted to database memory cache), running Windows 2003 Server.

As in the experiments involving the data masking solution, the columns chosen for testing the masking solution were those referring to numerical datatype columns belonging to the fact tables. The TPC-H schema has one fact table (*LineItem*), and seven dimension tables. In TPC-H setups, four numerical columns of *LineItem* were encrypted (*L_Quantity*, *L_ExtendedPrice*, *L_Tax* and *L_Discount*). The Sales DW database schema has one fact table (*Sales*) and four dimension tables. In the Sales DW, five numerical columns were encrypted (*S_ShipToCost*, *S_Tax*, *S_Quantity*, *S_Profit*, and *S_SalesAmount*).

In these experiments, we compare the storage size overhead and query response time of SES-DW with the column-based AES (with 128 bit and 256 bit security) and 3DES168 algorithms available as built-in options of

each DBMS, and OPES [Agrawal *et al.*, 2004] and Salsa20/20 [Bernstein, 2005; Bernstein, 2008]. OPES and Salsa20 were implemented using C#. We use the column-based solutions for the same reasons as explained in the previous chapter in Section 4.4 (see Table 4-2).

All loading time and query response time results shown in this section were obtained from an average of six executions in each described setup/scenario, given the relatively small standard deviation values, as in Chapter 4. The complete set of results and respective statistical measures can be seen in Appendix B. Note that the experiments included in this chapter cannot be directly compared with those of the data masking chapter, since different CPUs were used.

5.4.1. Analyzing Storage Space

The storage space results measured in these experimental evaluations are exactly the same as those presented for the data masking experimental evaluation in Subsection 4.4.1, making it redundant and unnecessary to repeat the analysis here. This happens because the implementation of SES-DW requires exactly the same changes to the DW data schemas as MOBAT, and the remaining encryption algorithms that we tested against are also the same as in the previous experiments. Therefore, in this subsection we just remind the main storage space results and conclusions.

For TPC-H 1GB:

- OPES and SES-DW produce much smaller storage space overheads than the remaining solutions;
- OPES adds a 2% overhead for both DBMS, corresponding to 18MB of extra storage space in Oracle and 21MB in SQL Server;
- SES-DW adds 8% and 12% overhead respectively in Oracle and SQL Server, corresponding to an extra 96MB and 102MB of storage space;
- Salsa20 introduces 38% overhead in Oracle, corresponding to adding 292MB, and 26% in SQL Server, which adds 316MB of extra storage space;
- The standard encryption solutions produce the highest overhead, with AES being the worst by requiring 154% in Oracle and 95% in SQL Server of storage space overhead, corresponding to respectively

adding 1188MB and 1173MB and 154% in each DBMS, while 3DES168 produced a storage space overhead of 104% in Oracle and 76% in SQL Server, respectively corresponding to 800MB and 944MB of extra storage space.

In what concerns the TPC-H 10GB DW, the extra storage space added to the 10GB database by each encryption solution is approximately proportional to those of the 1GB database, which means ten times bigger. Thus, the analysis of the results for the 10GB sized TPC-H database is similar to that of the 1GB.

For the Sales DW:

- OPES and SES-DW continue to produce much smaller storage space overheads than the remaining solutions, similarly to the occurred with TPC-H;
- OPES shows a 4% overhead for both DBMS, corresponding to an extra 62MB of storage space in Oracle and 73MB in SQL Server
- SES-DW presents 25% and 33% overhead for SES-DW respectively in Oracle and SQL Server, corresponding to an extra 415MB and 636MB of storage space;
- Salsa20 also introduces more storage space overhead than OPES and SES-DW, namely 88% in Oracle, corresponding to adding 1464MB, and 94% in SQL Server, which adds 1818MB of extra storage space;
- The standard encryption solutions continue to produce the highest overhead, with AES also being the worst by requiring 462% in Oracle and 591% in SQL Server of storage space overhead, corresponding to respectively adding 7688MB and 11424MB of storage space, while 3DES168 produced a storage space overhead of 308% in Oracle and 390% in SQL Server, respectively corresponding to 5125MB and 7532MB of extra storage space.

5.4.2. Analyzing Loading Time

Figures 5-4a and 5-4b respectively show the results for the total loading time (in seconds) and percentage of time overhead for loading the TPC-H 1GB *LineItem* fact table in Oracle, while Figures 5-5a and 5-5b show the same results in SQL Server. It can be observed that the standard loading

time for the TPC-H LineItem fact table without using any sort of encryption solution is 253 seconds in Oracle and 171 seconds in SQL Server.

As shown, SES-DW produces much smaller loading time overheads than the remaining solutions with the same bit security. SES-DW with 128 bit security shows 10% and 12% overhead in Oracle and SQL Server, respectively corresponding to an extra 26 and 20 seconds in loading time. SES-DW with 256 bit security shows 16% and 18% in Oracle and SQL Server, respectively corresponding to adding 41 and 30 seconds, and when using 1024 bit security (at least four times higher than the remaining solutions) the loading time overhead was 78% in Oracle and 66% in SQL Server, respectively corresponding to an extra 198 and 113 seconds of loading time.

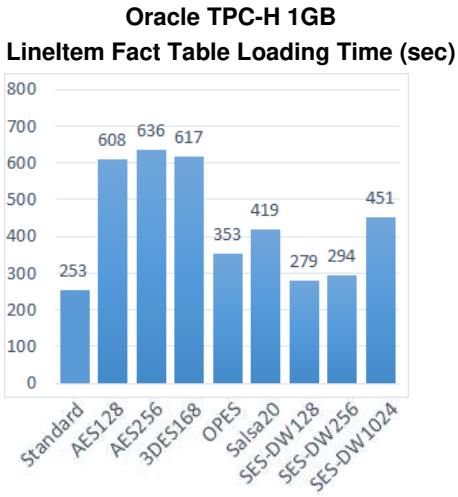


Figure 5-4a. Loading Time in Oracle for the TPC-H 1GB Fact Table per Encryption Solution

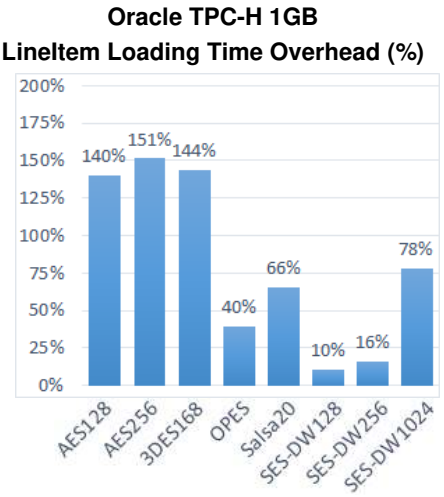


Figure 5-4b. Loading Time Overhead (%) in Oracle for the TPC-H 1GB Fact Table per Encryption Solution

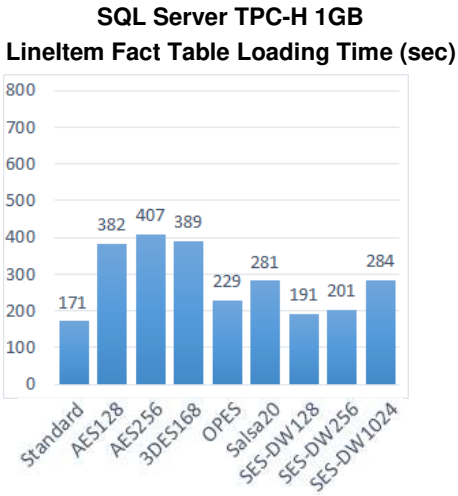


Figure 5-5a. Loading Time in SQL Server for the TPC-H 1GB Fact Table per Encryption Solution

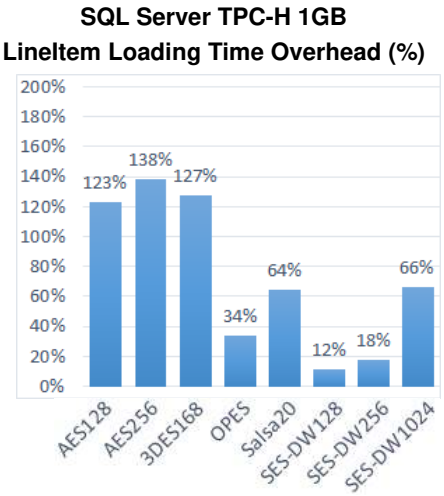


Figure 5-5b. Loading Time Overhead (%) in SQL Server for the TPC-H 1GB Fact Table per Encryption Solution

OPES comes after SES-DW 128 and 256 bit security in loading time performance, showing an overhead of 40% in Oracle and 34% in SQL Server, which respectively correspond to adding 100 and 110 seconds. Salsa20 introduces more loading time overhead than OPES and the referred bit security versions of SES-DW, namely 66% in Oracle, corresponding to adding 166 seconds, and 64% in SQL Server, which adds 110 seconds of extra loading time.

Note that, although SES-DW 1024 does introduce higher overhead than OPES and Salsa20, it does use a much higher security bit strength, which consequently has impact on its performance.

Similarly to what occurred with storage space, the standard encryption solutions produced the highest loading time overheads. AES with 128 bit security produced 140% in Oracle and 123% in SQL Server, respectively corresponding to adding 355 and 211 seconds to the standard loading time. AES with 256 bit security shows an overhead of 151% in Oracle and 138% in SQL Server, respectively corresponding to 383 and 236 seconds of extra loading time. 3DES168 introduces 144% loading time overhead in Oracle, corresponding to adding 364 seconds, and 127% in SQL Server, which adds 218 seconds of extra loading time.

Figures 5-6a and 5-6b respectively show the results of total loading time (in seconds) and percentage of loading time overhead for loading the TPC-H 10GB *LineItem* fact table in Oracle, while Figures 5-7a and 5-7b show the same results in SQL Server.

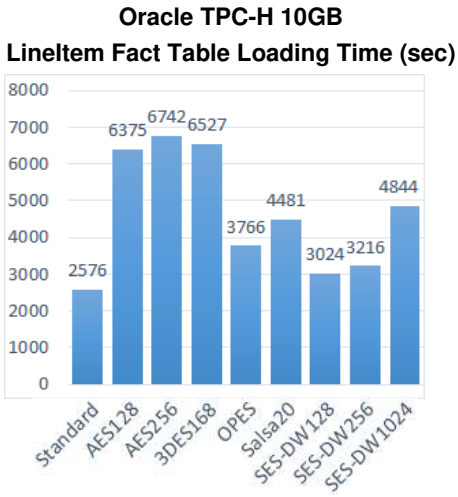


Figure 5-6a. Loading Time in Oracle for the TPC-H 10GB Fact Table

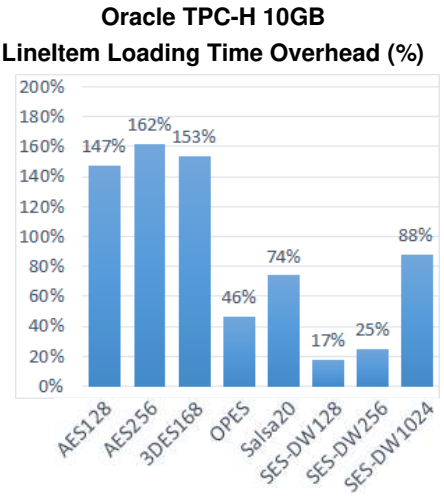


Figure 5-6b. Loading Time Overhead (%) in Oracle for the TPC-H 10GB

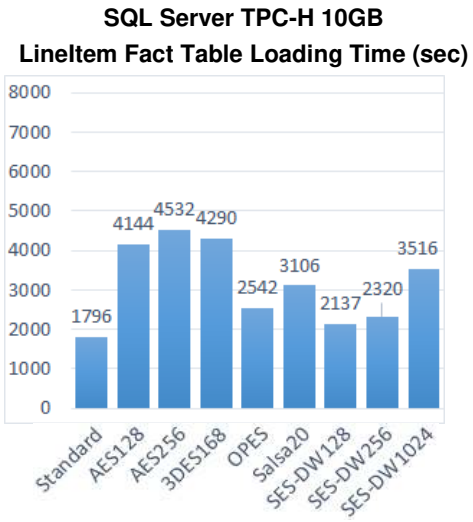


Figure 5-7a. Loading Time in SQL Server for the TPC-H 10GB

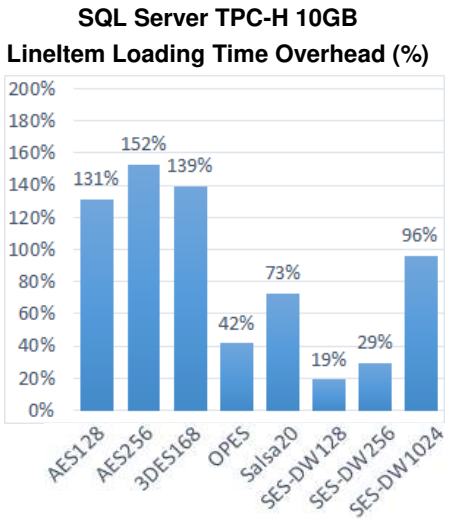


Figure 5-7b. Loading Time Overhead (%) in SQL Server for the TPC-H 10GB

From observing the results in Figures 5-6a to 5-7b, it can be seen that the extra loading time added to the 10GB database by each encryption solution is approximately proportional to those of the 1GB database, as occurred with the storage space, which means ten times bigger. Thus, the analysis of the results for the 10GB sized TPC-H database is also similar to that of the 1GB sized TPC-H database.

Figures 5-8a and 5-8b respectively show the results of total loading time (in seconds) and percentage of time overhead for loading the Sales DW fact table in Oracle, while Figures 5-9a and 5-9b show the same results in SQL Server. It can be seen that the standard loading time for the Sales fact table without using any encryption solution is 994 seconds in Oracle and 1013 seconds in SQL Server. As seen in both Figures, SES-DW continues to produce much smaller loading time overheads than the remaining solutions, similarly to the occurred with TPC-H. SES-DW with 128 bit security shows 13% and 15% overhead in Oracle and SQL Server, respectively corresponding to an extra 130 and 148 seconds in loading time. SES-DW with 256 bit security shows 22% in both DBMS, corresponding to adding 217 seconds in Oracle and 224 seconds in SQL Server, and when using 1024 bit security the loading time overhead was 82% in Oracle and 86% in SQL Server, corresponding to an extra 814 and 868 seconds of loading time.

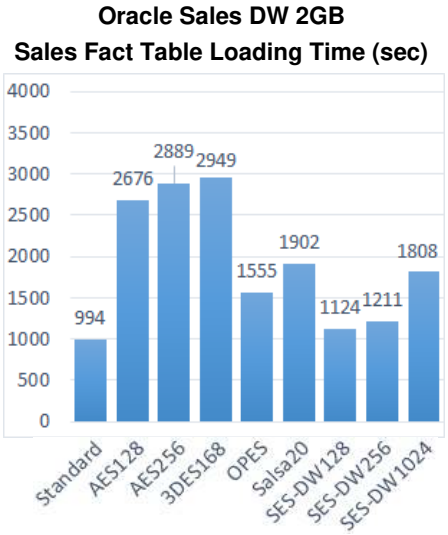


Figure 5-8a. Loading Time in Oracle for the Sales DW Fact Table

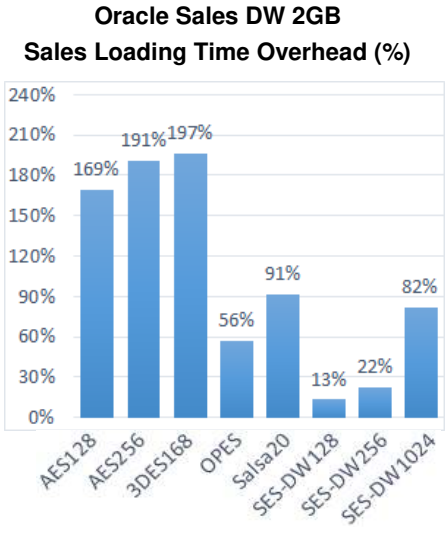


Figure 5-8b. Loading Time Overhead (%) in Oracle for the Sales DW

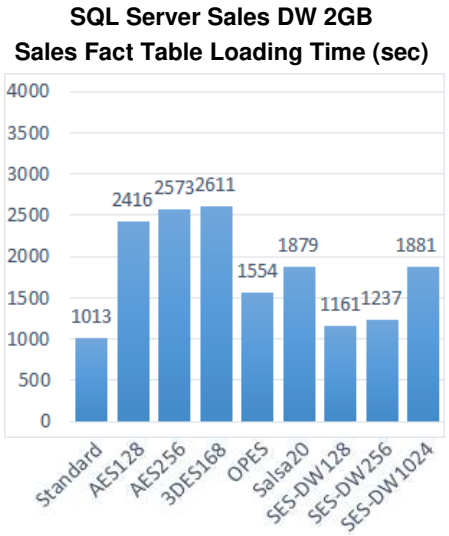


Figure 5-9a. Loading Time in SQL Server for the Sales DW Fact Table per Encryption Solution

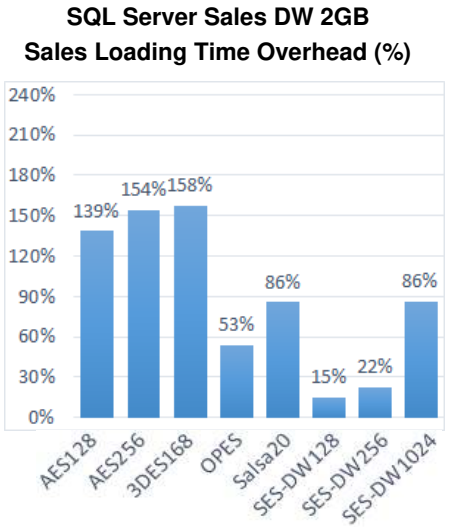


Figure 5-9b. Loading Time Overhead (%) in SQL Server for the Sales DW Fact Table per Encryption Solution

OPES comes after SES-DW in loading time performance, showing a 56% overhead in Oracle and 53% in SQL Server, respectively corresponding to an extra 561 seconds and 541 seconds of loading time, and Salsa20 presents 91% and 86% overhead for respectively in Oracle and SQL Server, corresponding to an extra 908 and 866 seconds of loading time in each DBMS.

The standard encryption solutions continue to produce the highest overhead, where AES with 128 bit security produced 169% in Oracle and 139% in SQL Server, respectively corresponding to adding 1682 and 1403 seconds to the standard loading time. AES with 256 bit security shows an overhead of 191% in Oracle and 154% in SQL Server, respectively corresponding to 1895 and 1560 seconds of extra loading time. 3DES168 introduces 197% loading time overhead in Oracle, corresponding to adding 1955 seconds, and 158% in SQL Server, which adds 1598 seconds of extra loading time.

Overall, the loading time results presented in this section mostly confirm those shown in the previous chapter, although different CPUs were used between them. Tables 5-2, 5-3 and 5-4 summarize the fact table loading

time results respectively for the TPC-H 1GB, TPC-H 10GB and Sales DW, for each DBMS, highlighting the best solutions in each case.

Table 5-2. TPC-H 1GB Lineitem Fact Table Loading Time Overhead

	Oracle TPC-H 1GB Loading Time (Overhead)	SQL Server TPC-H 1GB Loading Time (Overhead)
Standard Loading Time	00:04:13	00:02:51
AES128	00:10:08 (00:05:55 / 140%)	00:06:22 (00:03:31 / 123%)
AES256	00:10:36 (00:06:23 / 151%)	00:06:47 (00:03:56 / 138%)
3DES168	00:10:17 (00:06:04 / 144%)	00:06:29 (00:03:38 / 127%)
OPES	00:05:53 (00:01:40 / 40%)	00:03:49 (00:00:58 / 34%)
Salsa20	00:06:59 (00:02:46 / 66%)	00:04:41 (00:01:50 / 64%)
SES-DW128	00:04:39 (00:00:26 / 10%)	00:03:11 (00:00:20 / 12%)
SES-DW256	00:04:54 (00:00:41 / 16%)	00:03:21 (00:00:30 / 18%)
SES-DW1024	00:07:31 (00:03:18 / 78%)	00:04:44 (00:01:53 / 66%)

Table 5-3. TPC-H 10GB Lineitem Fact Table Loading Time Overhead

	Oracle TPC-H 10GB Loading Time (Overhead)	SQL Server TPC-H 10GB Loading Time (Overhead)
Standard Loading Time	00:42:56	00:29:56
AES128	01:46:15 (01:03:19 / 147%)	01:09:04 (00:39:08 / 131%)
AES256	01:52:22 (01:09:26 / 162%)	01:15:32 (00:45:36 / 152%)
3DES168	01:48:47 (01:05:51 / 153%)	01:11:30 (00:41:34 / 139%)
OPES	01:02:46 (00:19:50 / 46%)	00:42:22 (00:12:26 / 42%)
Salsa20	01:14:41 (00:31:45 / 74%)	00:51:46 (00:21:50 / 73%)
SES-DW128	00:50:24 (00:07:28 / 17%)	00:35:37 (00:05:41 / 19%)
SES-DW256	00:53:36 (00:10:40 / 25%)	00:38:40 (00:08:44 / 29%)
SES-DW1024	01:20:44 (00:37:48 / 88%)	00:58:36 (00:28:40 / 96%)

Table 5-4. Sales DW 2GB Fact Table Loading Time Overhead

	Oracle Sales DW 2GB Loading Time (Overhead)	SQL Server Sales DW 2GB Loading Time (Overhead)
Standard Loading Time	00:16:34	00:16:53
AES128	00:44:36 (00:28:02 / 169%)	00:40:16 (00:23:23 / 139%)
AES256	00:48:09 (00:31:35 / 191%)	00:42:53 (00:26:00 / 154%)
3DES168	00:49:09 (00:32:35 / 197%)	00:43:31 (00:26:38 / 158%)
OPES	00:25:55 (00:09:21 / 56%)	00:25:54 (00:09:01 / 53%)
Salsa20	00:31:42 (00:15:08 / 91%)	00:31:19 (00:14:26 / 86%)
SES-DW128	00:18:44 (00:02:10 / 13%)	00:19:21 (00:02:28 / 15%)
SES-DW256	00:20:11 (00:03:37 / 22%)	00:20:37 (00:03:44 / 22%)
SES-DW1024	00:30:08 (00:13:34 / 82%)	00:31:21 (00:14:28 / 86%)

5.4.3. Analyzing Query Performance

To analyze the query performance of the encryption algorithms, we defined a decision support query workload for each database similar to what was described in the data masking technique's experimental evaluation in the previous chapter. The TPC-H workload included the benchmark queries were the same as those used in the data masking experiments in the previous chapter (i.e., TPC-H queries number 1, 3, 6, 7, 8, 10, 12, 14, 15, 17, 19 and 20, which correspond to all that access the LineItem fact table). For the Sales DW, the workload was also the same set of 29 queries all processing the Sales fact table. For fairness, databases were also optimized in a best practice manner (including primary keys, foreign keys, and referential integrity constraints and join indexes).

As we previously mentioned, all response time results are an average obtained from six executions in each scenario on each DBMS. The standard execution time (average of execution time of the workload against a non-encrypted database) for each scenario is 492, 5037, and 1766 seconds in Oracle 11g, and 452, 4694, and 1690 seconds in SQL Server 2008, for the 1GB, 10GB TPC-H and Sales DW, respectively.

Figures 5-10a to 5-11b show the total workload execution time and its overhead in Oracle and SQL Server for the TPC-H 1GB database, while Figure 5-12 shows the CPU execution time overhead in Oracle and SQL

Server for the same database. SES-DW with 128-bit and 256-bit security has the best response and CPU time overheads for all scenarios, respectively 106% and 154% of execution time overhead in Oracle, corresponding to 523 and 759 seconds of added response time, and 105% and 152% in SQL Server, corresponding to 475 and 688 seconds of added response time. The results are followed by Salsa20 and further by AES, while OPES has results leveled between AES and 3DES, while SES-DW with 1024 bit security presents values approximately similar to AES.

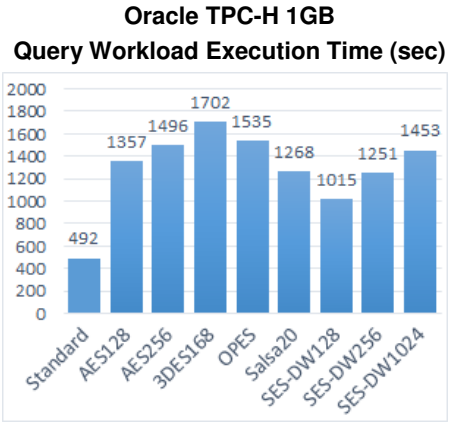


Figure 5-10a. Query Workload Execution Time in Oracle for the TPC-H 1GB per Encryption Solution

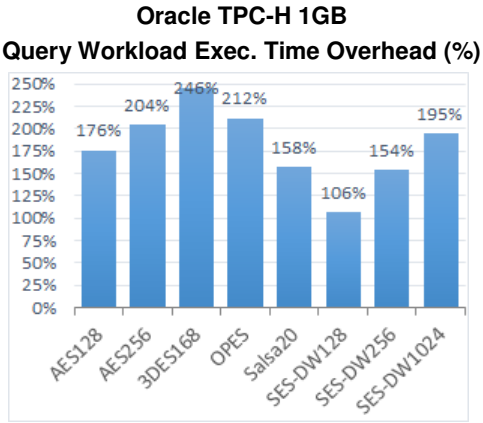


Figure 5-10b. Query Workload Exec. Time Overhead (%) in Oracle for the TPC-H 1GB per Encryption Solution

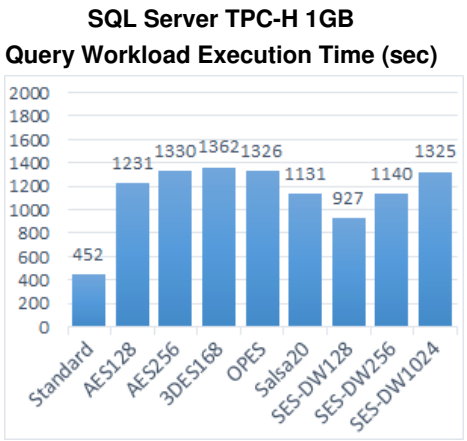


Figure 5-11a. Query Workload Execution Time in SQL Server for the TPC-H 1GB per Encryption Solution

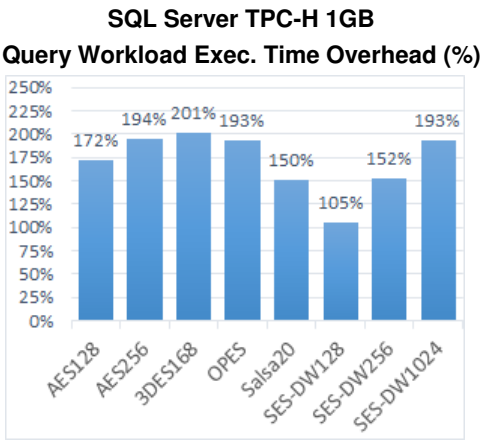


Figure 5-11b. Query Workload Exec. Time Overhead (%) in SQL Server for the TPC-H 1GB p/ Encryption Solution

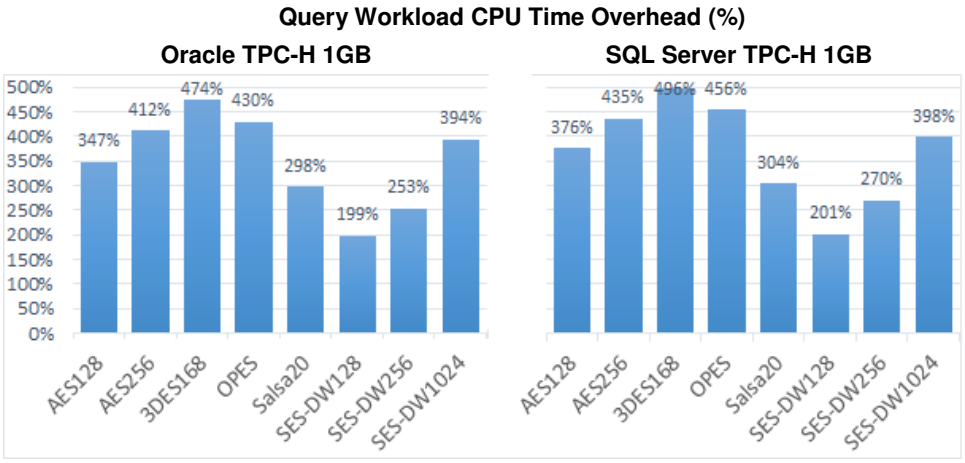


Figure 5-12. Query Workload CPU Time Overhead (%) for the TPC-H 1GB per Encryption Solution in each DBMS

It can be seen that in what concerns the processing efforts of the encryption algorithms themselves, which can be observed through analyzing the CPU execution time overhead, the results shown in Figure 5-12 show that SES-DW introduces an overhead of approximately 200% to 270% respectively with 128 and 256 bit security. Salsa20, which is the best of the remaining solutions, introduces approximately 300%, while all other solutions add nearly 400% of CPU execution time overhead.

Figures 5-13a to 5-14b show the total workload execution time and its overhead in Oracle and SQL Server for the TPC-H 10GB database, while Figure 5-15 shows the CPU execution time overhead in Oracle and SQL Server for the same database. As can be observed, the results lead to similar conclusions as those seen in the TPC-H 1GB database, in what respects the ranking performance of the tested solutions. SES-DW remains the solution having the best response and CPU time overheads for all scenarios, with 128-bit and 256-bit security in both DBMS. When compared with the results for the TPC-H 1GB database, it can be seen that the differences between the solutions are slightly enforced with the higher amount of data which needs to be processed in the 10GB scale size.

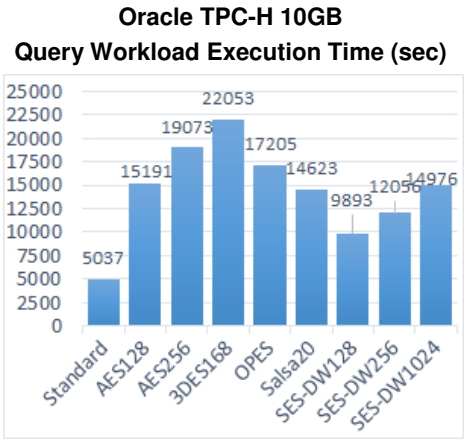


Figure 5-13a. Query Workload Execution Time in Oracle for the TPC-H 1GB per Encryption Solution

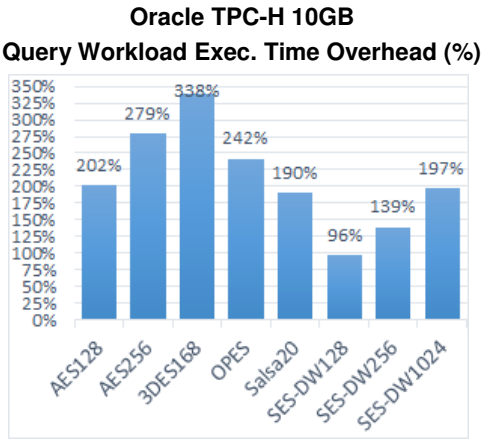


Figure 5-13b. Query Workload Exec. Time Overhead (%) in Oracle for the TPC-H 1GB per Encryption Solution

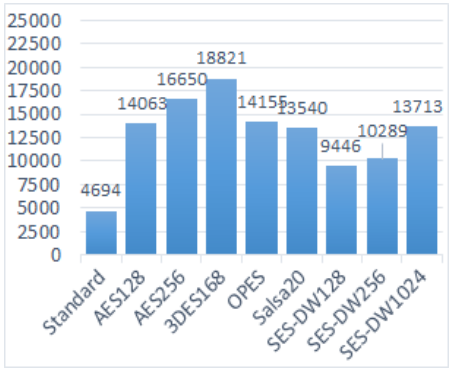


Figure 5-14a. Query Workload Execution Time in SQL Server for the TPC-H 10GB per Encryption Solution

Figure 5-14b. Query Workload Exec. Time Overhead (%) in SQL Server for the TPC-H 10GB p/ Encryption Solution

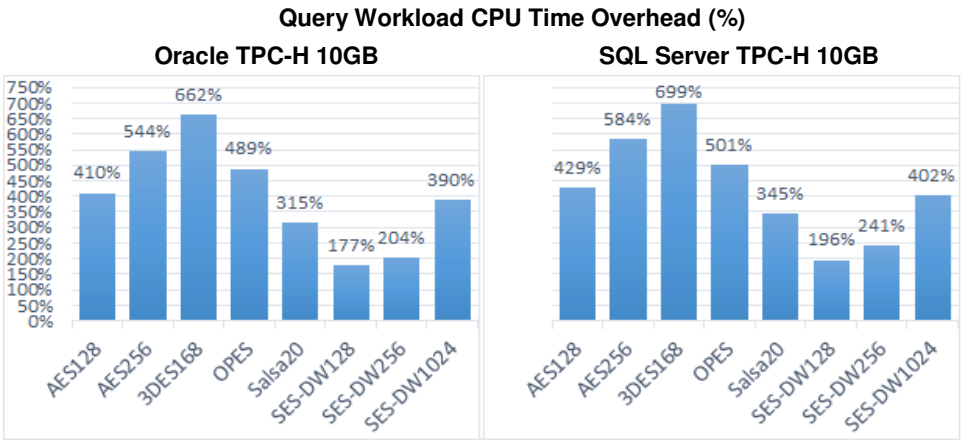


Figure 5-15. Query Workload CPU Time Overhead (%) for the TPC-H 10GB per Encryption Solution in each DBMS

Figures 5-16a to 5-17b show the results of total workload execution time and respective overhead for the Sales DW fact table in both DBMS. It can be seen that SES-DW continues to produce much smaller execution time overheads than the remaining solutions, similarly to the occurred with TPC-H. SES-DW with 128 bit security shows 262% and 236% overhead in Oracle and SQL Server, respectively corresponding to an extra 4627 and 3988 seconds in response time. SES-DW with 256 bit security shows 409% and 361% in Oracle and SQL Server, corresponding to adding 7223 seconds in Oracle and 6101 seconds in SQL Server, and when using 1024 bit security the loading time overhead was 610% in Oracle and 493% in SQL Server, respectively corresponding to an extra 10773 and 8332 seconds of loading time.

Salsa20 comes after SES-DW 128 bit and 256 bit in execution time performance, showing a 539% overhead in Oracle and 492% in SQL Server, and OPES presents more than 700% and 600% overhead respectively in Oracle and SQL Server. The standard encryption solutions continue to produce the highest overhead, roughly between 700% and 800% of extra loading time in both DBMS.

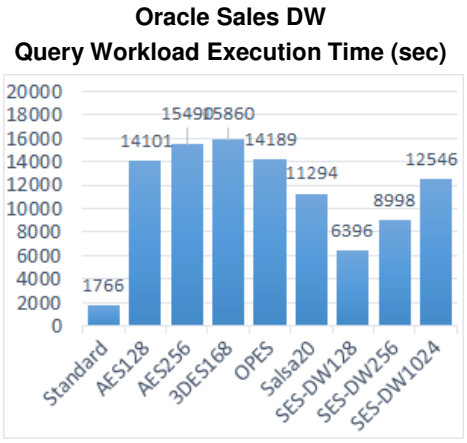


Figure 5-16a. Query Workload Execution Time in Oracle for the Sales DW per Encryption Solution

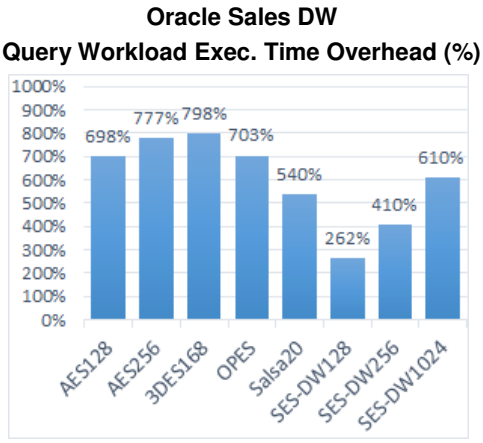


Figure 5-16b. Query Workload Exec. Time Overhead (%) in Oracle for the Sales DW per Encryption Solution

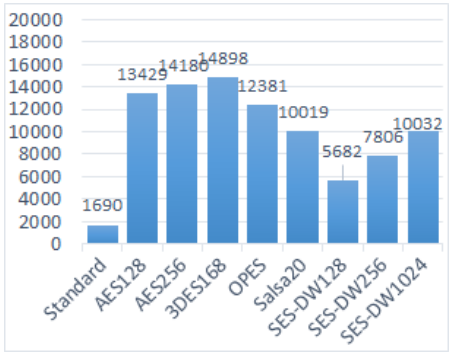


Figure 5-17a. Query Workload Execution Time in SQL Server for the Sales DW per Encryption Solution

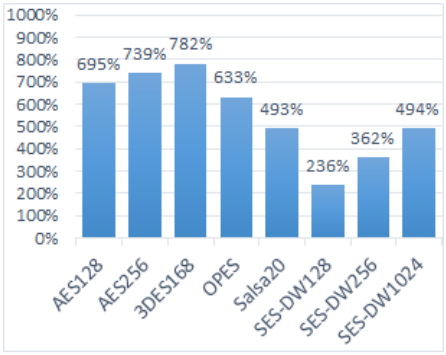


Figure 5-17b. Query Workload Exec. Time Overhead (%) in SQL Server for Sales DW p/ Encryption Solution

Figure 5-18 shows the CPU time overhead per solution for the Sales DW in each DBMS. In what concerns CPU time overhead, by observing Figure 5-18 and comparing it with the results from the TPC-H 1GB in Figure 5-12 and TPC-H 10GB in Figure 5-15, it can be seen that the CPU execution time overhead obtained in the Sales DW are very leveled and similar to those obtained in the TPC-H databases. This reveals a similar difference and impact in CPU processing efforts between the different solutions.

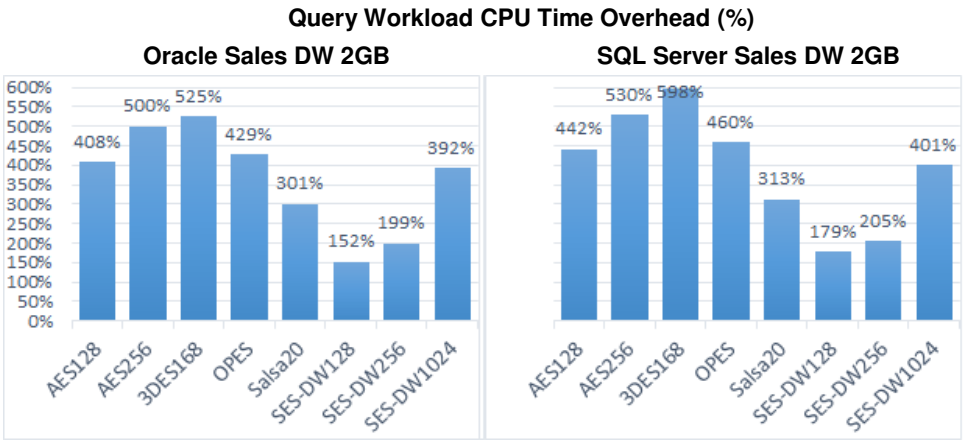


Figure 5-18. Query Workload CPU Time Overhead (%) for the Sales DW 2GB per Encryption Solution in each DBMS

Tables 5-5, 5-6 and 5-7 summarize the query workload execution time results respectively for the TPC-H 1GB, TPC-H 10GB and Sales DW, for each DBMS. We highlight SES-128 as the solution that achieves the best results.

Table 5-5. TPC-H 1GB Query Workload Execution Time Overhead

	Oracle TPC-H 1GB Execution Time (Overhead)	SQL Server TPC-H 1GB Execution Time (Overhead)
Standard Loading Time	00:08:12	00:07:32
AES128	00:22:37 (00:14:25 / 176%)	00:20:31 (00:12:59 / 172%)
AES256	00:24:56 (00:16:44 / 204%)	00:22:10 (00:14:38 / 194%)
3DES168	00:28:22 (00:20:10 / 246%)	00:22:42 (00:15:10 / 201%)
OPES	00:25:35 (00:17:23 / 212%)	00:22:06 (00:14:34 / 193%)
Salsa20	00:21:08 (00:12:56 / 158%)	00:18:51 (00:11:19 / 150%)
SES-DW128	00:16:55 (00:08:43 / 106%)	00:15:27 (00:07:55 / 105%)
SES-DW256	00:20:51 (00:12:39 / 154%)	00:19:00 (00:11:28 / 152%)
SES-DW1024	00:24:13 (00:16:01 / 195%)	00:22:05 (00:14:33 / 193%)

Table 5-6. TPC-H 10GB Query Workload Execution Time Overhead

	Oracle TPC-H 10GB Execution Time (Overhead)	SQL Server TPC-H 10GB Execution Time (Overhead)
Standard Loading Time	01:23:57	01:18:14
AES128	04:13:11 (02:49:14 / 202%)	03:54:23 (02:36:09 / 200%)
AES256	05:17:53 (03:53:56 / 279%)	04:37:30 (03:19:16 / 255%)
3DES168	06:07:33 (04:43:36 / 338%)	05:13:41 (03:55:27 / 301%)
OPES	04:46:45 (03:22:48 / 242%)	03:55:55 (02:37:41 / 202%)
Salsa20	04:03:43 (02:39:46 / 190%)	03:45:40 (02:27:26 / 188%)
SES-DW128	02:44:53 (01:20:56 / 96%)	02:37:26 (01:19:12 / 101%)
SES-DW256	03:20:56 (01:56:59 / 139%)	02:51:29 (01:33:15 / 119%)
SES-DW1024	04:09:36 (02:45:39 / 197%)	03:48:33 (02:30:19 / 192%)

Table 5-7. Sales DW 2GB Query Workload Execution Time Overhead

	Oracle Sales DW 2GB Execution Time (Overhead)	SQL Server Sales DW 2GB Execution Time (Overhead)
Standard Loading Time	00:29:26	00:28:10
AES128	03:55:01 (03:25:35 / 698%)	03:43:49 (03:15:39 / 695%)
AES256	04:18:10 (03:48:44 / 777%)	03:56:20 (03:28:10 / 739%)
3DES168	04:24:20 (03:54:54 / 798%)	04:08:18 (03:40:08 / 782%)
OPES	03:56:29 (03:27:03 / 703%)	03:26:21 (02:58:11 / 633%)
Salsa20	03:08:14 (02:38:48 / 540%)	02:46:59 (02:18:49 / 493%)
SES-DW128	01:46:36 (01:17:10 / 262%)	01:34:42 (01:06:32 / 236%)
SES-DW256	02:29:58 (02:00:32 / 410%)	02:10:06 (01:41:56 / 362%)
SES-DW1024	03:29:06 (02:59:40 / 610%)	02:47:12 (02:19:02 / 494%)

To demonstrate the effects of using encryption on each individual query, the results for individual query execution time in Oracle for the TPC-H 10GB scenarios are shown in Figure 5-19, with a logarithmic scale. These results show that all queries have similar proportional overhead to those of the complete workload. This is also true for all the other scenarios, making it redundant to include all. It can be seen that most queries

processed by AES and 3DES have overheads of several orders of magnitude higher than SES-DW.

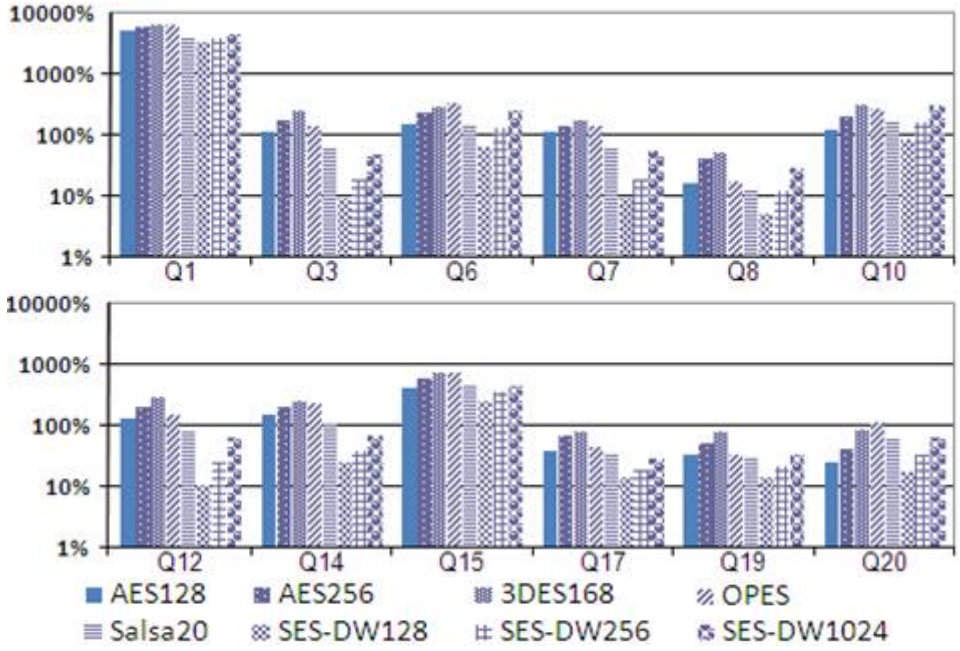


Figure 5-19. TPC-H 10GB Individual Query Execution Time Overhead per Encryption Algorithm in Oracle 11g

For the same reasons as in the experimental evaluation subchapter of the proposed data masking solution, the individual query execution time results for the Sales DW are not included, given this set of queries can produce a certain amount of insight as a whole, but should mainly not be considered as appropriate for individual analysis, since this DW is a specific real-world database and it is not a standard nor a benchmark.

5.5 Discussion on SES-DW

Contrarily to typical encryption packages such as those supplied by the leading commercial DBMS, SES-DW preserves the encrypted columns' datatype and bit length. This avoids introducing storage space overhead and type conversions in decryption, consequently decreasing the amount of data that needs to be accessed in order to process queries, as well as computation efforts, when compared with typical encryption. As the data

masking technique proposed in the previous chapter, SES-DW accomplishes continuous data protection similarly to commercial encryption packages, since it maintains data-at-rest encrypted at all times, while adding the mentioned benefits of datatype preservation.

SES-DW also has similar advantages to MOBAT, such as executing direct queries against encrypted/decrypted data without having that data transferred between the database and the encryption/decryption mechanisms. This also avoids I/O and network bandwidth congestion that other solutions introduce due to those data roundtrips, enabled by the fact that the encrypting and decrypting processes simply rely on SQL rewriting. As MOBAT, SES-DW is a straightforward and portable technique to be universally used in any DBMS regardless of the CPU and operating system, contrarily to what occurs with most standard encryption packages supplied by DBMS.

Another advantage in SES-DW that is similar to MOBAT is that SES-DW is specifically designed for masking numerical values, and in this sense, is therefore much more performance efficient for protecting DW facts, when compared with standard encryption techniques that require executing data type conversions. The data loading and query execution response time results shown in the experimental evaluations demonstrate this, as it also proves that using encryption does in fact introduce extremely high storage space, loading time and query response time overheads.

In what concerns storage space, SES-DW presents similar overhead as MOBAT, concerning the addition of an extra column in the fact table. OPES and SES-DW introduce much smaller storage space overheads than the remaining solutions (less than 25% of extra storage space), followed by Salsa20 at a considerable difference (adding approximately 30% of storage space in TPC-H and almost 100% in the Sales DW), while the standard encryption solutions produce the highest storage space overheads by far.

In what concerns loading time, SES-DW with 128 bit and 256 bit security (those similar to the key lengths of the other solutions) is much faster than all the remaining solutions, introducing 10% to 29% of extra loading time in the tested scenarios. OPES has the second best performance, introducing 34% to 61% of extra loading time, more than two times worse than SES-DW on average. Salsa20 presents loading time overheads from 64% to 102%, on average roughly four times worse than SES-DW, while the

standard encryption solutions introduce overheads of more than 100%, reaching more than 200% in several scenarios. On the other hand, while SES with 1024 bit security does present greater overhead than OPES and Salsa20, it does have a superior bit security strength than these solutions.

Considering the results obtained in the query workload executions, SES-DW with security strengths similar to the remaining solutions is also much faster. By observing the results, it can be seen that the relative differences between the solutions are approximately proportional throughout the different scenarios, with SES-DW being always the fastest solution (using the same bit security strength as the key length of the remaining solutions) and therefore introducing the smallest execution time overheads by several orders of magnitude, roughly half to a quarter, on average, of the remaining solutions.

SES-DW can be considered as a much more efficient overall solution, introducing small overheads when compared to the remaining solutions, for similar key sizes. Note that the worst result for SES-DW is that with 1024 bit security, which is similar to Salsa20. However, it does refer to using 1024 bit encryption, far higher than the remaining tested solutions.

As we previously mentioned, given that decision support environments typically execute long running queries (*i.e.*, queries that can run for many minutes up to hours), the response time overheads introduced due to the use of encryption solutions represent high absolute values that can easily make query responses overdue and jeopardize the usefulness of the DW itself. Considering the magnitude of the results shown in the experimental evaluations, even a minimum gain in response/CPU time can be considered as an important achievement.

The proposed encryption technique is straightforward and nearly effortless to implement in a similar fashion as the data masking technique, and the encryption keys may also be periodically refreshed and used to refresh the encrypted table values without much effort, by frequently switching the values of all or any one of the set of encryption keys for each encrypted column before refreshing encrypted data in order to ensure that data is properly protected. Therefore, given all of its security and performance features discussed and demonstrated in this chapter, we believe our technique is secure enough to be acceptable for use and that it

may be considered as a valid alternative for enhancing data confidentiality in DWs.

5.6 Summary

In this chapter we propose an encryption solution specifically designed for enhancing data confidentiality in DWs. The proposed encryption algorithm requires only operations that can be executed using standard SQL, such as modulus, exclusive or and arithmetic operators. As the masking technique, it requires small computational efforts and is straightforward and easily implemented in any DBMS. The proposed solution is transparently used and to query the database the user interfaces only need to send their queries to a middleware broker instead of the DBMS. Data-at-rest is always encrypted and only the final processed results are returned to the authorized user interfaces that requested them. All SQL commands and actions are encrypted and stored in a log by the middleware security broker, which can be audited by any security staff.

We have compared SES-DW with the AES and 3DES encryption algorithms provided by leading commercial DBMS, as well as two state-of-the-art encryption proposals. The experimental results confirm the same kind of storage space and database performance results as in the previous chapter. Given that most DW data consists on numerical values, our encryption technique is tailored for this kind of data. Given both security proof and performance results, our technique shows better security strength *versus* database performance tradeoffs than the remaining encryption solutions. Thus, it is an efficient overall solution and a valid alternative for balancing performance and security issues from the DW perspective.

Chapter 6

DW-DIDS: An Intrusion Detection Mechanism for Data Warehouses

In a defense in depth scenario, an intruder needs to overcome a series of security mechanisms against invasive or unauthorized actions, such as routers, firewalls, network-based intrusion detectors, OS-based intrusion detectors, and finally, Database Intrusion Detection Systems (DIDS). The DIDS represents the last bastion of defense before any intruder gains access to the data itself. In this chapter, we propose a Data Warehouse Database Intrusion Detection System (DW-DIDS) based on the analysis of user actions at the SQL command level, including measures concerning what data was processed as well as the resulting datasets from the command's execution. The proposed DIDS complies with the principles defined by the framework presented in Chapter 3.

To accomplish this, we define what an intruder is and what types of attack can occur against data warehouses, proposing a classification of each intruder action according to those intents. Given this classification and the characteristics of typical end user workloads, we propose a set of features analyzed by the DIDS which we consider relevant to analyze and monitor their behavior.

We then define how to construct each user's behavior profile using the chosen Intrusion Detection (ID) features in a defined learning phase for the DIDS, and how to perform ID in the detection phase for generating alerts.

For performing alert and response management, we propose a risk exposure method that assesses the risk inherent to each generated alert, given its probability and impact, which indicates the alerts that potentially present greater risk to the enterprise. This allows security staff to quickly check the alerts showing the highest risk and deal with the potentially most

dangerous intrusions first. The approach includes a SQL-like set of rules that allow determining the probability that each alert refers to a true intrusion given the feature that generated that alert, as well as the impact that the user action can produce on the enterprise. These rules also enable to deal with intrusions automatically, given the alert's risk exposure measure.

The chapter is organized as follows. In Section 6.1 we describe the basics of intrusion behavior in data warehousing environments, classifying the types of intrusion actions and proposing the relevant features for monitoring user behavior and performing intrusion detection. In Section 6.2 we present the overall architecture of the proposed DIDS, describing each of its components and how they operate together during the workflow of the user command's execution. Section 6.3 describes how to build user profiles, while Section 6.4 describes how to perform ID given each user action. Section 6.5 presents the risk exposure method for alert and response management. Section 6.6 includes an experimental evaluation of the proposed DIDS against two other ID techniques proposed by recent state-of-the-art research. In Section 6.7 we discuss open issues regarding the proposed DIDS and finally, Section 6.8 summarizes and concludes the chapter.

6.1. Selecting Intrusion Detection Features in Data Warehouses

Selecting the appropriate features for performing intrusion detection requires understanding what an intruder is and which are the distinct type of intentions that can drive an attack, *i.e.*, what the intruder aims to achieve with the attack.

From a database perspective, an intruder in a data warehousing environment can be one of the following [Treinen and Thurimella, 2006]:

- **An authorized user**, which is someone that has regular access to authorized database interfaces and acts with malicious intent;
- **A masqueraded user**, which is someone that obtains the credentials of an authorized user and impersonating that user takes control of an authorized interface connecting to the database;
- **An insider attacker**, which is someone that holds valid credentials to access the database as a regular activity;

- **An external attacker**, which is someone that does not have valid credentials to access the database, but is able to bypass database security mechanisms and gain direct database access using SQL injection or other exploiting techniques;
- **Any combination of the above**.

Considering the possible intruders' intentions, there are mainly three types of attacks mobilized against DWs [Douligeris and Mitrokotsa, 2004]:

- **Attacks aiming at corrupting data (integrity attacks).** In this type of attack, the intruder seeks access to the database for executing actions that compromise its integrity, such as corrupting or deleting the data in a given database object (*e.g.* such as a table or view);
- **Attacks aiming at stealing information (confidentiality attacks).** In these attacks, the intruder focuses on confidentiality issues, such as stealing business information, rather than damaging data;
- **Attacks aiming at making the DW unavailable (availability attacks).** These attacks aim on making database services unavailable, *i.e.*, they are mainly Denial of Service (DoS) attacks (*e.g.* flooding database services and bandwidth with a large number of requests, and halting or crashing database server instances).

Given these intruder intents and types of attacks, we define ten classes of intrusion action types (A...J) as shown in Table 6-1. This classification distinguishes the intruder's intentions apart from each other (shown in the "Attack Profile/Intent/Focus" column), defining a taxonomy for each action accordingly to what s/he might be aiming to achieve with the attack.

Considering that integrity attacks focus on compromising the consistency and accurateness of the data content itself, we consider as integrity attacks all intruder actions that attempt to insert new false data values (class H), change the existing data values in order to make them incorrect or inaccurate (class I) and deleting existing data (class J). Any one of these attacks will cause inaccurate query responses against the affected data and they can also compromise referential integrity constraints if dimensional data is affected.

Table 6-1. SQL Intrusion Action Type Classification

SQL Action Class	Security dimension affected by the intrusion			Intruder Command Action Description	Attack Profile/Intent/Focus
	Confid	Integrity	Availab		
A	X			Attempts to discover valid database credentials/logins	Brute force attack or dictionary-based attacks for attempting to obtain valid application/database logins
B	X			Query retrieving information on database objects or data structures	Retrieving information on database tables, views, triggers, etc. as well as index column names and types, in order to compose further attack instructions
C			X	Malicious modification of auxiliary data structures	Erasing or renaming performance optimization data structures (*e.g.* erasure of indexes or materialized views), database objects (*e.g.* tables or physical datafiles)
D	X		X	Query retrieving all data from a table (integral table copy)	Retrieving all possible information of fact tables (in order to steal business secrets or strangle network bandwidth) or dimension tables (*e.g.* customer information)
E	X			Query retrieving a significant portion of data from a table	Stealing of selected sensitive factual (*e.g.* fact rows about sales concerning a given product or time period, or the rows with a small well-chosen set of sensitive table columns) or dimensional data (*e.g.* a list of customer credit cards or addresses)
F	X			Query retrieving a specific and relatively small portion of data	Stealing a small amount of specifically targeted data (*e.g.* total year sales value of a given product)
G			X	Query flooding	Execution of an overwhelming amount of concurrent queries that access large volumes of data (creating database server processing bottlenecks) or that return large volumes of data (causing network bandwidth strangulation)
H		X		Insertion of false data	Insertion of rows with false data in fact tables and/or dimension tables to compromise user query results
I		X		Malicious modification of data	Modification of stored data values in fact tables and/or dimension tables to compromise user query results
J		X		Deletion of data	Deletion of fact and/or dimensional table rows to produce false user query results and erase sensitive data

We consider as confidentiality attacks all those that attempt to disclose information that should not be disclosed. In these intruder actions, there can be distinct intentions such as: attempting to retrieve valid credentials to access the database with certain privileges (class A, which will allow the intruder to gain access to certain parts of the database), retrieving information on the database structures, such as table names and column names, for example (class B, which will allow identifying how the data is stored in the database and how the business is analyzed); and retrieving all or certain amounts of data from the database (classes D, E and F, which discloses business information to the intruder that s/he may use in her/his benefit or dismay the enterprise).

Data availability attacks aim at keeping the database services from providing the responses back to the users or to simply keep them from operating. We consider as availability attacks user actions that: attempt to rename or delete database objects that hold data, such as tables or materialized views, or which are required to process regular user commands, such as table views (class C); request the database server to process a huge amount of data in a single command (e.g. retrieving all data from a fact table, defined in class D); and overwhelming the database server with commands, alias known as query flooding (class G).

As can be seen, the classes defined in Table 6-1 cover a broad scope of intentions posed by intrusions. This classification is generic and can be easily modified in order to widen its scope by including other classes of different types of attack.

As previously discussed, a DIDS at the database command level should be able to analyze all the aspects triggered by the execution of the user's action: the commands themselves, processed data, and resulting datasets. Given the described issues, the features required for monitoring database user actions are those focusing on the following usability dimensions:

- **Action-type:** what type of actions are being requested;
- **Traceability:** from who/where does the requested action come;
- **Selectivity:** what data will be affected by that action and what data composes the resulting dataset;
- **Time:** when are the actions requested to execute and their duration.

In order to analyze the referred dimensions given each user action, we need to capture observable measures of user behavior from each of the following inputs:

- **The user's ID and his/her session ID.** Identifying the user and session allows building individual behavior profiles, as well as trace back each requested database command;
- **The SQL commands issued by the user.** The SQL command allows using features that identify the type of command (insert, update, select, delete, etc.) and accessed data structures (columns, tables, materialized views, etc.), selection attributes and values, grouping attributes, union queries, etc.;
- **A timestamp of the issued execution request.** This allows defining the temporal behavior of each user, identifying sequences of measures as well as frequencies of occurrences, how long does it take to process each command (elapsed time), etc.;
- **The data processed by each SQL command.** The measures from the processed data allow using features concerning the data that is processed by each command that is not intrinsic to the command (*e.g.* how many rows were processed in the command's execution);
- **The dataset resulting from each SQL command's execution.** The measures from the dataset resulting from the command's execution allow using features that enable analyzing what sort of data is returned to the user (the size of the resulting dataset, how many rows and columns, data values, etc.).

Considering these inputs and the characterization of data warehousing environments and intrusion actions previously described, the intrusion detection features considered interesting to capture the relevant measures for the proposed DIDS are shown in Table 6-2. Note that although these features may seem general-purpose and well fit for intrusion detection in most types of databases, they are in fact the most relevant features for collecting the required information for monitoring data warehouse user actions and analyze their behavior, given the characteristics inherent to data warehouse user activity, as described in [Bockermann *et al.* 2009; Douligeris and Mitrokotsa, 2004; Kimball and Ross, 2013; TPC-H; TPC-DS; Treinen and Thurimella, 2006].

Table 6-2. SQL Intrusion Detection Features

F#	FeatureName	Description
User-based features		
F_1	#ConsecFailedLoginAttempts	The number of consecutive failed database login attempts by a UserID or from an IPAddress (accumulated or in a given timespan)
F_2	#SimultaneousSQLSessions	The number of active simultaneous database connections on behalf of a UserID or IPAddress
F_3	#UnauthorizedAccessAttempts	The number of consecutive requests to execute unauthorized actions (*e.g.* requesting to modify read-only data, or query data that he does not have access privileges) from a UserID or IPAddress
SQL Command-based features		
F_4	CPUTime	CPU time spent by the DBMS to process each command
F_5	ResponseSize	Size (in bytes) of the result of the command's execution
F_6	#ResponseLines	Number of lines and columns in the result of the command's execution
F_7	#ResponseColumns	Number of columns in the result of the command's execution
F_8	#ProcessedRows	Number of accessed rows for processing the command's execution
F_9	#ProcessedColumns	Number of accessed columns for processing the command's execution
F_{10}	CommandLength	Number of characters in the command
F_{11}	#GroupBy	Number of GROUP BY columns in the command
F_{12}	#Union	Number of UNION clauses in the command
$F_{13}…F_{17}$	#Sum, #Max, #Min, #Avg, #Count	Number of appearances of SUM, MAX, MIN, AVG and COUNT functions in the command
F_{18}, F_{19}	#And , #Or	Number of appearances of AND and OR operators in the command's WHERE clause(s)
F_{20}	#LiteralValues	Number of appearances of literal values in the command's WHERE clause(s)
Session-based features		
$F_{21}…F_{27}$	#Select, #Insert, #Delete, #Update, #Create, #Alter, #Drop	Number of executed SELECT, INSERT, DELETE, UPDATE, CREATE, ALTER, and DROP commands per session
F_{28}	#Insert-Select	Number of executed INSERT commands that used SELECT commands for inserting or building datasets, per session
F_{29}	#Create-Select	Number of executed CREATE commands that used SELECT commands for inserting or building datasets, per session
F_{30}	TimeBetwCommands	Time period (in seconds) between execution of commands, per session
F_{31}	#SimultaneousCommands	Number of commands simultaneously executing, per session

Table 6-2. SQL Intrusion Detection Features (continued)

F#	FeatureName	Description
Table-based features		
F_{32}	#ProcessedRows	Number of accessed rows per table
F_{33}	#ProcessedColumns	Number of accessed columns per table
F_{34}...F_{38}	#Sum, #Max, #Min, #Avg, #Count	Nr. of appearances of SUM, MAX, MIN, AVG and COUNT functions executed per table
F_{39}…F_{42}	#Select, #Insert, #Delete, #Update	Number of executed SELECT, INSERT, DELETE, and UPDATE commands per table
Column-based features		
F_{43}	#GroupBy	Number of issued GROUP BY clauses per column
F_{44}…F_{48}	#Sum, #Max, #Min, #Avg, #Count	Nr. of SUM, MAX, MIN, AVG and COUNT functions executed per column
F_{49}, F_{50}	#Select, #Update	Number of executed SELECT, and UPDATE commands per column

As can be observed in Table 6-2, the features are divided into five main groupings: user-based, command-based, session-based, table-based and column-based. This allows testing features by applying different levels of grouping (per user / per user session / per SQL command / per table / per column) as roll-up and drill-down techniques, widening the detection scope and coverage of user behavior variability.

Table 6-3 shows the coverage of the intrusion detection features defined in Table 6-2 against the intrusion action classes described in Table 6-1. Given the diverse types of intrusion detection techniques discussed in Chapter 2, the set of proposed features presented in our approach manages to cover an extremely broad scope of possible forms of intrusion detection. For example, features F_1, F_2, F_3, F_4, F_5, F_{30}, F_{31} are commonly used in intrusion detection systems that inspect network traffic; F_6, F_8…F_{29}, F_{34}…F_{50} are widely used for SQL command analysis; F_4, F_5, F_6, F_{13}…F_{17}, F_{34}…F_{38}, F_{44}…F_{48} are used in statistical intrusion detection systems; F_4, F_{21}…F_{27}, F_{30}, F_{31}, F_{39}…F_{42}, F_{49}, F_{50} are used for sequence analysis; F_6…F_9, F_{11}, F_{12}, F_{21}…F_{29}, F_{32}, F_{33}, F_{39}…F_{43}, F_{49}, F_{50} focus on the accessed data and are used in intrusion detection systems for data access pattern analysis; and features F_4…F_7, F_{30}…F_{33} are used in intrusion detection systems that analyze the action's resulting dataset.

Table 6-3. SQL Intrusion Detection Features Coverage per Intrusion Action Class

SQL Action Class	Intrusion Detection Features
A	F_1, F_2, F_3, F_8, F_9, F_{10}, F_{12}, F_{18}, F_{19}, F_{20}, F_{30}, F_{31}, F_{32}, F_{33}, F_{49}
B	F_2, F_3, F_8, F_9, F_{18}, F_{19}, F_{20}, F_{30}, F_{31}, F_{32}, F_{33}, F_{49}
C	F_3, F_8, F_9, F_{10}, F_{18}, F_{19}, F_{20}, F_{22}, F_{23}, F_{24}, F_{25}, F_{26}, F_{27}, F_{28}, F_{30}, F_{31}, F_{32}, F_{33}, F_{40}, F_{41}, F_{42}, F_{50}
D	F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10}, F_{12}, F_{18}, F_{19}, F_{20}, F_{21}, F_{22}, F_{25}, F_{26}, F_{28}, F_{29}, F_{30}, F_{31}, F_{32}, F_{33}, F_{39}, F_{49}
E	F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10}, F_{11}, F_{12}, F_{13}, F_{14}, F_{15}, F_{16}, F_{17}, F_{18}, F_{19}, F_{20}, F_{21}, F_{22}, F_{25}, F_{26}, F_{28}, F_{29}, F_{30}, F_{31}, F_{32}, F_{33}, F_{34}, F_{35}, F_{36}, F_{37}, F_{38}, F_{39}, F_{43}, F_{44}, F_{45}, F_{46}, F_{47}, F_{48}, F_{49}
F	F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10}, F_{11}, F_{12}, F_{13}, F_{14}, F_{15}, F_{16}, F_{17}, F_{18}, F_{19}, F_{20}, F_{21}, F_{22}, F_{25}, F_{26}, F_{28}, F_{29}, F_{30}, F_{31}, F_{32}, F_{33}, F_{34}, F_{35}, F_{36}, F_{37}, F_{38}, F_{39}, F_{43}, F_{44}, F_{45}, F_{46}, F_{47}, F_{48}, F_{49}
G	F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10}, F_{11}, F_{12}, F_{18}, F_{19}, F_{20}, F_{21}, F_{22}, F_{28}, F_{29}, F_{30}, F_{31}, F_{32}, F_{33}, F_{39}, F_{40}, F_{43}, F_{49}
H	F_2, F_3, F_4, F_8, F_9, F_{10}, F_{18}, F_{19}, F_{20}, F_{22}, F_{28}, F_{29}, F_{30}, F_{31}, F_{32}, F_{33}, F_{39}, F_{40}, F_{49}
I	F_2, F_3, F_4, F_8, F_9, F_{10}, F_{18}, F_{19}, F_{20}, F_{24}, F_{26}, F_{30}, F_{31}, F_{32}, F_{33}, F_{42}, F_{50}
J	F_2, F_3, F_4, F_8, F_{10}, F_{18}, F_{19}, F_{20}, F_{23}, F_{27}, F_{30}, F_{31}, F_{32}, F_{41}

6.2. DW-DIDS Architecture

The Data Warehouse Database Intrusion Detection System's (DW-DIDS) architecture is shown in Figure 6-1. The *DataBase Administrator (DBA)* is the person in charge of managing the *DW Database(s)*, namely managing all database objects such as datafiles, tablespaces, tables, indexes, views, etc. The *Authorized End User* is a regular authorized DW end user that is interested in querying data for decision support purposes or an ETL tool. The *Intruder* represents the attackers as defined in the previous section.

The *DW Security Administrator* is responsible for handling the DW-DIDS through the *Security Manager Interface* by managing the contents of the *DW-DIDS Database* (which is a part of the *Security Framework Database*, as explained in Chapter 3). This database contains:

- A historical SQL command log for storing all commands requested to be executed by the DBMS;
- The individual user feature profiles and respective statistical models;
- A historical alert log for storing and monitoring all generated alerts;

- A rule-base dataset containing the rules for computing risk exposure and indicating how to deal with intrusions according to each generated alert (the syntax of the risk exposure rules will be explained further in Section 6.5).

The generated alerts stored in the alert log are also manually confirmed as true or false positive outcomes by the *DW Security Administrator*, after their veracity have been checked out. The true and false positive outcomes are used to fine-tune each feature's contribution in the overall intrusion detection process, as explained further in Subsection 6.5.4.

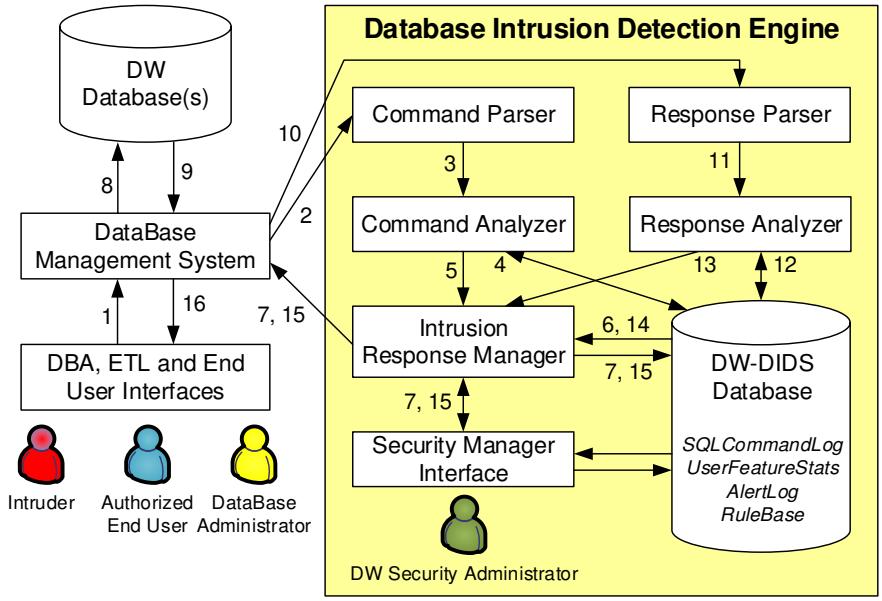


Figure 6-1. DW-DIDS Architecture

In our approach, intrusion detection is handled at the SQL command level in two moments:

1) when the DBMS receives a command to execute, that command is analyzed before it is executed (step 2);

2) after its execution is completed (if the command is not considered an intrusion in step 2), its response and the data that was processed is also analyzed before being returned to the user which requested the execution (step 10).

The sequence of steps is labeled in Figure 6-1. In practice, before executing any command, the *Command Parser* retrieves the command text and starting date/time, as well as user identification (User type, UserID, IPAddress, SessionID), parses the command according to the intrusion detection features and passes all the information to the *Command Analyzer* (step 3). The *Command Analyzer* stores this information in the *SQL Command Log* and retrieves the respective user features' statistical models (step 4), and applies the intrusion detection algorithms (explained in the next subsections) to determine if an alert should be generated concerning the analyzed command. The information referring the parsed user command and its outcome results from the intrusion detection tests is then passed on to the *Intrusion Response Manager* (*IRM*) (step 5).

When the *IRM* receives indication that an alarm should be generated, it retrieves the probability, impact and risk exposure rule set from the *DW-DIDS Database* (step 6), evaluates the intrusion's risk exposure and stores the data concerning the alert and the features which generated it in the database (for future reference), and notifies the *DW Security Administrator* through the *Security Manager Interface* (step 7). Moreover, it also takes the suitable actions for dealing with the possible intrusion through the DBMS, accordingly to what is defined by the risk exposure rules. The *IRM* takes action against intrusions by suspending or killing its execution, or killing the user session, either automatically or on request of the *DW Security Administrator* after s/he has seen the alert information and decided what action should be taken.

If the command is not considered an intrusion *a priori* to its execution, *i.e.*, if no alarm is generated after analyzing the command, DW-DIDS will simply update each feature's statistical model for the corresponding user in the *DW-DIDS Database* and notify the DBMS to execute the command. In this case, after its execution, the resulting dataset and the data that was processed is parsed by the *Response Parser* and analyzed by the *Response Analyzer* (in a similar way as the applied by the *Command Parser* to the user command) (steps 10 to 13).

If the *Response Analyzer* does not request to generate an alarm against the command's resulting dataset, *i.e.*, if it is not considered an intrusion, then each feature's statistical models for the concerning user is updated once more in the *DW-DIDS Database* and the command's results are disclosed

back to the user that requested them. On the other hand, if the IRM receives indication that an intrusion alert should be generated, then it takes action similarly to what was previously described for steps 6 and 7.

6.3. Learning Phase: Building User Behavior Profiles

Our user profiling approach is based on adjusting a probabilistic distribution for each ID feature { F_1, …, F_{50} } (as shown in Table 6-2) per user, except F_1 and F_3 (which use absolute values), from observations (feature values) extracted in an initial training (alias learning) stage. To obtain those observations, we assume the existence of a previous "intrusion-free" database command log or a set of queries supplied by the DW administrator, which also identify the user that issued each command.

To build the user profiles, each SQL user command in that log or set of queries is parsed and executed against the DW to extract the required information, *i.e.*, the observations from the command itself, those referring to the data processed by the command, and the resulting dataset, for building each feature's statistical distribution per user. The workflow of this training stage is shown in Figure 6-2, where the continuous lines show the flow *a priori* to the user command's execution and the dashed lines indicate the flow *a posteriori*.

Statistical adjustment tests are performed in order to obtain each population's distribution model at a level of 5% significance using Qui-square (which is valid for any distribution), Kolmogorov-Smirnov (which is valid for a continuous distribution) or Shapiro-Wilks (valid for normal distributions) to verify if each set of observations comes from a population with a given distribution function F_0, specified on the null hypothesis.

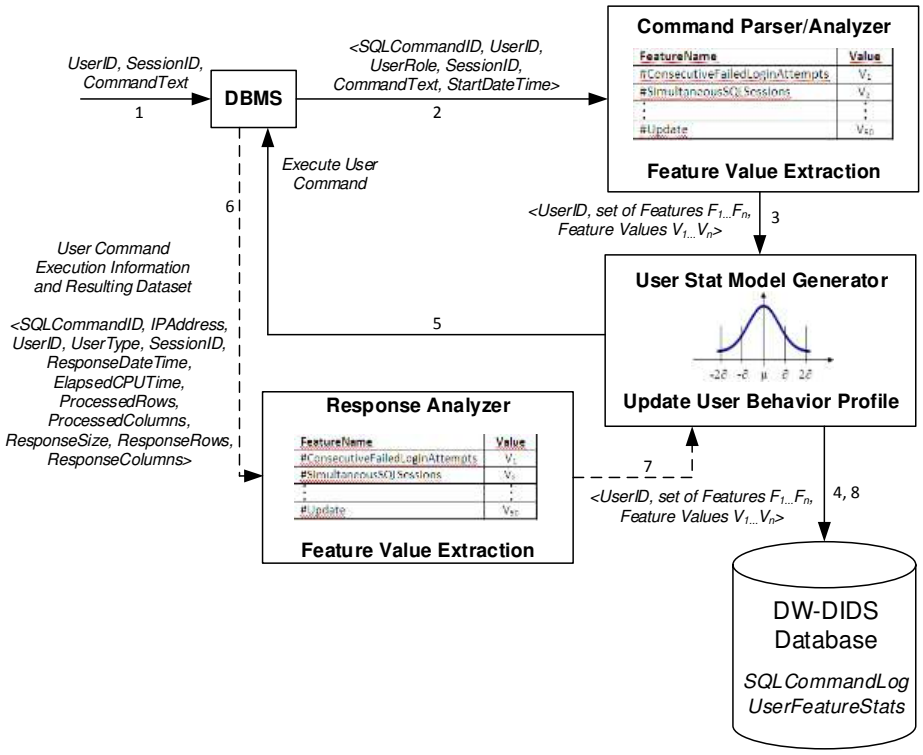


Figure 6-2. DW-DIDS Learning Stage Workflow for each SQL User Command

6.4. Detection Phase: Intrusion Detection against User Commands

The testing phase workflow for performing intrusion detection is shown in Figure 6-3, where continuous lines show the flow *a priori* to the user command's execution and the dashed lines indicate the flow *a posteriori*. To detect an intrusion, each user command is analyzed before it is executed by the DBMS. A statistical test is performed for each feature given its original statistical model for the respective user and a new sample set built by gathering the existent observations with the current respective user session sample set for that feature. New statistical tests are performed to adjust a new probability distribution to the former data collection. Afterwards, we test if the new distribution matches the original distribution of the feature (H_o).

The Chi-square, Kolmogorov-Smirnov or Shapiro-Wilk statistical tests, mentioned in the previous subsection, are always used as the testing methods in all cases, all performed at a level of 5% significance. These

methods test whether one distribution (*e.g.* one data set) is significantly different from another (*e.g.* a normal distribution) and produce a binary answer, corresponding to yes or no. We use the Shapiro-Wilk test if the sample size is small (between 3 and 2000) and the Kolmogorov-Smirnov test if the sample size is big (greater than 2000). The Chi-square test is used to verify if a data sample came from a population with a specific distribution.

If no test in this first phase (*i.e.*, *a priori* to the user command's execution by the DBMS) rejects H_0, then the DBMS is notified to run the command. After the command has been processed, feature value extraction is performed on the resulting dataset and the processed data and the corresponding statistical tests are executed in a similar fashion as described in the previous paragraph. In any testing phase, for each feature's test result that rejects the distribution's equality (H_o) in any moment, the respective user action is considered an intrusion and an alarm is generated.

For features F_1 and F_3 a different approach is chosen, considering the following: in systems such as ATM, banking, e-governance, and most web applications, for instance, the number of allowed consecutive unsuccessful login attempts is typically three (which is the most used option) to five (usually the maximum number of allowed consecutive unsuccessful attempts). It is considered common to accept two consecutive unsuccessful attempts followed by a successful attempt as a non-intrusion, while more consecutive unsuccessful attempts indicate a possible intrusion tentative or a true user that has forgotten his/her login information. Thus, DW-DIDS considers an intrusion more than two consecutive failed login attempts (F_1>2) on behalf of a given user/IP address and generates the correspondent alert.

In a similar fashion, a situation where a user that manages to login and tries to view or process data to which s/he does not have or is not supposed to access may also match an intrusion action. Therefore, two consecutive attempts from a given user/IP address for accessing unauthorized data or for executing an unauthorized command (*e.g.* an INSERT, UPDATE, DROP, etc., by a *DW End User*, which has only SELECT statement privileges) (F_3>=2) is also considered an intrusion by DW-DIDS, generating the correspondent alert.

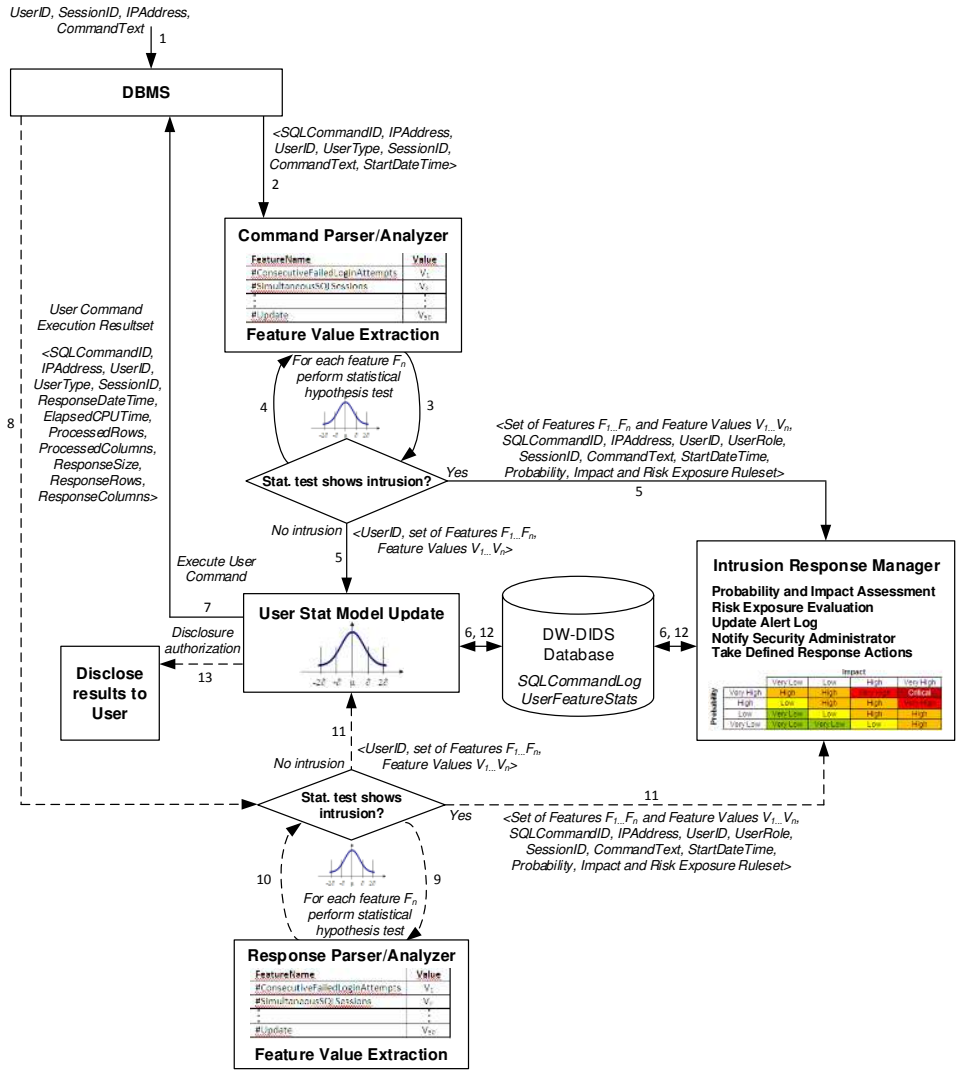


Figure 6-3. DW-DIDS Intrusion Test/Detection Stage Workflow for each SQL User Command

6.5. Alert and Response Management

For each user action that flags an alert, the *Intrusion Response Manager (IRM)* evaluates the potential damage the action may cause to the enterprise, assessing the action's risk exposure according to the feature(s) that generated the alert. After computing that risk exposure measure, it notifies the *DW Security Administrator* about the alert and adequately responds to

the intrusion accordingly with the defined risk exposure rule. In this section we define risk exposure and explain how this measure is computed in order to rank the alerts and take action against the attack. We also show how to calibrate the contribution of each intrusion detection feature in the overall intrusion detection process.

6.5.1. Defining the Risk Exposure

Many DIDS evaluate *what* data is accessed, while others focus on *how* data is accessed. Both assess the probability of a given user action being suspicious to classify that particular action or set of actions to which it belongs as an intrusion; when that probability exceeds a predefined threshold, an alert is generated. As we have previously mentioned, any thresholds used to filter out intrusion alerts given their probability should be defined with low values that minimize the risk of false negatives, *i.e.*, to minimize the number of true intrusions that pass undetected. Given the sensitivity of DW data, it is preferable to have low thresholds, as the potential cost of undetection is often considered too high or unacceptable. However, this exponentially increases the number of generated alerts in most scenarios, making alert management one of the most critical issues in intrusion detection scenarios.

To improve the efficiency of intrusion detection systems when the number of generated alerts is extremely high, alert correlation techniques such as [Debar and Wespi, 2001; Ning *et al.*, 2002; Pietraszek, 2004; Pietraszek and Tanner, 2005; Valdes and Skinner, 2001; Yu *et al.*, 2007] have been proposed. These techniques typically filter sets of alerts to distinguish which are worthy of being checked from those that are more probably false alarms. However, we argue that alert correlation on itself is not the best way to determine which alerts should be checked and in which order of priority.

Since the value of DWs resides on the fact that they store the secrets of the business, the impact resulting from an intrusion on the enterprise is intimately linked with what data was exposed or corrupted. When using alert correlation techniques, there can be an alert that has been positively correlated for checking but has a low potential impact on the enterprise (*e.g.* the exposed or damaged data is not very sensitive), while an alert referring a true intrusion with high impact can be filtered out if it has a low

correlation value. Moreover, not evaluating the potential impact of the intrusion means that security staff do not know which alerts are more important, implying that resources may be wasted in checking intrusion alerts referring to actions that would cause minimal damage to the enterprise, while a highly prejudicial intrusion occurs and is left to be dealt with later on.

To avoid this, we propose considering all alerts admissible and apply a method for ranking them, given a measure of risk exposure. Given a user action, *risk exposure* is a function of both the *probability* that the action has of referring an intrusion and the *impact* that it may produce, *i.e.*, the potential magnitude of the cost to the enterprise related with the damage or disclosure of the data targeted by the action. The computation of the risk exposure of each alert is done according to the matrix shown in Figure 6-4, given its measured probability and impact.

		Impact			
		Very Low	Low	High	Very High
Probability	Very High	High	High	Very High	Critical
	High	Low	High	High	Very High
	Low	Very Low	Low	High	High
	Very Low	Very Low	Very Low	Low	High

Figure 6-4. The risk exposure matrix

The risk exposure method assures that all generated alerts will be ranked and automatically inform security staff to check out and deal with the most significant intrusions (given alerts with higher risk exposure) prior to possible intrusions that might potentially produce less damage, thus performing alert management more efficiently.

To determine which actions are taken as a response for each alert given its risk exposure assessment, the *DW Security Administrator* should define rules with the following syntax (where the values enclosed in {} represent sets of values to choose from and those in [] are optional clauses):

```
 GIVEN RISK EXPOSURE AS {VeryLow, Low, High, VeryHigh,
                         Critical}
    ON FEATURE {FeatureName1, FeatureName2, ...},
               [AllFeatures]
[WHERE {List of filtering conditions}]
 [WHEN {List of time-based conditions}]
  TAKE ACTION {DoNothing, PauseUserCommand,
               TerminateUserCommand, KillUserSession}
   FOR USERS {User1, User2, ...} [, [AllUsers,]
       USERS WITH ROLE {Role1, Role2, ...}
```

This SQL-like rule covers all user action classes and dimensions mentioned in Section 6.1. The *FOR USERS, WHEN* and *WHERE* clauses allow conditioning the application of the intrusion response actions defined in the *TAKE ACTION* clause, according to the specified features included in the *ON FEATURE* clause to which the generated alert refers. The *FOR USERS* clause allows the rule to be applied only to a limited subset of users, the *WHEN* clause allows the rule to be valid only during a given time schedule, and the *WHERE* clause allows the rule to be valid only given certain conditions using feature weight values – feature weighting is explained in the next subsection.

As an example of defining risk exposure rules, consider feature *#ConsecFailedLoginAttempts* from Table 6-2. Supposing the *DW Security Administrator* wants to be alerted each time an alert is risen by this feature and defines that *High* and *Very High* risk exposure assessments for this feature should terminate the respective user commands, while a *Low* assessment should suspend the user command until the administrator checks if everything is alright, for all users. This is accomplished by:

```
GIVEN RISK EXPOSURE AS Low
   ON FEATURE #ConsecFailedLoginAttempts
 TAKE ACTION PauseUserCommand
  FOR USERS AllUsers
GIVEN RISK EXPOSURE AS VeryHigh, High
   ON FEATURE #ConsecFailedLoginAttempts
 TAKE ACTION TerminateUserCommand
  FOR USERS AllUsers
```

As another example, if all users requesting to execute any command that generates critical alerts – regardless of the feature that generated them – should immediately be banned, the following rule can be defined:

```
GIVEN RISK EXPOSURE AS Critical
   ON FEATURE AllFeatures
 TAKE ACTION TerminateUserCommand, KillUserSession
  FOR USERS AllUsers
```

On the other hand, considering that all the command that generate alerts which present a *Very Low* risk exposure measure can be executed normally, although the *Security Manager Interface* still displays the alert to the *DW Security Administrator* so they can be checked out, the following rule can be defined:

```
GIVEN RISK EXPOSURE AS VeryLow
   ON FEATURE AllFeatures
 TAKE ACTION DoNothing
  FOR USERS AllUsers
```

6.5.2. Defining the Probability

DW-DIDS defines the probability of each intrusion alert with rules, given the feature that generated the alert. In a similar manner to the risk exposure rules, these rules have the following syntax:

```
DEFINE PROBABILITY AS {VeryLow, Low, High, VeryHigh}
    ON FEATURE {FeatureName1, FeatureName2, ...},
               [AllFeatures]
[WHERE {List of filtering conditions}]
 [WHEN {List of time-based conditions}]
   FOR USERS {User1, User2, ...}, [AllUsers,]
       USERS WITH ROLE {Role1, Role2, ...}
```

It is quite obvious that, depending on each DW's context, each feature has its own importance in the overall intrusion detection process, which is directly related to its risk probability, *i.e.*, its efficiency in producing high true positive rates (detection of a high amount of true intrusions) and low false positive rates (small amounts of false alarms). To define this importance, each feature has a weight attributed to it, which is a real value within the range [0…1]. Using the probability rule syntax, we propose that the risk probability of each feature F_i should have a significance directly linked to its weight, as:

```
DEFINE PROBABILITY AS VeryLow
    ON FEATURE Fi WHERE Weight(Fi)<0.25
   FOR AllUsers
DEFINE PROBABILITY AS Low
    ON FEATURE Fi WHERE Weight(Fi)>=0.25 AND Weight(Fi)<0.50
   FOR AllUsers
DEFINE PROBABILITY AS High
    ON FEATURE Fi WHERE Weight(Fi)>=0.50 AND Weight(Fi)<0.75
   FOR AllUsers
DEFINE PROBABILITY AS VeryHigh
    ON FEATURE Fi WHERE Weight(Fi)>=0.75
   FOR AllUsers
```

After the learning phase in which all user profiles are built and DW-DIDS runs for the first time to detect and respond to intrusions, we suggest giving an equal weight of 0.5 to all features (`Weight(Fi)=0.5`), since it is not possible to know *a priori* which features will reveal to be more significant in the intrusion detection process. However, after the DW security staff checks each generated intrusion alert, the value of each feature's weight is calibrated by its revealed efficiency. This weight calibration technique is explained in Subsection 6.5.4.

For the fixed value features F_1 and F_3 we use predefined constants for defining the probability rule. For example, in banking and e-governance applications the number of consecutive unsuccessful login attempts that are allowed typically ranges from three to five. As mentioned before, it is common to accept that two consecutive unsuccessful login attempts followed by a successful attempt as a non-intrusion, while more consecutive unsuccessful tries indicate a possible intrusion attempt. Given this, the probability of an intrusion given the number of consecutive failed login attempts can be defined as:

```
DEFINE PROBABILITY AS VeryLow
    ON FEATURE #ConsecFailedLoginAttempts
 WHERE #ConsecFailedLoginAttempts<=2
   FOR AllUsers
DEFINE PROBABILITY AS Low
    ON FEATURE #ConsecFailedLoginAttempts
 WHERE #ConsecFailedLoginAttempts=3
   FOR AllUsers
DEFINE PROBABILITY AS High
    ON FEATURE #ConsecFailedLoginAttempts
 WHERE #ConsecFailedLoginAttempts=4
   FOR AllUsers
```

```
DEFINE PROBABILITY AS VeryHigh
    ON FEATURE #ConsecFailedLoginAttempts
 WHERE #ConsecFailedLoginAttempts>=5
   FOR AllUsers
```

Note that this is only an example and that although the statistical features have a proposed predefined set of rules given their computed importance/weight in the overall intrusion detection process, the *DW Security Administrator* can define new rules to widen the probability scope (in the same way s/he can add new features). To give an example on using temporal conditioning on any feature, consider a context in which no user is expected to access the DW between 8p.m. and 7a.m. on the server time clock. This may be defined in a rule as:

```
DEFINE PROBABILITY AS VeryHigh
    ON FEATURE #ProcessedRows, CommandLength
 WHERE (Server.Time>20:00 OR Server.Time<7:00) AND
       (#ProcessedRows>0 OR CommandLength>0)
   FOR AllUsers
```

Given the wide scope allowed by the defined rules, there may be more than one type of probability assessed when checking the rules that concern a generated intrusion alert. For instance, the same feature might have a *High* probability given from one of the rules and a *VeryHigh* probability attributed by another rule. In this case, the *Intrusion Response Manager* always chooses to assign the highest value (in this case, *VeryHigh*).

6.5.3. Defining the Impact

The assessment of the impact caused by a user action is also defined by rules in a similar fashion as those previously described. This assessment is based on *which*, *how much*, and *when* sensitive data can be exposed or damaged by the user command, as well as *who* is the user. The impact for the actions ranged by each user's command is managed by the following rules, valid for the list of nominal-based, value-based and/or temporal-based conditions is defined through rules with the following syntax:

```
DEFINE IMPACT AS VeryLow, Low, High, VeryHigh
    ON FEATURE {FeatureName1, FeatureName2, ...},
       [AllFeatures]
   [ON COMMAND Insert, Update, Delete, Select,
               CreateAll, DropAll, AlterAll,
               CreateTable, DropTable, AlterTable,
               CreateIndex, DropIndex, AlterIndex,
               CreateProcedure, DropProcedure,
               AlterProcedure, CreateFunction,
               DropFunction, AlterFunction,
               CreateView, DropView, AlterView,
               CreateTrigger, DropTrigger,
               AlterTrigger, AllCommands, DML, DDL]
  [WITH COLUMNS {Column1, Column2, ...}, [AllColumns]]
 [WHERE {List of filtering conditions}]
  [WHEN {List of time-based conditions}]
[JOINED WITH {Column1, Column2, ...}, [AllColumns]]
    FOR USERS {User1, User2, ...}, [AllUsers,]
       USERS WITH ROLE {Role1, Role2, ...}
```

This impact assessment is left entirely to the *DW Security Administrator*, as it depends on the nature and structure of each DW itself and is mostly unique in each real-world context. The clauses are used in a similar manner to those in the probability rules, plus the clause that allows distinguishing which is the user command (*ON COMMAND*), which columns are processed (*WITH COLUMNS*), and the clause defining the impact of two or more columns being processed or shown together by the same command (*JOINED WITH COLUMNS*).

As an example, suppose that a credit sales DW has a *Sales* fact table with column *SalesAmount*, storing the total amount value of each sale. It is probable that a command that retrieves a single row or two of *SalesAmount* values from the fact table probably represents low exposure risk for the enterprise in case of an intrusion, while that risk may probably be very high if the number of retrieved rows is higher (*e.g.* greater than 20). This can be defined by the following rules:

```
DEFINE IMPACT AS Low
    ON FEATURE #ProcessedRows
    ON COMMAND Select
  WITH COLUMNS Sales.SalesAmount
 WHERE COUNT(*)<=2 FOR USERS AllUsers
```

```
DEFINE IMPACT AS VeryHigh
    ON FEATURE #ProcessedRows
    ON COMMAND Select
  WITH COLUMNS Sales.SalesAmount
  WHERE COUNT(*)>=20 FOR USERS AllUsers
```

6.5.4. Calibrating Feature Weight

The efficiency of intrusion detection mechanisms is typically analyzed recurring to several measures [Kamra *et al.*, 2008; Kamra, 2010]:

- *True Positive (TP)*: an alert referring to a true intrusion;
- *False Positive (FP)*: an alert which reveals a false alarm;
- *True Negative (TN)*: a user action that is correctly classified as a non-intrusion by the ID process;
- *False Negative (FN)*: an intrusion action that is misclassified by the ID process as a non-intrusion (*i.e.*, resulting in a missed intrusion).

The importance of each feature in DW-DIDS is computed by a self-calibrating technique, using its individual ∑TP and ∑FP values. For each feature F_i, its weight is given by:

$$Weight\ (F_i) = 0.5 + \frac{\left(\frac{\sum TP_i - \sum FP_i}{\sum TP_i + \sum FP_i}\right)}{2},\ \sum TP_i > 0 \vee \sum FP_i > 0$$

where $\sum TP_i$ and $\sum FP_i$ respectively represent the total number of true positives and false positives achieved by all the alerts generated by feature F_i. In our approach we assume *a priori* that each statistical feature initially has the same relevance. When DW-DIDS runs for the first time (and until the first alert generated by F_i, which allows computing TP_i and FP_i), each feature's weight is set to an initial value of 0.50, as previously explained in Subsection 6.5.2. This value represents a neutral value in the formula, where the number of alerts generated by the feature refers to a true intrusion are the same as the number of alerts referring to false alarms:

$$\sum TP_i = \sum FP_i \Rightarrow \left(\frac{\sum TP_i - \sum FP_i}{\sum TP_i + \sum FP_i}\right) = 0 \Rightarrow Weight\ (F_i) = 0.5 + \frac{0}{2} = 0.5$$

Every time an intrusion alert is generated, it needs to be checked *a posteriori* by the *DW Security Administrator* and then its status (true positive or false positive) is stored in the *DW-DIDS Database*. Each feature's weight linked

to that alert is then updated accordingly to the calibration weight formula. In case $\sum TP_i > \sum FP_i$, the second term of the sum is positive, which makes the feature's weight higher than 0.5. Contrarily, when $\sum TP_i < \sum FP_i$ the second term of the sum is negative, which makes the feature's weight lower than 0.5, implying it erroneously alerts intrusions more than it accurately does. As the values of TP_i or FP_i grow, the computed weight will also respectively get higher or lower, meaning that as the values of TP_i and FP_i vary through time the computed weight will faithfully reflect the feature's intrusion detection probability.

6.6. Experimental Evaluation

Given the inexistence of an intrusion detection benchmark at the SQL command level, we used the well-known TPC-H decision support benchmark [TPC-H] to build the "true" non-intrusion workloads and a set of diverse artificially created "intrusion" workloads in the experiments.

For the "true" DW users, the respective workloads were taken from the TPC-H benchmark due to its representativeness of typical DW workloads, and defined according to the following assumptions:

- A number of randomly chosen TPC-H benchmark queries were selected for each user's workload, *i.e.*, each user has different queries to execute, as well as a different number of queries to execute;
- Within the queries for each workload, several were randomly picked for modifying the benchmark's fixed parameters (namely in their WHERE clause) by randomly changing their values to obtain a larger scope of diverse user actions from the benchmark queries;
- A number of randomly built queries (by randomly picking a set of tables, columns, functions to execute, grouping and sorting, and literal restrictions for columns in the WHERE clauses) were also generated for each workload, representing the *ad hoc* user queries in DW environments;
- The proportion of TPC-H and randomly built queries used in each workload is respectively 80% and 20% (on average), representing the typical reporting behavior in DW's as the majority of the running tasks, while *ad hoc* queries are simulated by the random queries, in smaller number.

Given that TPC-H has 23 predefined queries, the composed workload for each "true" user is shown in Table 6-4 for a setup consisting of 10 users, where *O* means that we are using the original TPC-H query, and *M* stands for a TPC-H query with modified parameters, as explained previously.

Table 6-4. "Non-Intrusion" True User Workloads (TUW)

	Users									
Queries	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
TPC-H Q1	O	O			M			M	O	
TPC-H Q2		M	O			O				O
TPC-H Q3	M				O		O	M	O	
TPC-H Q4		O	M			M			M	O
TPC-H Q5				M	O		M			
TPC-H Q6	O	M						O		M
TPC-H Q7			M	O		O			M	
TPC-H Q8	M	O					M			O
TPC-H Q9			M	O		M		O		
TPC-H Q10		O			M		O		O	
TPC-H Q11	O				M					M
TPC-H Q12	M		O			O	M	O		
TPC-H Q13		O			M				M	
TPC-H Q14			O	M						O
TPC-H Q15	M	M				O	O			
TPC-H Q16	O		M					M	O	
TPC-H Q17		O			M		O			M
TPC-H Q18		M			O	O	M		M	
TPC-H Q19	M				O	M		O		
TPC-H Q20		O			M		O		M	M
TPC-H Q21	O				M	M		M		O
TPC-H Q22		M			O					
TPC-H Q23			O	M		M		O		O
Nr. of Random Queries	2	3	1	5	3	2	5	1	2	2

To build each "intruder" workload, we generated a random number of actions for each intrusion action type defined in Section 6.1 and executed them in a random order. The types of intrusion actions cover a wide range

of attacks against the database, accordingly with the DW attack actions and classes formerly described in Section 6.1, as follows:

- Inserting a random amount of rows;
- Changing a random amount of rows and columns;
- Deleting a random amount of rows and columns;
- Selecting a random amount of columns from a random number of tables, without range value restrictions (1);
- Selecting a random amount of columns with a random amount of functions (MAX, SUM, etc.) from a random number of tables, without range value restrictions (2);
- Selecting a random amount of columns from a random number of tables with a random amount of grouping columns, without range value restrictions (3);
- Selecting a random amount of columns with a random amount of functions (MAX, SUM, etc.) from a random number of tables with a random amount of grouping columns, without range value restrictions (4);
- Similar to (1), with range value restrictions;
- Similar to (2), with range value restrictions;
- Similar to (3), with range value restrictions;
- Similar to (4), with range value restrictions;
- Union queries with a random number of columns and tables;
- Query flooding;
- Unauthorized DW end user actions (create, drop, etc).

For comparison with other DIDS, we repeated the experiments using the fine-grained Role-Based access control DIDS (RB-DIDS) solution proposed in [Kamra *et al.*, 2008] and the clustered Data-Centered DIDS (DC-DIDS) proposed in [Mathew *et al.*, 2010]. Both these solutions are explained in Chapter 2. The machine used in these experiments was the same used for the experiments presented in Chapter 5, with a Core2Duo 3GHz CPU and 2GB of RAM, using Oracle 11g as the DBMS.

DC-DIDS was implemented accordingly to the referred paper, using K-means clustering [Mathew *et al.*, 2010]. In their paper [Kamra *et al.*, 2008], the authors of RB-DIDS define vectors named *quiplets* that store information on the columns used in the WHERE selection clause as well as the accessed tables and columns to be displayed included in the SELECT projection clause. They also propose three types of granularity (coarse, medium-grain and fine-grained quiplets) for building the user profiles. For fairness, we include the results from the implementation using the medium-coarse quiplet, which obtained the best results in our tests, using K-means clustering with 10 iterations and the statistical Median Absolute Deviation (MAD) test for the detection process.

For our testing scenario, we consider that the most sensitive data relates to the most recent data. Since TPC-H has approximately seven years of business data, the implementation of DW-DIDS defined the data from the most recent year to have *very high* impact due to intrusion actions, the data from the two previous years as *high* impact, the data from the two years before that as *low* impact and the remaining as having *very low* impact. Of course, this is not a real scenario, but we consider it to be a sufficiently realistic setup to test our approach. As we previously explained, the definition of impact on the data is directly related to the sensitivity of the data values themselves, which varies from case to case. This is why this assessment should be done by the *DW Security Administrator* according to the specific business context.

Four user scenarios were considered for testing, with a total of 10 users in each scenario. Scenario 10-0 specifies a setup without any intruder activity, *i.e.*, there is no "intruder" workload running, while in scenarios 9-1, 8-2 and 5-5 there are respectively one, two and five "intruders" amongst the 10 users.

6.6.1. Building User Profiles

Each user profile is comprised by the set of statistical models (one per feature) that refer to his/her workloads. To build the statistical models for each feature of each "true" user, we used 5, 25, 50 and 100 executions of the "True" Users' Workloads (TUW) previously shown in Table 6-4. The data and user workload in the learning phases are considered intrusion-free and representative of normal usage because they are built and run "as defined"

in the TPC-H benchmark. We shall now analyze the time and resources required to build these profiles.

Table 6-5 shows the required storage space (in kilobytes) for building the user profiles. As can be seen, the smallest user profile database was built from 305 SQL commands, referring to the 5-5 Scenario with the execution of 5 TUW workloads, while the largest user profile database, referring to Scenario 10-0 with the execution of 100 TUW workloads, which contains a set of 12000 SQL queries.

As shown in Table 6-5 in the largest setup, RBAC-DIDS, DC-DIDS and DW-DIDS respectively needed nearly 234 KB, 1031 KB and 2767 KB of storage space, corresponding to an average of 20, 88 and 236 bytes of data per SQL command. Given that the storage space typically required by DWs ranges through many gigabtyes or terabtyes, we may conclude that the measured sizes for the user profiles can be considered insignificant.

Table 6-5. Required Storage Space for building User Profiles

Scenario	# Executions	# TUW SQL Commands	Required Storage Space (Kbytes)		
			RBAC-DIDS	DC-DIDS	DW-DIDS
10-0	5	600	11.7	51.6	138.3
	25	3000	58.6	257.8	691.7
	50	6000	117.2	515.6	1383.4
	100	12000	234.4	1031.3	2766.8
9-1	5	540	10.5	46.4	124.5
	25	2700	52.7	232.0	622.5
	50	5400	105.5	464.1	1245.1
	100	10800	210.9	928.1	2490.1
8-2	5	485	9.5	41.7	111.8
	25	2425	47.4	208.4	559.1
	50	4850	94.7	416.8	1118.1
	100	9700	189.5	833.6	2236.3
5-5	5	305	6.0	26.2	71.4
	25	1525	29.8	131.1	356.8
	50	3050	59.6	262.1	713.7
	100	6100	119.1	524.2	1427.3

In what concerns the time spent in building the user profiles, the measured costs can also be deemed insignificant when compared with the typical response time of long running queries, intrinsic characteristics of user actions in DW environments. For building all the user profiles, RBAC-DIDS took less than 1 minute, DC-DIDS took approximately 4 minutes and DW-DIDS nearly 6 minutes.

6.6.2. Intrusion Detection Efficiency

The complete "true" user and intruder workload of the testing (intrusion detection) phase for each scenario is shown in Table 6-6.

Table 6-6. Workload Quantification for each User Scenario

Scenario	# "True" Queries	# Attack Queries
10-0	1250	0
9-1	1130	100
8-2	1020	200
5-5	660	500

Based on the previously mentioned TP, TN, FP and FN measures, derived calculations are commonly used to measure the efficiency of intrusion detection mechanisms, such as [Kamra *et al.*, 2008; Kamra, 2010]:

- *TP Rate (TPR)* $= \frac{TP}{TP+FN}$
- *FP Rate (FPR)* $= \frac{FP}{FP+TN}$
- *Accuracy* $= \frac{TP+TN}{TP+FP+TN+FN}$
- *Precision* $= \frac{TP}{TP+FP}$

For the performed experiments, Figures 6-5a to 6-5c respectively show the *TP Rate (TPR)* and *FP Rate (FPR)* of DW-DIDS, RBAC-DIDS and DC-DIDS for each scenario using the user profiles built in the learning stage for each TUW training set, and Figures 6-6a to 6-6c show their *Accuracy* and *Precision*. All results are the average of 10 repeated executions for each setup (and there full statistical measures can be seen in Appendix C).

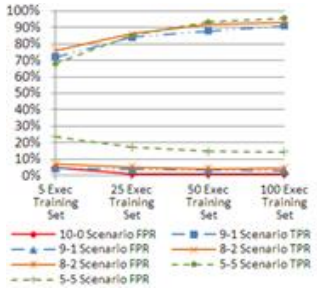


Figure 6-5a. DW-DIDS TP and FP rates

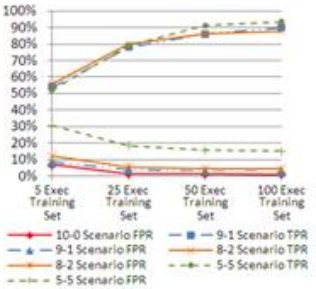


Figure 6-5b. RBAC-DIDS TP and FP rates

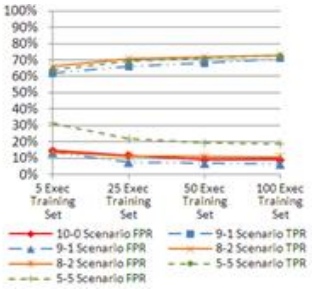


Figure 6-5c. DC-DIDS TP and FP rates

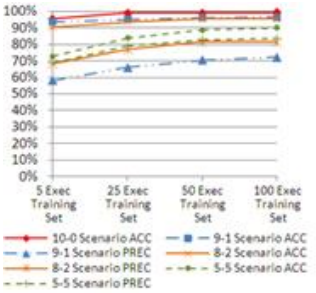


Figure 6-6a. DW-DIDS Accuracy (ACC) and Precision (PREC)

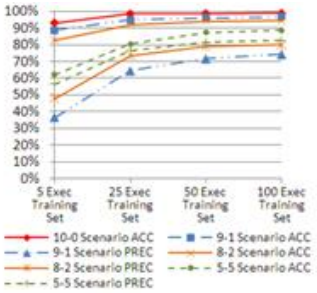


Figure 6-6b. RBAC-DIDS Accuracy (ACC) and Precision (PREC)

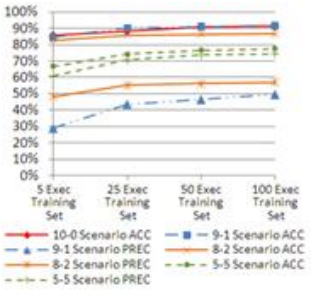


Figure 6-6c. DC-DIDS Accuracy (ACC) and Precision (PREC)

As shown in Figures 6-5.a to 6-5.c, the TP rates resulting from the scenarios in which the user profiles were built from only 5 TUW executions are relatively low for all DIDS (ranging from 52% to 78%), while in those built from 25 or more TUW executions the TP rates ranged between 85% and 94% for DW-DIDS and between 79% and 94% for RB-DIDS, while DC-DIDS obtains the worst TPR result, ranging between 65% and 72%.

The observed FP rates are all relatively low for DW-DIDS and RB-DIDS (ranging from 1% to 7%) in all scenarios except the 5-5 scenario, where 14% to 23% of the alerts result in false alarms for DW-DIDS, 15% to 30% for RB-DIDS, and 19% to 31% for DC-DIDS. This should be somewhat expected, since the 5-5 scenario represents an environment with heavy intrusion activity ($\pm$50% of the total input workload). This results in a heavy increase of alarm generation, and given the high difficulty in distinguishing normal from abnormal behavior (as previously described), the probability of generating false alarms consequently increases.

As seen in Figures 6-6.a to 6-6.c, the accuracy is high in all scenarios except 5-5, ranging between 90% and 99% for DW-DIDS, between 83% and 99% for RB-DIDS, and between 82% and 90% for DC-DIDS. In the 5-5 scenario, DW-DIDS maintains the best accuracy results between 72% and 90%, RB-DIDS between 62% and 89%, and DC-DIDS between 68% and 78%. The precision results are considerably high for DW-DIDS in all scenarios, ranging from 58% to 83%, variable in RB-DIDS by ranging from 36% to 83%, and the poorest for DC-DIDS, ranging from 29% to 50%.

Another commonly used metric to evaluate ID efficiency is the *F-score* (or *F-measure*) [Kamra *et al.*, 2008; Kamra, 2010]. This measure is preferred by many authors to score efficiency, because it scores the balance (as a harmonic mean) between *Precision* and *Recall* (alias *TP rate*) in a single output:

- $F\text{-}score = \frac{2*Precision*Recall}{Precision+Recall}$

Figures 6-7.a to 6-7.c show the *F-score* results in each scenario for each DIDS. It can be seen that DW-DIDS obtains the best results for all scenarios and TUW setups, followed by RB-DIDS, while DC-DIDS has the worst results in most cases. DW-DIDS and RB-DIDS present very similar results for the setups in which the training SQL dataset is fairly significant in size (>=25 TUW), although DW-DIDS has always a slight advantage. On the other hand, the DC-DIDS presents better results than RB-DIDS when the training dataset is small (5 TUW) in the 9-1 and 8-2 scenarios, suggesting that in these cases the data-centric analysis produces more efficient results than the command-centric analysis. Since DW-DIDS includes analysis on both data and command features, this mostly explains why DW-DIDS presents better results in all cases.

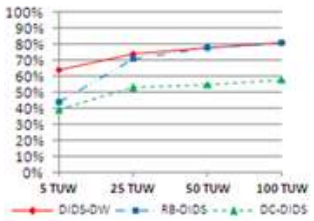


Figure 6-7a. F-Score for the 9-1 Scenario

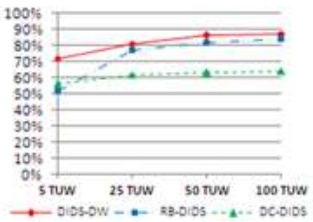


Figure 6-7b. F-Score for the 8-2 Scenario

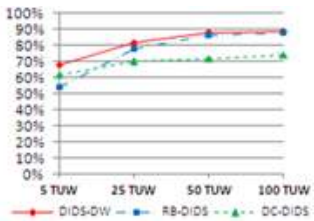


Figure 6-7c. F-Score for the 5-5 Scenario

6.6.3. Analyzing the Generated Alerts per Risk Exposure Measure

Given that one of the main advantages of ranking the alerts using the risk exposure approach is to separate the most urgent alerts that need to be checked out from those which represent a lower risk to the enterprise, we shall now analyze the generated alerts per risk exposure measure. Tables 6-7a to 6-7d show the number of generated alerts for each risk exposure measure, in each scenario. Recall the previously presented Table 6-6 referring to the number of "true" SQL instructions versus the number of "intrusion" SQL instructions for each scenario (10-0, 9-1, 8-2 and 5-5).

Table 6-7a. Alerts per Risk Exposure Measure w/ Profiles built from 5 TUW Executions

Scenario	Very Low	Low	High	Very High	Critical	Total # Alerts
10-0	11	13	14	12	7	57
9-1	27	28	28	30	12	125
8-2	41	52	59	42	29	223
5-5	88	93	116	111	89	497

Table 6-7b. Alerts per Risk Exposure Measure w/ Profiles built from 25 TUW Executions

Scenario	Very Low	Low	High	Very High	Critical	Total # Alerts
10-0	2	2	3	3	3	13
9-1	22	30	31	29	15	127
8-2	40	52	57	47	29	225
5-5	103	118	139	116	63	539

Table 6-7c. Alerts per Risk Exposure Measure w/ Profiles built from 50 TUW Executions

Scenario	Very Low	Low	High	Very High	Critical	Total # Alerts
10-0	1	2	3	4	2	12
9-1	20	30	30	26	17	123
8-2	36	50	59	49	31	225
5-5	108	123	138	125	69	563

Table 6-7d. Alerts per Risk Exposure Measure w/ Profiles built from 100 TUW Executions

Scenario	Very Low	Low	High	Very High	Critical	Total # Alerts
10-0	1	1	2	4	2	10
9-1	22	23	26	35	20	126
8-2	34	51	55	54	32	226
5-5	109	125	131	136	71	572

By observing the previous tables it can be seen that in scenario 10-0, while there are no "intrusion" actions, DW-DIDS using profiles built from 25 TUW raises 19 false alerts (corresponding to 1,5% of user statements), while in the remaining setups that amount of false alarms decreases to 1% or less, as a result of building more accurate profiles due to having more TUW to build it from.

Figure 6-8 shows the percentage of alerts per risk exposure measure, given each scenario and user profile database setup. It can be seen that the most relevant alerts (very high and critical) represent approximately one third of all alerts, which should be the ones first deserving full attention on behalf of the security staff, instead of wasting potentially precious time checking the remaining alerts.

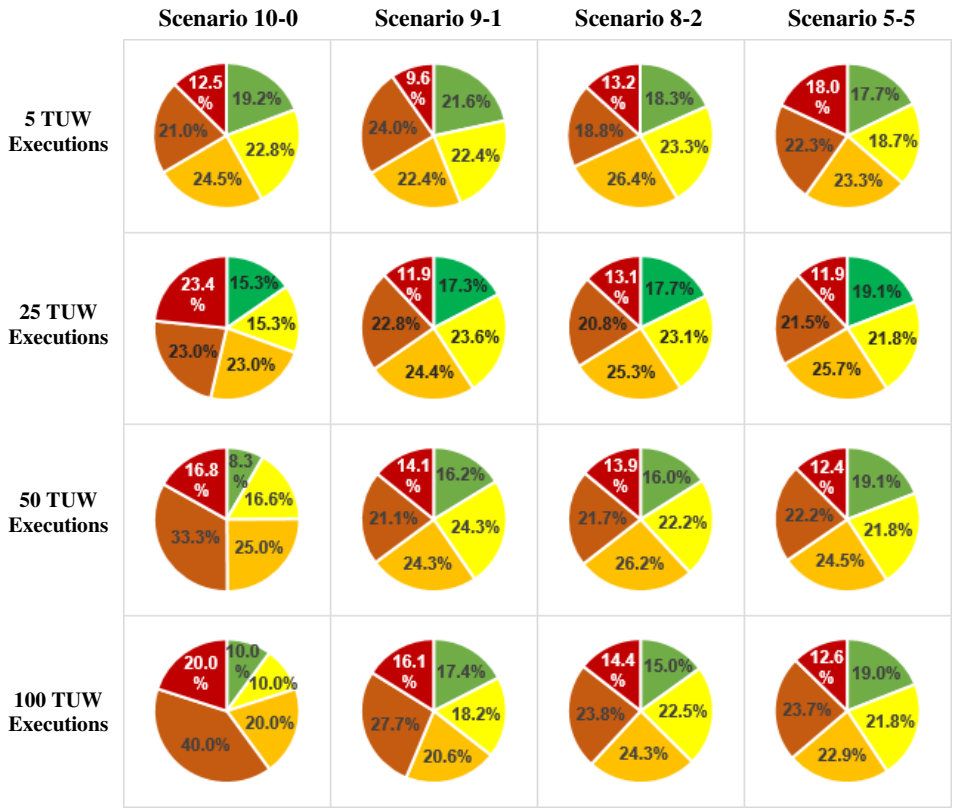


Figure 6-8. Percentage of Alerts per Risk Exposure Method in each Setup

It can be seen that the alerts that are potentially most critical to the enterprise (assuming this to be *Very High* + *Critical*) are approximately 35-40% of the total number of alerts in most cases. This gives a measure of how many alerts (60-65%) can be left to check subsequently to those that are most urgent to check.

To analyze the efficiency of the risk exposure alert ranking method, we recalculated the TPR, FPR, Accuracy, Precision and F-score measures for DW-DIDS, considering only the generated alerts referring to attacks that fell within the *High*, *Very High* and *Critical* measures, *i.e.* filtering those which present a greater threat to the enterprise. Figures 6-9 to 6-11 show these results.

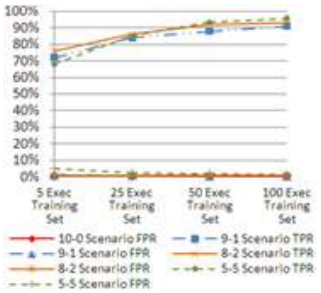


Figure 6-9. DW-DIDS TPR and FPR considering only High, Very High and Critical Risk Exposure Alerts

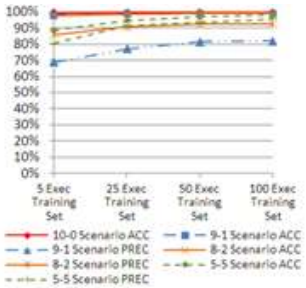


Figure 6-10. DW-DIDS Accuracy and Precision considering only High, Very High and Critical Risk Exposure Alerts

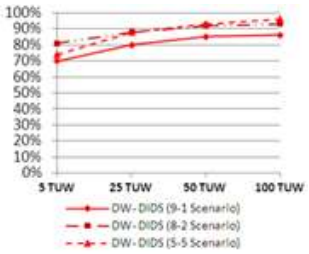


Figure 6-11. DW-DIDS F-Score considering only High, Very High and Critical Risk Exposure Alerts

Considering Figure 6-9, the TP rate presents nearly the same results as when all alerts are considered (Figure 6-5a), but the FP rate is much better than the previous, obtaining much fewer false alarms. The measured accuracy and precision, shown in Figure 6-10, is very high and also significantly better than the previous (Figure 6-6a). In fact, the accuracy for the majority of the scenarios in almost 100%, while in many setups the precision rises above 90%. Figure 6-11 shows that the overall F-score measure translates this, by presenting almost 10% of improvement for each scenario considering the previous results shown in Figures 6-7a to 6-7c.

Conclusively, this allows to state that considering the alerts referring to higher risk exposure present higher efficiency results in intrusion

detection, thus demonstrating that the risk exposure method is an adequate form of defining the priority on which alerts should be checked first and consequently reduce intrusion damage.

6.6.4. Database Response Time Overhead due to Intrusion Detection

In what concerns the impact on database performance, *i.e.* the increase of query response time, we measured an average overhead for each DIDS in each scenario as shown in Figure 6-12.

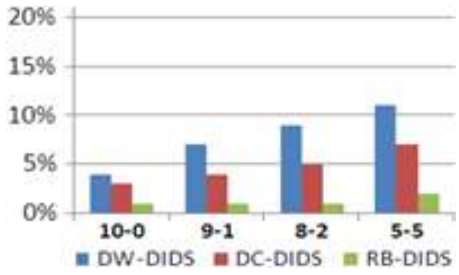


Figure 6-12. Database Response Time Overhead for each DIDS in each Scenario

By observing the previous figure, it can be seen that RB-DIDS is the fastest, introducing an overhead of equal or lesser than 2% to user query workload response time, while DW-DIDS is the slowest, given that it joins data-centric and command-centric analysis and processes a significantly higher amount of data than the remaining DIDS in the intrusion detection process, introducing response time overheads ranging from 4% to 11%. However, although DW-DIDS does in fact have the worse results, we argue that its intrusion detection efficiency shown in the experiments make these overheads worthwhile when compared to the remaining solutions.

6.7. Discussion on DW-DIDS

In DW-DIDS, all risk exposure, probability and impact rules are stored in the *DW-DIDS Database* and used by the *Intrusion Response Manager (IRM)* as formerly explained. Although probability is initially predefined, each rule may be redefined by the *DW Security Administrator* at any time for fine tuning. For instance, the *DW Security Administrator* may grant a different probability to any feature or grant higher or lower weights to specific

features that s/he knows are most likely to lead to better or less reliable ID rates given the DW's context.

The conditional clauses in the DW-DIDS rules (similarly to SQL clauses) allow an extremely wide range of definitions that, due to space feasibility issues, are not included. We just want to make clear that, besides the examples described in the former subsections, the algorithms can be easily adapted to cope with a wide range of rule possibilities using standard SQL functions with the DW-DIDS features, tables and columns, and the DW's tables and columns, providing a very wide intrusion detection scope coverage.

Using qualitative measures instead of quantitative measures allows providing a much more comprehensive rank; it is humanly much more intuitive and straightforward to interpret a *High* or *Low* measure of evaluation than the difference between a value of 0.46 and 0.58, or having just a *High* measure instead of differencing values such as 0.76 and 0.78. The qualitative measures smoothen the ranges of the quantitative values, providing better understanding to security staff.

Combining quantitative probability and impact assessments into a unique qualitative risk exposure measure also improves the efficiency of alert management. For example, if an alert refers to an attack with *Low* probability – probably, a false positive – or refers to an attack with *Low* impact – probably, against non-sensitive data – it can be assessed as having *Low* risk exposure, which means that checking it can be postponed (or the intrusion may even be tolerated); if another alert with higher risk exposure – and thus, probably capable of causing greater damage – is generated simultaneously, it is more significant and quickly dealt with. The credibility and assertiveness of these assumptions are demonstrated by the experimental results shown in Figures 6-9, 6-10 and 6-11 described in Subsection 6.6.3, where the analysis containing the most relevant alerts (*i.e. High*, *Very High* and *Critical* risk exposure) shows particularly good accuracy, precision and F-score results.

Figure 6-12 illustrates the alert correlation and risk exposure approaches for alert management. Standard alert correlation techniques are a weakness in most existing DIDS because they may exclude part of the generated alerts and do not consider the impact of the user actions, while our approach

considers all the alerts, focusing on their importance rather than solely on their probability.

The alert correlation approach might lead to wasting time dealing with an attack on unimportant data while an attack on vital data occurs. With the risk exposure alert ranking method proposed in this thesis, it is guaranteed that the attacks focusing on the most sensitive data or capable of producing more damage to the enterprise can be dealt with first, effectively increasing damage containment. Furthermore, while alert correlation may exclude some alert that refers to an intrusion potentially capable of producing high impact on the enterprise, the risk exposure does not exclude any alert, but rather ranks them given their respectively assessed risk measure.

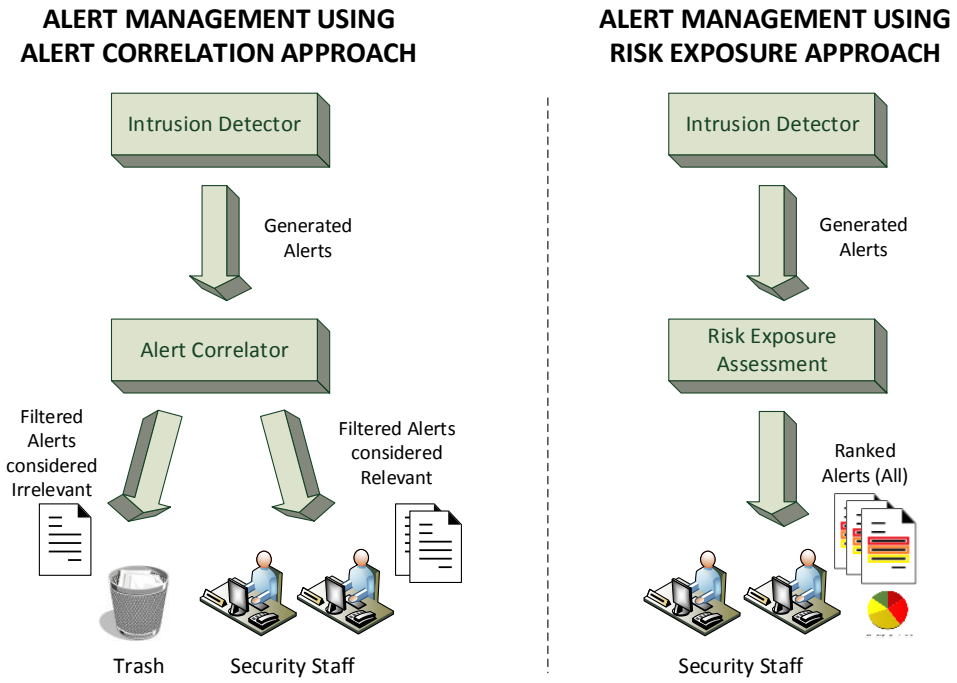


Figure 6-12. Risk Exposure Approach vs Alert Correlation for Alert Management

Although discussable, we argue that the contribution of each feature to the overall intrusion detection efficiency is subjective. The rules that define attack probability depend on the intrusion detection features in DW-DIDS are initially tuned to 0.5 by default, given the system has no knowledge *a priori* on which feature is more relevant for the intrusion detection process.

However, if the *DW Security Administrator* has know-how or any way of defining the relevance of each feature *a priori*, the rules provide a way to accomplish this by adding whichever extra rules s/he wishes to the rule base. On the other hand, the rules that define the sensitivity of data (*i.e.* impact rules) must be defined by the *DW Security Administrator* because it depends on the nature and importance of that data to the enterprise, which only s/he (and mainly business managers) know, and depends on the DW context itself. Therefore, there isn't any automatic setup for these rules because, from our point of view, it is not relevant.

Our proposal is both syntax-centric and data-centric. Although this rises its execution time overhead, we argue that this is worthwhile because it allows our approach to analyze the complete set of dimensions affecting the data due to the user action – the command itself, the processed data and the data resulting from the command's execution – which is left out by the IDS used for comparison in the experiments (they only analyze command syntax - RBAC - and resulting dataset - Data-Centric). To the best of our knowledge, no other DIDS proposes this threefold analysis.

The main reasons why we chose the role-based and data-centered approaches proposed in [Kamra *et al.*, 2008; Mathew, 2010] is that DIDS analyzing data access patterns such as [Bertino *et al.*, 2005; Kamra *et al.*, 2010] and analyzing the targeted data such as [Mathew *et al.*, 2010; Spalka and Lehnhardt, 2005] seem more adequate for DW intrusion detection than solutions using other techniques such as sequence alignment, fingerprinting commands or transactional read-write rules, as we previously discussed in Chapter 2. Therefore, we chose one of each type of these intrusion detection techniques.

The differences in storage size and time cost are justified by the type of dataset required by each DIDS to build the profiles: RBAC-DIDS just parses the SQL command and splits it into the relevant features, which basically works by accessing the command log and executing string manipulation; DC-DIDS considers, on average, a higher number of features than RBAC-DIDS and executes statistics per feature on each resulting command's dataset, thus requiring data access actions, which are much more time-expensive than those executed by RBAC-DIDS; and DW-DIDS executes both types of actions of RBAC-DIDS and DC-DIDS, plus accessing the data rows processed by the command, and has the highest number of features.

Although this makes DW-DIDS the slowest solution in building the profiles and the one that requires the highest amount of storage space, collecting and combining the information regarding the user command with the resulting dataset and the rows processed by the command enables it to compose the richest feature dataset, which would add value to improve its intrusion efficiency, as was demonstrated in the experiments.

By analyzing all results, it may be concluded that DW-DIDS showed the best results, followed closely by RB-DIDS in most scenarios, mainly when the training set was significantly large (>=25 TUW), while DC-DIDS obtained the worst results. By integrating features that enable both data-centric and command-centric analysis, DW-DIDS is capable of producing the expected added value when compared with the application of those distinct analysis in separate. We may also conclude that a training set of 5 TUW is insufficient in size for producing an efficient user profile database, as these scenarios yielded relatively low intrusion detection efficiency. The better results were obtained using the highest number of user workloads in the training stage.

The results presented in the experiments suffer from the predefined data values and user commands used in the setups. Although both the DW-DIDS and RBAC-based approaches obtained good results in our experiments, it is extremely difficult to state that these results can be generalized to assess the efficiency of both DIDS. Most DIDS use the well-known KDD99 benchmark [DARPA] to compare results. However, this benchmark uses network-based traffic for its purpose, which in our case is not applicable. In fact, given the absence of an SQL-based intrusion detection benchmark, the results published in this field of research are not comparable and thus, they cannot be generalized. We therefore argue that research in both the data warehousing and intrusion detection communities should make an effort to propose a benchmark for DIDS at the SQL level, possibly a compromise between the well-known TPC-DS or TPC-H decision support benchmarks and the KDD99 benchmark.

6.8. Summary

In this chapter, we proposed a DIDS specifically designed for DWs, which can work transparently between the user interfaces and the database server as an extension of the DBMS itself. User behavior profiles are built using

features that enable analyzing the diverse dimensions of DBMS user behavior: SQL commands, processed data and result datasets. Statistical tests are used to verify user actions against those profiles and generate intrusion alerts.

The probability of each alert referring to a true intrusion and the impact that might be caused by the user action to which the alert refers can be managed by a set of SQL-like rules previously defined by the DW Security Administrator. This rule-base allows extending DBMS data access policies and provides a mean to assess the risk exposure of each intruder action for an extremely wide range of possibilities. The risk exposure method is used to rank the generated alerts and prioritize response to intrusions, presenting clear advantages when compared to standard correlation techniques: it does not allow any intrusion alert to be neglected and it enables rapidly responding to alerts which may cause greater damage to the enterprise. The experimental results show the proposed approach achieves high intrusion detection efficiency and accuracy results in the tested setups.

Chapter 7

Conclusions and Future Work

Protecting business secrets from disclosure is a critical issue for many enterprises. This implies that ensuring data confidentiality in extremely sensitive data repositories such as DWs, which store many of those secrets, is of vital importance. To deal with this, many data security solutions have been proposed in the past. Research and best practice guides have stated that the best way to promote confidentiality at the database level is probably to use a mix of DIDS together with encryption for live user databases, and use data masking techniques for protecting sensitive published or outsourced data.

Despite the development of these solutions for protecting data confidentiality, internal as well as external attacks against databases in the recent past have been rising in both number and complexity. This makes the continuous development and improvement of data security solutions an imposing business requirement, in which this thesis seeks to make a contribution. In this sense, this thesis addressed the feasibility issues involving solutions that promote data confidentiality and deal with intrusions against DWs at the database level, focusing on data masking, encryption and DIDS.

As discussed, data masking solutions are typically not used to protect live databases because they are not considered secure enough, and have been mostly applied as an irreversible process as a mean to secure sensitive data that has to be outsourced or publicly published. On the other hand, it is revealed throughout this thesis that the database performance overheads introduced by encryption techniques might effectively lead business stakeholders and end users to consider their use infeasible in many data warehousing environments. Finally, the reasons why there should be DIDS specifically tailored for data warehousing environments have also been discussed, as well as the issues relating alert management and dealing with

intrusions against DWs according to the potential cost they represent to the business.

Founded on the research and analysis of current commercial and state-of-the-art data masking and encryption solutions as well as database intrusion detection techniques, the overall objective of this thesis was the proposal of new feasible, efficient and effective solutions in these fields that contribute to enhance data security in data warehousing environments. To achieve this overall objective given the importance of securing confidentiality in DWs and comparing with the currently available data security solutions from the fields covered, our work introduces a series of solid key contributions, which are detailed in the following paragraphs:

- **A body of knowledge focusing on the impact on database performance caused by the use of encryption in very large databases.** Most discussions around the development of new encryption techniques are focused on their security proof, *i.e.*, on the demonstration of how secure they are against attackers. The focus on their performance, *i.e.*, how fast they are able to execute, is often considered a secondary issue. We have built a body of knowledge focusing on the development guidelines of modern encryption solutions and their performance concerning implementations to be used against very large databases. Experimental evaluations included in state-of-the-art standards and published research as well as experimental results throughout this thesis effectively show that the storage space and response time overheads introduced by encryption algorithms dramatically degrade database performance to a magnitude that jeopardizes their feasibility in data warehousing environments. Since database performance is a critical issue in DWs, we conclude that current encryption solutions are not suitable. Data warehouses operate in a well-determined specific environment with tight security, performance and scalability requirements and, therefore, need specific solutions able to cope with these directives. Since there is always a tradeoff between security strength and performance, developing specific data confidentiality solutions for DWs must always balance security requirements with the desire for high performance, *i.e.*, ensuring a strong level of security while keeping database performance acceptable. This is a critical issue that

justifies the development of new solutions in this domain, given the lack of specific solutions for data warehousing environments.

- **A body of knowledge on database intrusion detection techniques.** Although intrusion detection has been well studied in the past decades, it has mostly focused on network and operating system level intrusions rather than on the data level intrusions. We have built a body of knowledge that gathers, describes and classifies the most recently proposed intrusion detection techniques that can be used at the data level to develop DIDS. We have discussed their usage from a data warehousing perspective, given the typical DW workloads. We have justified why DWs are database systems with unique user and data processing requirements that differ from other types of systems and require distinctively tailored intrusion detection approaches. To the best of our knowledge, we have concluded that up to date there has been no database intrusion detection proposal that accounts for: 1) the impact that the intrusion might cause to the business; 2) realizing intrusion detection and response both *a priori* and *a posteriori* to the execution of the user action; and 3) performing intrusion detection by analyzing the user action, processed data and the outcome of processing the user action, together in the same workflow. We have also discussed why alert correlation techniques are not the most appropriate solutions for performing alert management, given that these techniques exclude possible intrusions that could be alerted, by relying solely on probability assessments. Given the sensitivity of DW data and its critical security requirements, these facts justify the development of new DIDS that incorporate these capabilities.
- **An integrated data security framework that enables the use of data masking, encryption and intrusion detection in a single workflow.** To the best of our knowledge, this is the first framework that transversally integrates a diversity of solutions across several distinct security domains/purposes such as masking, encryption and intrusion detection. The proposed framework describes the implementation of an architecture that enables integrating all solutions proposed in this thesis together in a unique workflow. The framework also proposes the guidelines for improving or developing

new data masking, encryption and DIDS from a data warehousing perspective, considering the issues pointed out by the discussion derived from the bodies of knowledge in each domain presented in Chapter 2. This framework provides an overall functional security architecture and guides the development of the solutions proposed in this thesis for each referred domain.

- **A reversible data masking technique for numeric values on live databases using only standard SQL operators.** Although data masking techniques are not seen as reliable solutions to be used in live sensitive databases and are mostly used as an irreversible process which is applied to the data that is to be publicly available or outsourced, we have shown that they might still be a viable option in data warehousing environments in which response time is a critical concern. Given the overhead introduced by using encryption, using a lightweight data masking solution that provides some security strength is better than not having any sort of security at all. In this thesis, we have proposed a reversible data masking technique, which provides a certain level of security strength while producing low database performance overheads. It relies on data type preservation and restrains its data transformations to operators existing in standard SQL, requiring only SQL rewriting to achieve its security purpose. This gives it several advantages: 1) data type preservation avoids database storage space overhead and extra computational efforts in datatype conversions when compared with standard encryption; 2) executing SQL commands directly against the masked data; 3) due to the previous advantage, it avoids data roundtrips between the database and the masking/unmasking mechanisms, thus avoiding critical path I/O and network bandwidth consumption bottlenecks, contrarily to other solutions which require this; 4) data-at-rest is masked at all times; 5) It executes faster than standard and state-of-the-art encryption algorithms; and 6) the solution can be transversally and transparently applied and used in any DBMS against any database. The experimental results have confirmed these advantages and demonstrated that it can effectively be a valid way to protect data confidentiality in DWs.

- **A lightweight encryption algorithm for securing numeric values using only standard SQL operators.** We have proposed a novel encryption algorithm that, although might not be as secure as other standard and state-of-the-art encryption algorithms, presents significantly better database performance while providing considerable security strength, i.e., better performance-security tradeoffs. It follows similar guidelines as those on which the data masking technique was based, also relying on data type preservation and restraining its data transformations to operators existing in standard SQL, requiring only SQL rewriting to achieve its security purpose. Thus, it also achieves the same advantages, when compared with standard and state-of-the-art encryption algorithms. The experimental results have also confirmed these advantages and the included security proof makes it an acceptable alternative to the former, making it a feasible and efficient encryption option to protect data confidentiality in DWs.
- **A DIDS focused on typical end user workloads and intrusions in DWs**, capable of analyzing the user action, processed data and resulting outcome from the execution of the user action, performing intrusion detection and response both *a priori* and *a posteriori*, and a risk exposure method for ranking alerts and responding to possible intrusions in a much more reliable and efficient way than standard alert correlation techniques. Our DIDS specifically accounts for the characteristics of DW users, gathering the set of features that allow adequately building their behavior profiles and analyze their actions. The proposed features handle intrusion detection by analyzing from several aspects of user workloads, such as the user command, the data processed by the command and the results of its processing. The intrusion detection processes may run before the command's execution and after it finishes executing (but before disclosing results back to the user). Each generated intrusion alert is never discarded, but ranked by a risk exposure method that is able to prioritize dealing with the intrusions that potentially present a higher threat to the business. The proposed set of risk exposure rules (including probability and impact) enables defining a particularly large scope of possibilities that provide a wide coverage of intrusions. The relevance of each feature in the intrusion detection processes is

adjusted according to its efficiency given its TP and FP rates and self-calibrates through time. Therefore, the proposed solution is effectively better than those that perform intrusion detection in only one of the mentioned moments or only on one aspect of the user action, and particularly better than those that rely on alert correlation techniques for alert management purposes. The experimental results demonstrate its efficiency against similar state-of-the-art intrusion detection solutions, comproving these statements.

Future Work

The work presented in this thesis represents the initial ground for our research in data security for data warehousing. Related to the issues and questions addressed in this thesis, we propose the following priority developments and improvements:

- **Increase the scope of both data masking and encryption techniques to consider protecting the confidentiality of textual attributes, besides numerical attributes.** Both data masking and encryption techniques proposed in this thesis were specifically designed as intended to mask and encrypt numeric values, because in most DWs the main portion of sensitive data is numerical. Nevertheless, other datatypes may also be used to store sensitive data. A natural and logical improvement of the proposed solutions is its adaptation to be able to accomplish protecting data of all datatypes. Therefore, researching the best ways to develop and implement these improvements, and verify their feasibility, namely by assessing performance impact as well as security strength, is one of the future works to be executed.
- **Investigate ways to enhance the security strength of the proposed data masking and encryption solutions, without losing focus on their feasibility for data warehousing environments.** As we have discussed in this thesis, the execution performance and security strength of both data masking and encryption techniques depend on their algorithm, keys and block length. Investigating changes to the proposed data masking formula or encryption algorithm in any one of these aspects to improve their performance or their security and

respective tradeoffs is always an open research possibility for future work, as in any other similar solution. Additionally, any of the proposed solutions can use the row masking keys to enable a method for injecting false rows into the fact tables. This would make it increasingly difficult to distinguish true and false data, increasing the overall DW security level and misleading attackers that gain direct access to the database. To achieve this, instead of generating independent random numbers for the values of the masking or encryption row keys in each fact table row j, we redefine those keys K_j 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:

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

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

Given R = $K_{3,j}$ MOD ($\sum C_{i,j}$) , $\{ i = 1 \ldots n \}$
IF R=0 THEN row j is True ELSE row j is False

However, although potentially increasing the fact table's security strength, there is a tradeoff between security and performance that needs to be considered 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. Thus, the injection of false data to increase security strength is at least, arguable, since it increases the amount of data to be accessed when the queries are being processed, consequently introducing overhead in response time.

- **Investigate ways to enhance the efficiency and effectiveness of the intrusion detection methods.** Our intrusion detection approach appears to work best when the number of intrusions is relatively low. This happens because its statistical probability provides that a greater number of false alarms is likely to be generated given an increasing number of attack attempts. However, the statistical approach used to detect abnormal commands was our first approach. As future work, testing techniques such as the Naive Bayes Classifier, Clustering, SVM, etc. for the intrusion detection process should be approached and their efficiency should be compared in order to choose the most efficient solution(s).
- **Improve the practical application and performance of the risk exposure method rules.** The execution of the verification tasks referring to the impact and probability rules introduce extra response time because they need to be processed before the user command is executed and before the results are disclosed back to the user. Given the expressiveness of the rules' syntax (similar to SQL), the efforts in processing them may be significant. Therefore, the impact produced in database performance by the referred verification tasks for the generated alerts should be thoroughly evaluated and analyzed, and ways of improving and optimizing the execution of these tasks should be researched.
- **Develop a database intrusion detection benchmark.** Benchmarks are an essential instrument used in the development and implementation of many systems. They provide a mean to test those systems and significantly contribute to supply end users and developers with feedback on their performance, allowing to compare between different solutions, as well as give the developers insight for improving the proposed solutions. In the past, the KDD99 benchmark [KDD99] has been widely used for testing intrusion detection solutions, as we have previously mentioned. However, this benchmark focuses on intrusion actions at the network and operating system (OS) level, and the datasets and attack loads used in most published research are either synthetic or come from real-world applications. To the best of our knowledge, there has been no proposal from the research community regarding an intrusion

detection benchmark focusing on the data level. In such a sensitive matter as data security, we find that the inexistence of a recognized standard database intrusion detection benchmark at the data level is an important lack in assessing the feasibility, credibility and efficiency of DIDS. Therefore, we propose a first draft version of such a benchmark, which can be seen in Appendix D of this thesis.

- **Realize and produce a survey with an objective comparison between distinct state-of-the-art database intrusion detection techniques and mechanisms using the proposed database intrusion detection benchmark.** Once the benchmark is defined and accepted by the database research and security communities, use it to test a sample of distinct state-of-the-art intrusion detection techniques (*e.g.* those described in Chapter 2). The obtained results can then be used to produce a formal report to disclose them to those communities and drive discussion around them as well as around the benchmark itself.
- **Demonstrate the feasibility, efficiency and effectiveness of the proposed solutions in real-world data warehousing contexts.** Perform implementations and tools using the proposed solutions in real-world DWs and gather feedback to measure and analyze their accomplishments in order to assess their feasibility, efficiency and effectiveness in real-world data warehousing contexts.

In conclusion, this thesis has focused on proposing feasible, efficient and effective techniques that can enhance data security in data warehousing environments. Overall, the main objective for the future is to investigate ways of enhancing these proposals and go from research prototypes and laboratory environments to real-world scenarios as much as possible. We will aim verifying our experimental results and expectations and to provide both the research community as well as the industrial community with knowledge and tools that can truly enhance data security in data warehousing environments. We also wish that our work can make way for innovative solutions in this domain, not only data masking, encryption and intrusion detection techniques specifically designed for DWs, but also for the conception of a novel standard database intrusion detection benchmark

at the database level. Ultimately, we hope our work is considered as an effective concrete valid contribution to keep the secrets of the business safe.

References

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.

AES (2001), Advanced Encryption Standard, *National Bureau of Standards, National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS)*-197.

Agrawal, R., Kiernan, J., Srikant, R. and Y. Xu (2004), "Order-Preserving Encryption for Numeric Data", *ACM SIG International Conference on Management Of Data (SIGMOD).*

Avizienis, A., Laprie, J., Randell, B. and C. Landwehr (2004), Basic Concepts and Taxonomy of Dependable and Secure Computing, *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Vol. 1, No. 1, January-March.

Baer, H. (2004), On-Time Data Warehousing with Oracle Database 10g – Information at the Speed of Your Business, *Oracle White Paper, Oracle Corporation.*

Barker, E., Barker, W., Burr, W., Polk, W. and M. Smid (2012), *Recommendation for Key Management*, Special Publication, National Institute of Standards and Technology (NIST), available at http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf .

Bartle, R. G. (1976), *The Elements of Real Analysis*, 2nd Edition, John Wiley & Sons.

Bernstein, D. J. (2005), *Snuffle 2005: The Salsa Encryption Function*, 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.

Bertino, E. and R. Sandhu (2005a), Database Security – Concepts, Approaches and Challenges, *IEEE Transactions on Dependable and Secure Computing (TDSC)*, Vol. 2, No. 1.

Bertino, E., Leggieri, T. and E. Terzi (2005b), "Intrusion Detection in RBAC-Administered Databases", *Annual Computer Security Applications Conference (AC-SAC)*.

Bockermann, C., Apel, M. and M. Meier (2009), "Learning SQL for Database Intrusion Detection using Context-Sensitive Modeling", *International Conference on Knowledge Discovery and Machine Learning (KDML)*.

Castro, V. G. (2009), *The Use of Alternative Data Models in Data Warehousing Environments*, PhD dissertation thesis, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK.

Chaudhuri, S. and U. Dayal (1997), An Overview of Data Warehousing and OLAP technology, *SIGMOD Record*, 26(1), September.

Chung, C. Y., Gertz, M. and K. Levitt (1999), "DEMIDS: A Misuse Detection System for Database Systems", *IFIP TC11 WG11.5 Conference on Integrity and Internal Control in Information Systems*, Kluwer Academic Publishers.

DARPA archive, *Task Description of the KDD99 Benchmark*, http://www.kdd.ics.uci.edu/databases/kddcup99/task.html.

Debar, H. and A. Wespi (2001), "Aggregation and Correlation of Intrusion-Detection Alerts", *International Conference on Recent Advances in Intrusion Detection (RAID)*.

DES (1977), Data Encryption Standard, *National Bureau of Standards, National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS)* Pub 46, 1977.

Devlin, B. A. and P. T. Murphy (1988), An Architecture for a Business and Information System, *IBM Systems Journal*, Vol. 27, No. 1, USA.

Douligeris, C. and A. Mitrokotsa (2004), DDoS Attacks and Defense Mechanisms: Classification and State-of-the-Art, *International Journal of Computer Networks (IJCN)*, Elsevier B. V., 44.

Elminaam, D., Kader, H. and M. Hadhoud (2010), Evaluating the Performance of Symmetric Encryption Algorithms, *International Journal of Network Security*, Vol. 10, No. 3.

Fonseca, J., Vieira, M. and H. Madeira (2008), "Online Detection of Malicious Data Access using DBMS Auditing", *ACM International Symposium on Applied Computing (SAC)*.

Gartner Inc. (2009), *Selection Criteria for Data Masking Technologies*, Research Report ID G00165388, February.

Ge, T. and S. Zdonik (2007), "Fast, Secure Encryption for Indexing in a Column-Oriented DBMS", *International Conference on Data Engineering (ICDE)*.

Hacigumus, H., Iyer, B. R., Li, C. and S. Mehrotra (2002), "Executing SQL over Encrypted Data in the Database-Service-Provider Model", *ACM SIG International Conference on Management Of Data (SIGMOD)*.

Hacigumus, H., Iyer, B. R. and S. Mehrotra (2004), "Efficient Execution of Aggregation Queries over Encrypted Relational Databases", *International Conference on Databases Systems for Advanced Applications (DASFAA)*.

Halfond, W., Viegas, J. and A. Orso (2006), "A Classification of SQL Injection Attacks and Prevention Techniques", *International Symposium on Secure Software Engineering (SSE)*.

Hu, Y. and B. Panda (2004), "A Data Mining Approach for Database Intrusion Detection", *International Symposium on Applied Computing (SAC)*.

Huey, P. (2008), Oracle Database Security Guide 11g, *Oracle Corporation*.

Inmon, W. H. (1996), *Building the Data Warehouse*, 2nd Edition, John Wiley & Sons, Inc.

Inmon, W. H. (2002), *Building the Data Warehouse*, 3rd Edition, John Wiley & Sons, Inc.

Jabbour, G. and D. Menasce (2009), "The Insider Threat Security Architecture: A Framework for an Integrated, Inseparable, and

Uninterrupted Self-Protection Mechanism", *International Conference on Computational Science and Engineering (ICCSE)*.

Kamra, A., Terzi, E. and E. Bertino (2008), Detecting Anomalous Access Patterns in Relational Databases, *Springer VLDB Journal* (17).

Kamra, A. (2010), *Mechanisms for Database Intrusion Detection and Response*, PhD dissertation thesis, Purdue University, USA, August.

Kim, J. (2011), "Injection Attack Detection Using the Removal of SQL Query Attribute Values", *International Conference on Information Science and Applications (ICISA)*.

Kim, J., Lee, Y. and S. Lee (2010), DES with any reduced masked rounds is not secure against side-channel attacks, *International Journal of Computers and Mathematics with Applications*, 60.

Kimball, R. (1996), *The Data Warehouse Toolkit, 1st Edition*, Wiley & Sons, Inc.

Kimball, R. and M. Ross (2002), *The Data Warehouse Toolkit, 2nd Edition*, Wiley & Sons, Inc.

Kimball, R. and M. Ross (2013), *The Data Warehouse Toolkit, 3rd Edition*, Wiley & Sons, Inc.

Kindy, D. A. and A. K. Pathan (2012), *A Detailed Survey on Various Aspects of SQL Injection: Vulnerabilities*, Innovative Attacks and Remedies, Computing Research Repository (CoRR), Cornell University, USA.

Kobielus, J. (2009), The Forrester Wave: Enterprise Data Warehousing Platforms, *Forrester Research Report*, Q1.

Kundu, A., Sural, S. and A. K. Majumdar (2010), Database Intrusion Detection Using Sequence Alignment, *International Journal of Information Security* (9).

Lappas, T., and K. Pelechrinis (2007), *Data Mining Techniques for (Network) Intrusion Detection Systems*, Technical Report, Department of Computer Science and Engineering, University of California, Riverside.

Lee, S. Y., Low W. L. and P. Y. Wong (2002), "Learning Fingerprints for a Database Intrusion Detection System", *European Symposium on Research in Computer Security (ESORICS)*.

Lee, V. C. S., Stankovic, J. A. and S. H. Son (2000), "Intrusion Detection in Real-time Database Systems via Time Signatures", *Real-time Technology and Applications Symposium (RTAS)*.

Lee, W. (2002), Applying Data Mining to Intrusion Detection: the Quest for Automation, Efficiency, and Credibility, *SIGKDD Explorations*, Vol. 4, Issue 2.

Lee, W. and D. Xiang (2001), "Information-Theoretic Measures for Anomaly Detection", *IEEE Symposium on Security and Privacy (S&P)*.

Marsland, S. (2011), *Machine Learning*, CRC Press, §4.1.1, 2011.

Mathew, S., Petropoulos, M., Ngo, H. Q. and S. Upadhyaya (2010), "A Data-Centric Approach to Insider Attack Detection in Database Systems", *International Conference on Recent Advances in Intrusion Detection (RAID)*.

Matsumoto, M. and T. Nishimura (1998). "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", *ACM Transactions on Modeling and Computer Simulation* **8** (1): 3–30.

Mattson, U. T. (2004), *Database Encryption – How to Balance Security with Performance*, Protegrity Corporation Technical Paper.

McKendrick, J. (2012), IOUG Enterprise Data Security Survey 2012: Closing the Security Gap, *The Independent Oracle Users Group (IOUG) Security Report*, November.

Mogull, R. (2006), *Top Five Steps to Prevent Data Loss and Information Leaks*, Gartner Research Report.

Motwani, R., Nabar, S. U. and D. Thomas (2004), "Auditing SQL Queries", *International Conference on Data Engineering (ICDE)*.

Nadeem, A. and M. Y. Javed (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.

Newman, A. C. (2011), *Intrusion Detection and Security Auditing in Oracle*, Application Security Inc. White Paper.

Nicolett, M., and J. Wheatman (2007), *DAM Technology Provides Monitoring and Analytics with Less Overhead*, Gartner Research Report.

Ning, P., Cui, Y. and D.S. Reeves (2002), "Analyzing Intensive Intrusion Alerts via Correlation", *International Conference on Recent Advances in Intrusion Detection (RAID)*.

Oracle Corporation (2005), Security and the Data Warehouse, *Oracle White Paper*, April.

Oracle Corporation (2010a), *Oracle Advanced Security Transparent Data Encryption Best Practices*, Oracle White Paper, July.

Oracle Corporation (2010b), *Oracle Real Application Clusters (RAC)*, http://www.oracle.com/us/products/database/options/real-application-clusters/index.htm, September.

Oracle Corporation (2010c), *Data Masking Best Practices*, Oracle White Paper.

Ponniah, P (2010), *Data Warehouse Fundamentals for IT Professionals, 2nd Edition*, Wiley & Sons, Inc.

Pei, J., Upadhyaya, S. J., Farooq, F. and V. Govindaraju (2004), "Data Mining for Intrusion Detection", Keynote in *International Conference on Data Engineering (ICDE)*.

Pietraszek, T. (2004), "Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection", *International Conference on Recent Advances in Intrusion Detection (RAID)*.

Pietraszek, T., and A. Tanner (2005), Data Mining and Machine Learning – Towards Reducing False Positives in Intrusion Detection, *Information Security Technical Report*, 10(3).

Pietraszek, T. (2006), *Alert Classification to Reduce False Positives in Intrusion Detection*, PhD dissertation thesis, University of Freiburg, Germany, July.

Radha, V. and N. H. Kumar (2005), "EISA – An Enterprise Application Security Solution for Databases", *International Conference on Information Systems Security (ICISS)*.

Ravikumar, G. K., Manjunath, T. N., Ravindra, S. H. and I. M. Umesh (2011), A Survey on Recent Trends, Process and Development in Data Masking for Testing, *International Journal of Computer Science Issues (IJCSI)*, Vol. 8, Issue 2.

Santos, R. J., Bernardino, J. and M. Vieira (2011a), "A Survey on Data Security in Data Warehousing", *International Conference on Computer as a Tool (EUROCON)*.

Santos, R. J., Bernardino, J. and M. Vieira (2011b), "A Data Masking Technique for Data Warehouses", *International Database Engineering & Applications Symposium (IDEAS)*.

Santos, R. J., Bernardino, J. and M. Vieira (2011c), "Balancing Security and Performance for Enhancing Data Privacy in Data Warehouses", in *IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM)*.

Santos, R. J., Bernardino, J. and M. Vieira (2012a), "Evaluating the Feasibility Issues of Data Confidentiality Solutions from a Data Warehousing Perspective", *International Conference on Data Warehousing and Knowledge Discovery (DAWAK)*.

Santos, R. J., Bernardino, J. and M. Vieira (2012b), "DBMS Application Layer Intrusion Detection for Data Warehouses", *International Conference on Information Systems Development (ISD)*.

Santos, R. J., Bernardino, J., Vieira, M. and D. Rasteiro (2012c), "Securing Data Warehouses from Web-based Intrusions", *International Conference on Web Information Systems Engineering (WISE)*.

Santos, R. J., Rasteiro, D. M. L., Bernardino, J. and M. Vieira (2013), "A Specific Encryption Solution for Data Warehouses", *International Conference on Databases Systems for Advanced Applications (DASFAA)*.

Scarfone, K. and P. Mell (2007), Guide to Intrusion Detection and Prevention Systems (IDPS), *Recommendations of the National Institute of Standards and Technology (NIST), Special Publication 800-94*.

Schneier, B. (2013), *The Blowfish Encryption Algorithm*, http://www.schneier.com/blowfish.html.

Schulman, A. (2007), *Top 10 Database Attacks*, The Chartered Institute for IT – Enabling the Information Society, http://www.bcs.org/content/ConWebDoc/8852.

Simitsis, A. (2005), *Modelling and Optimization of Extraction-Transformation-Loading (ETL) Processes in Data Warehouse Environments*, PhD dissertation thesis, School of Electrical and Computer Engineering, National Technical University of Athens, Greece.

Spalka, A. and J. Lehnhardt (2005), "A Comprehensive Approach to Anomaly Detection in Relational Databases", *IFIP International Conference on Data and Applications Security and Privacy (DBSec)*.

Srivastava, A., Sural, S. and A. K. Majumdar (2006), Database Intrusion Detection using Weighted Sequence Mining, *Journal of Computers*, Vol. I, No. 4.

TPC-H, Transaction Processing Performance Council, *The TPC Decision Support Benchmark H*, http://www.tpc.org/tpch/

TPC-H Specifications, Transaction Processing Performance Council, *The TPC Decision Support Benchmark H Standard Specifications review 2.16.0*, http://www.tpc.org/tpch/spec/tpch2.16.0.pdf

TPC-DS, Transaction Processing Performance Council, *The TPC Decision Support Benchmark*, http://www.tpc.org/tpcds/

Treinen, J. and R. Thurimella (2006), "A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures", *International Conference on Recent Advances in Intrusion Detection (RAID)*.

Tsunoo, Y., Saito, T., Kubo, H., Suzaki, T. and H. Nakashima (2007), Differential Cryptanalysis of Salsa20/8, in *Workshop Record of SASC*

2007: The State-of-the-art of Stream Ciphers, eSTREAM Report 2007/010.

Vaudenay, S. (2006), *A Classical Introduction to Cryptography – Applications for Communications Security*, Swiss Federal Institute of Technologies (EPFL), Springer Science+Business Media Inc.

Valdes, A. and K. Skinner (2001), "Probabilistic Alert Correlation", *International Conference on Recent Advances in Intrusion Detection (RAID)*.

Vimercati, S. C., Foresti, S., Jajodia, S., Paraboschi, S. and P. Samarati (2007), "Over-encryption: Management of Access Control Evolution on Oursourced Data", *International Conference on Very Large DataBases (VLDB)*.

Wheeler, D. and R. Needham (1995), *TEA, a Tiny Encryption Algorithm*, International Workshop on Fast Software Encryption, Springer Lecture Notes in Computer Science, Volume 1008, pp 363-366.

Yu, Z., Tsai, J. P. and T. Weigert (2007), An Automatically Tuning Intrusion Detection System, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 37, No. 2.

Yuhanna, N. (2009), Your Enterprise Database Security Strategy 2010, *Forrester Research*, September.

Zhong, Y. and X. Qin (2004), "Database Intrusion Detection based on User Query Frequent Itemsets Mining with Item Constraints", *International Information Security Conference (InfoSecu)*.

Appendix A

Sales Data Warehouse

In this appendix we describe the purpose and data schemas of the Sales DW as well as its scale and query workloads used in the experimental evaluations included in this thesis.

A.1. Purpose

The Sales DW is withdrawn from a real-world enterprise data mart of an online retail business, which aims on analyzing sales revenue, given customers, products and promotions.

A.2. Data Schema

The Sales DW data schema is shown in Figure A-1. It is a star schema with a central fact table named *Sales*, which stores the relevant measures regarding sales and promotions, and four dimension tables that describe the business, respectively containing the descriptive information concerning *Customers*, *Products* and *Promotions*, as well as a temporal dimension named as *Time*.

A.3. Table Scale Size

The number of rows and approximate storage space size for the Sales DW used in the experimental evaluations is shown in Table A-1, corresponding to one year of business activity.

Table A-1. Scale-size features of the Sales Data Warehouse

	Number of Rows	Storage Size
Time	8 760	0,12 MB
Customers	250 000	90 MB
Products	50 000	7 MB
Promotions	89 812	10 MB
Sales	31 536 000	1 927 MB

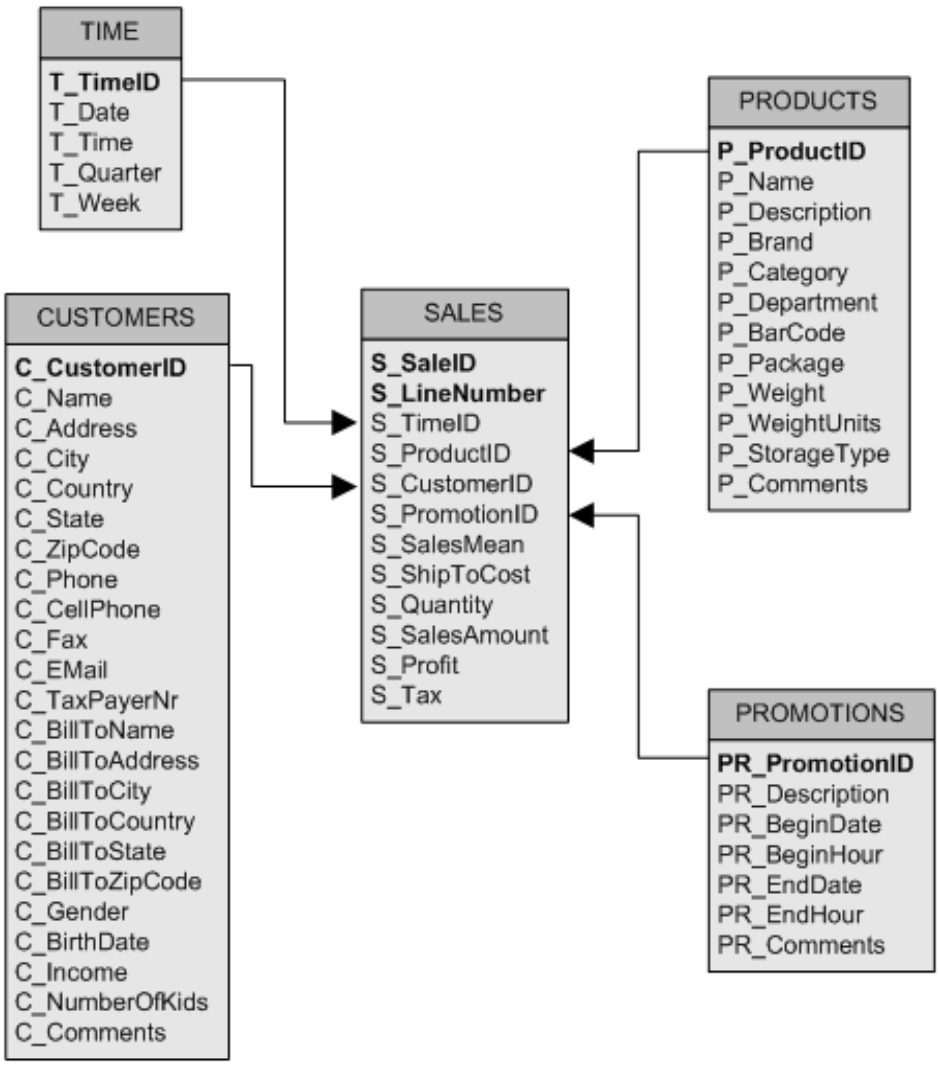


Figure A-1. Sales Data Warehouse Star Schema

A.4. Query Workloads

Following is the list of 29 queries against the Sales DW data schema that were used in the experiments.

Q1. YEAR SALES PROFITS QUOTA PER DEPARTMENT, ORDERED BY QUOTA

```
SELECT
   P_Department,
   Profit/TotalProfit*100 AS ProfitQuota
FROM
   (SELECT
       P_Department,
       SUM(S_profit) AS Profit
    FROM
       Products, Sales, Times
    WHERE
       S_ProductID=P_ProductID AND
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
       T_Date<=to_date('31-12-2008','DD-MM-YYYY')
    GROUP BY
       P_Department) A,
   (SELECT
       SUM(S_profit) AS TotalProfit
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
       T_Date<=to_date('31-12-2008','DD-MM-YYYY')) B
ORDER BY
   ProfitQuota DESC
```

Q2. MONTH SALES PROFITS QUOTA PER DEPARTMENT, ORDERED BY QUOTA

```
SELECT
   P_Department,
   Profit/TotalProfit*100 AS ProfitQuota
FROM
   (SELECT
       P_Department, SUM(S_profit) AS Profit
    FROM
       Products, Sales, Times
    WHERE
       S_ProductID=P_ProductID AND
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')
    GROUP BY
       P_Department) A,
   (SELECT
       SUM(S_profit) AS TotalProfit
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')) B
ORDER BY
   ProfitQuota DESC
```

Q3. DAY SALES PROFITS QUOTA PER DEPARTMENT, ORDERED BY QUOTA

```
SELECT
   P_Department,
   Profit/TotalProfit*100 AS ProfitQuota
FROM
   (SELECT
       P_Department,
       SUM(S_profit) AS Profit
    FROM
       Products, Sales, Times
    WHERE
       S_ProductID=P_ProductID AND
       S_TimeID=T_TimeID AND
       T_Date=to_date('01-12-2008','DD-MM-YYYY')
    GROUP BY
       P_Department) A,
   (SELECT
       SUM(S_profit) AS TotalProfit
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date=to_date('01-12-2008','DD-MM-YYYY')) B
ORDER BY
   ProfitQuota DESC
```

Q4. YEAR TOTAL SALES, PROFIT AND SHIPCOST VALUES

```
SELECT
   SUM(S_salesamount) AS TotalSalesAmount,
   SUM(S_profit) AS TotalSalesProfit,
   SUM(S_shiptocost) AS TotalShipToCost
FROM
   Sales, Times
WHERE
   S_TimeID=T_TimeID AND
   T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
   T_Date<=to_date('31-12-2008','DD-MM-YYYY')
```

Q5. MONTH TOTAL SALES, PROFIT AND SHIPCOST VALUES

```
SELECT
   SUM(S_salesamount) AS TotalSalesAmount,
   SUM(S_profit) AS TotalSalesProfit,
   SUM(S_shiptocost) AS TotalShipToCost
FROM
   Sales, Times
WHERE
   S_TimeID=T_TimeID AND
   T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
   T_Date<=to_date('30-11-2008','DD-MM-YYYY')
```

Q6. DAY TOTAL SALES, PROFIT AND SHIPCOST VALUES

```
SELECT
   SUM(S_salesamount) AS TotalSalesAmount,
   SUM(S_profit) AS TotalSalesProfit,
   SUM(S_shiptocost) AS TotalShipToCost
FROM
   Sales, Times
WHERE
   S_TimeID=T_TimeID AND
   T_Date=to_date('01-12-2008','DD-MM-YYYY')
```

Q7. TOP 100 CUSTOMERS OF A YEAR WITH HIGHEST TOTAL SALES VALUE, ORDERED BY VALUE

```
SELECT
   TOP 100
   S_CustomerID, C_Name, C_City, TotalSalesAmount
FROM
   (SELECT
       S_CustomerID,
       SUM(S_salesamount) AS TotalSalesAmount
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
       T_Date<=to_date('31-12-2008','DD-MM-YYYY')
    GROUP BY
       S_CustomerID) A, Customers
WHERE
   C_CustomerID=S_CustomerID
ORDER BY
   TotalSalesAmount DESC
```

Q8. TOP 100 CUSTOMERS OF A MONTH WITH HIGHEST TOTAL SALES VALUE, ORDERED BY VALUE

```
SELECT
   TOP 100
   S_CustomerID, C_Name, C_City, TotalSalesAmount
FROM
   (SELECT
       S_CustomerID,
       SUM(S_salesamount) AS TotalSalesAmount
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')
    GROUP BY
       S_CustomerID) A, Customers
WHERE
   C_CustomerID=S_CustomerID
ORDER BY
   TotalSalesAmount DESC
```

Q9. TOP 100 CUSTOMERS OF A DAY WITH HIGHEST TOTAL SALES VALUE, ORDERED BY VALUE

```
SELECT
   TOP 100
   S_CustomerID, C_Name, C_City, TotalSalesAmount
FROM
   (SELECT
       S_CustomerID,
       SUM(S_salesamount) AS TotalSalesAmount
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date=to_date('01-12-2008','DD-MM-YYYY')
    GROUP BY
       S_CustomerID) A, Customers
WHERE
   C_CustomerID=S_CustomerID
ORDER BY
   TotalSalesAmount DESC
```

Q10. YEAR TOTAL SALES QUANTITY AND VALUE PER PROMOTION/PRODUCT OF BRAND #1, ORDERED BY PROMOTION/PRODUCT

```
SELECT
   S_PromotionID, PR_Description, S_ProductID, P_Name,
   Qty, SalesAmount
FROM
   (SELECT
       S_PromotionID, S_ProductID,
       SUM(S_quantity) AS Qty,
       SUM(S_salesamount) AS SalesAmount #
    FROM
       Sales, Times, Products
    WHERE
       S_ProductID=P_ProductID AND
       P_Brand='BRAND #1' AND
       S_PromotionID>0 AND
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
       T_Date<=to_date('31-12-2008','DD-MM-YYYY')
    GROUP BY
       S_PromotionID, S_ProductID
    ORDER BY
       S_PromotionID, S_ProductID), Products, Promotions
WHERE
   S_PromotionID=PR_PromotionID AND
   S_ProductID=P_ProductID
```

Q11. MONTH TOTAL SALES QUANTITY AND VALUE PER PROMOTION/PRODUCT OF BRAND #1, ORDERED BY PROMOTION/PRODUCT

```
SELECT
   S_PromotionID, PR_Description, S_ProductID, P_Name,
   Qty, SalesAmount
FROM
   (SELECT
       S_PromotionID, S_ProductID,
       SUM(S_quantity) AS Qty,
       SUM(S_salesamount) AS SalesAmount
    FROM
       Sales, Times, Products
    WHERE
       S_ProductID=P_ProductID AND
       P_Brand='BRAND #1' AND
       S_PromotionID>0 AND
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')
    GROUP BY
       S_PromotionID, S_ProductID
    ORDER BY
       S_PromotionID, S_ProductID), Products, Promotions
    WHERE
       S_PromotionID=PR_PromotionID AND
       S_ProductID=P_ProductID
```

Q12. DAY TOTAL SALES QUANTITY AND VALUE PER PROMOTION/PRODUCT OF BRAND #1, ORDERED BY PROMOTION/PRODUCT

```
SELECT
   S_PromotionID, PR_Description, S_ProductID, P_Name,
   Qty, SalesAmount
FROM
   (SELECT
       S_PromotionID, S_ProductID,
       SUM(S_quantity) AS Qty,
       SUM(S_salesamount) AS SalesAmount
    FROM
       Sales, Times, Products
    WHERE
       S_ProductID=P_ProductID AND
       P_Brand='BRAND #1' AND
       S_PromotionID>0 AND
       S_TimeID=T_TimeID AND
       T_Date=to_date('01-12-2008','DD-MM-YYYY')
    GROUP BY
       S_PromotionID, S_ProductID
    ORDER BY
       S_PromotionID, S_ProductID), Products, Promotions
WHERE
   S_PromotionID=PR_PromotionID AND
   S_ProductID=P_ProductID
```

Q13. YEAR TOTAL SALES VALUE PER COUNTRY/ZONE, ORDERED BY COUNTRY/ZONE

```
SELECT
   C_Country, ZipCode,
   SUM(S_salesamount) AS TotalSalesAmount
FROM
   (SELECT
       DISTINCT(SUBSTR(c_zipcode,1,3)) AS ZipCode
    FROM
       Customers), Sales, Customers, Times
WHERE
   S_CustomerID=C_CustomerID AND
   S_TimeID=T_TimeID AND
   T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
   T_Date<=to_date('31-12-2008','DD-MM-YYYY') AND
   SUBSTR(C_ZipCode,1,3)=ZipCode
GROUP BY
   C_Country, ZipCode
ORDER BY
   C_Country, TotalSalesAmount DESC, ZipCode
```

Q14. MONTH TOTAL SALES VALUE PER COUNTRY/ZONE, ORDERED BY COUNTRY/ZONE

```
SELECT
   C_Country, ZipCode,
   SUM(S_salesamount) AS TotalSalesAmount
FROM
   (SELECT
       DISTINCT(SUBSTR(c_zipcode,1,3)) AS ZipCode
    FROM
       Customers), Sales, Customers, Times
WHERE
   S_CustomerID=C_CustomerID AND
   S_TimeID=T_TimeID AND
   T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
   T_Date<=to_date('30-11-2008','DD-MM-YYYY') AND
   SUBSTR(C_ZipCode,1,3)=ZipCode
GROUP BY
   C_Country, ZipCode
ORDER BY
   C_Country, TotalSalesAmount DESC, ZipCode
```

Q15. DAY TOTAL SALES VALUE PER COUNTRY/ZONE, ORDERED BY COUNTRY/ZONE

```
SELECT
   C_Country, ZipCode,
   SUM(S_salesamount) AS TotalSalesAmount
FROM
   (SELECT
       DISTINCT(SUBSTR(c_zipcode,1,3)) AS ZipCode
    FROM
       Customers), Sales, Customers, Times
WHERE
   S_CustomerID=C_CustomerID AND
```

```
   S_TimeID=T_TimeID AND
   T_Date=to_date('01-12-2008','DD-MM-YYYY')
   SUBSTR(C_ZipCode,1,3)=ZipCode
GROUP BY
   C_Country, ZipCode
ORDER BY
   C_Country, TotalSalesAmount DESC, ZipCode
```

Q16. YEAR TOTAL SALES VALUE PER CUSTOMER AGE CLASS, PER PRODUCT, ORDERED BY SALES VALUE

```
SELECT
   S_ProductID, P_Name, C_Gender,
   SUM(CASE WHEN C_Income<600 THEN S_salesamount
            ELSE 0 END) AS MinimumIncome,
   SUM(CASE WHEN C_Income>=600 AND C_Income<1000 THEN S_salesamount
            ELSE 0 END) AS ReasonableIncome,
   SUM(CASE WHEN C_Income>=1000 AND C_Income<1500 THEN S_salesamount
            ELSE 0 END) AS MediumIncome,
   SUM(CASE WHEN C_Income>=1500 AND C_Income<2500 THEN S_salesamount
            ELSE 0 END) AS HighIncome,
   SUM(CASE WHEN C_Income>=2500 THEN S_salesamount
            ELSE 0 END) AS VeryHighIncome
FROM
   Sales, Products, Customers, Times
WHERE
   S_CustomerID=C_CustomerID AND
   S_ProductID=P_ProductID AND
   S_TimeID=T_TimeID AND
   T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
   T_Date<=to_date('31-12-2008','DD-MM-YYYY')
GROUP BY
   S_ProductID, P_Name, C_Gender
ORDER BY
   MinimumIncome+ReasonableIncome+MediumIncome+
   HighIncome+VeryHighIncome DESC
```

Q17. MONTH TOTAL SALES VALUE PER CUSTOMER AGE CLASS, PER PRODUCT, ORDERED BY SALES VALUE

```
SELECT
   S_ProductID, P_Name, C_Gender,
   SUM(CASE WHEN C_Income<600 THEN S_salesamount
            ELSE 0 END) AS MinimumIncome,
   SUM(CASE WHEN C_Income>=600 AND C_Income<1000 THEN S_salesamount
            ELSE 0 END) AS ReasonableIncome,
   SUM(CASE WHEN C_Income>=1000 AND C_Income<1500 THEN S_salesamount
            ELSE 0 END) AS MediumIncome,
   SUM(CASE WHEN C_Income>=1500 AND C_Income<2500 THEN S_salesamount
            ELSE 0 END) AS HighIncome,
   SUM(CASE WHEN C_Income>=2500 THEN S_salesamount
            ELSE 0 END) AS VeryHighIncome
FROM
   Sales, Products, Customers, Times
WHERE
   S_CustomerID=C_CustomerID AND
```

```
   S_ProductID=P_ProductID AND
   S_TimeID=T_TimeID AND
   T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
   T_Date<=to_date('30-11-2008','DD-MM-YYYY')
GROUP BY
   S_ProductID, P_Name, C_Gender
ORDER BY
   MinimumIncome+ReasonableIncome+MediumIncome+
   HighIncome+VeryHighIncome DESC
```

Q18. DAY TOTAL SALES VALUE PER CUSTOMER AGE CLASS, PER PRODUCT, ORDERED BY SALES VALUE

```
SELECT
   S_ProductID, P_Name, C_Gender,
   SUM(CASE WHEN C_Income<600 THEN S_salesamount
            ELSE 0 END) AS MinimumIncome,
   SUM(CASE WHEN C_Income>=600 AND C_Income<1000 THEN S_salesamount
            ELSE 0 END) AS ReasonableIncome,
   SUM(CASE WHEN C_Income>=1000 AND C_Income<1500 THEN S_salesamount
            ELSE 0 END) AS MediumIncome,
   SUM(CASE WHEN C_Income>=1500 AND C_Income<2500 THEN S_salesamount
            ELSE 0 END) AS HighIncome,
   SUM(CASE WHEN C_Income>=2500 THEN S_salesamount
            ELSE 0 END) AS VeryHighIncome
FROM
   Sales, Products, Customers, Times
WHERE
   S_CustomerID=C_CustomerID AND
   S_ProductID=P_ProductID AND
   S_TimeID=T_TimeID AND
   T_Date=to_date('01-12-2008','DD-MM-YYYY')
GROUP BY
   S_ProductID, P_Name, C_Gender
ORDER BY
   MinimumIncome+ReasonableIncome+MediumIncome+
   HighIncome+VeryHighIncome DESC
```

Q19. YEAR TOTAL SALES VALUE AND RESPECTIVE QUOTA PER COUNTRY, ORDERED BY VALUE

```
SELECT
   C_Country, SalesAmount,
   SalesAmount/TotalSalesAmount*100 AS SalesQuota
FROM
    (SELECT
        SUM(S_salesamount) AS TotalSalesAmount
     FROM
        Sales, Times
     WHERE
        S_TimeID=T_TimeID AND
        T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
        T_Date<=to_date('31-12-2008','DD-MM-YYYY')),
    (SELECT
        C_Country,
        SUM(S_salesamount) AS SalesAmount
```

```
    FROM
       Sales, Customers, Times
    WHERE
       S_CustomerID=C_CustomerID AND
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
       T_Date<=to_date('31-12-2008','DD-MM-YYYY')
    GROUP BY
       C_Country)
ORDER BY
   SalesAmount DESC
```

Q20. MONTH TOTAL SALES VALUE AND RESPECTIVE QUOTA PER COUNTRY, ORDERED BY VALUE

```
SELECT
   C_Country, SalesAmount,
   SalesAmount/TotalSalesAmount*100 AS SalesQuota
FROM
   (SELECT
       SUM(S_salesamount) AS TotalSalesAmount
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')),
   (SELECT
       C_Country,
       SUM(S_salesamount) AS SalesAmount
    FROM
       Sales, Customers, Times
    WHERE
       S_CustomerID=C_CustomerID AND
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')
    GROUP BY
       C_Country)
ORDER BY
   SalesAmount DESC
```

Q21. DAY TOTAL SALES VALUE AND RESPECTIVE QUOTA PER COUNTRY, ORDERED BY VALUE

```
SELECT
   C_Country, SalesAmount,
   SalesAmount/TotalSalesAmount*100 AS SalesQuota
FROM
   (SELECT
       SUM(S_salesamount) AS TotalSalesAmount
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date=to_date('01-12-2008','DD-MM-YYYY')),
```

```
   (SELECT
       C_Country,
       SUM(S_salesamount) AS SalesAmount
    FROM
       Sales, Customers, Times
    WHERE
       S_CustomerID=C_CustomerID AND
       S_TimeID=T_TimeID AND
       T_Date=to_date('01-12-2008','DD-MM-YYYY')
    GROUP BY
       C_Country)
ORDER BY
   SalesAmount DESC
```

Q22. LIST OF PRODUCTS NEVER SOLD DURING THE YEAR, ORDERED BY PRODUCT

```
SELECT
   S_ProductID, P_Name, P_Brand, P_Category, P_Department
FROM
   Sales, Products
WHERE
   S_ProductID=P_ProductID AND
   S_ProductID NOT IN
      (SELECT
          DISTINCT(S_ProductID)
          FROM
             Sales, Times
          WHERE
             S_TimeID=T_TimeID AND
             T_Date=to_date('01-01-2008','DD-MM-YYYY'))
ORDER BY
   S_ProductID
```

Q23. LIST OF PRODUCTS NEVER SOLD DURING THE MONTH, ORDERED BY PRODUCT

```
SELECT
   S_ProductID, P_Name, P_Brand, P_Category, P_Department
FROM
   Sales, Products
WHERE
   S_ProductID=P_ProductID AND
   S_ProductID NOT IN
      (SELECT
          DISTINCT(S_ProductID)
       FROM
          Sales, Times
       WHERE
          S_TimeID=T_TimeID AND
          T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
          T_Date<=to_date('30-11-2008','DD-MM-YYYY'))
ORDER BY
   S_ProductID
```

Q24. LIST OF PRODUCTS NEVER SOLD DURING THE DAY, ORDERED BY PRODUCT

```
SELECT
   S_ProductID, P_Name, P_Brand, P_Category, P_Department
FROM
   Sales, Products
WHERE
   S_ProductID=P_ProductID AND
   S_ProductID NOT IN
      (SELECT
          DISTINCT(S_ProductID)
       FROM
          Sales, Times
       WHERE
          S_TimeID=T_TimeID AND
          T_Date=to_date('01-12-2008','DD-MM-YYYY'))
ORDER BY
   S_ProductID
```

Q25. NUMBER OF PURCHASES MADE PER CUSTOMER DURING THE YEAR, ORDERED BY COUNTRY, CITY, ZONE, NUMBER OF PURCHASES

```
SELECT
   S_CustomerID, C_Name, C_City, C_ZipCode, C_Country
FROM
   (SELECT
       S_CustomerID,
       COUNT(*) AS Conta
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
       T_Date<=to_date('31-12-2008','DD-MM-YYYY')
    GROUP BY
       S_CustomerID), Customers
WHERE
   S_CustomerID=C_CustomerID AND
   Conta>0
ORDER BY
   C_Country, C_City, C_ZipCode, Conta DESC
```

Q26. NUMBER OF PURCHASES MADE PER CUSTOMER DURING THE MONTH, ORDERED BY COUNTRY, CITY, ZONE, NUMBER OF PURCHASES

```
SELECT
   S_CustomerID, C_Name, C_City, C_ZipCode, C_Country
FROM
   (SELECT
       S_CustomerID,
       COUNT(*) AS Conta
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
```

```
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')
    GROUP BY
       S_CustomerID), Customers
WHERE
   S_CustomerID=C_CustomerID AND
   Conta>0
ORDER BY
   C_Country, C_City, C_ZipCode, Conta DESC
```

Q27. NUMBER OF PURCHASES MADE PER CUSTOMER DURING THE DAY, ORDERED BY COUNTRY, CITY, ZONE, NUMBER OF PURCHASES

```
SELECT
   S_CustomerID, C_Name, C_City, C_ZipCode, C_Country
FROM
   (SELECT
       S_CustomerID,
       COUNT(*) AS Conta
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date=to_date('01-12-2008','DD-MM-YYYY')
    GROUP BY
       S_CustomerID), Customers
WHERE
   S_CustomerID=C_CustomerID AND
   Conta>0
ORDER BY
   C_Country, C_City, C_ZipCode, Conta DESC
```

Q28. MONTHLY TOTAL SALES VALUE AND RESPECTIVE QUOTA FOR THE YEAR, ORDERED BY MONTH

```
SELECT
   SalesMonth, MonthTotalSalesAmount,
   MonthTotalSalesAmount/TotalSalesAmount AS MonthQuota
FROM
   (SELECT
       SUM(S_salesamount) AS TotalSalesAmount
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
       T_Date<=to_date('31-12-2008','DD-MM-YYYY')) a,
   (SELECT SalesMonth,
       SUM(TotSalesAmount) AS MonthTotalSalesAmount
    FROM
       (SELECT
           to_char(T_Date,'Month') AS SalesMonth,
           S_salesamount AS TotSalesAmount
        FROM
           Sales, Times
        WHERE
           S_TimeID=T_TimeID AND
```

```
          T_Date>=to_date('01-01-2008','DD-MM-YYYY') AND
          T_Date<=to_date('31-12-2008','DD-MM-YYYY')) b
    GROUP BY
       SalesMonth) c
ORDER BY
   SalesMonth
```

Q29. DAILY TOTAL SALES VALUE AND RESPECTIVE QUOTA FOR A MONTH, ORDERED BY DAY

```
SELECT
   T_Date, DayTotalSalesAmount,
   DayTotalSalesAmount/TotalSalesAmount AS DayQuota
FROM
   (SELECT
       SUM(S_salesamount) AS TotalSalesAmount
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')) a,
   (SELECT
       T_Date,
       SUM(S_salesamount) AS DayTotalSalesAmount
    FROM
       Sales, Times
    WHERE
       S_TimeID=T_TimeID AND
       T_Date>=to_date('01-11-2008','DD-MM-YYYY') AND
       T_Date<=to_date('30-11-2008','DD-MM-YYYY')
    GROUP BY
       T_Date) b
ORDER BY
   T_Date
```

Appendix B

Data Masking and Encryption Experimental Results

In this appendix we present the averages and standard deviations for the data masking and encryption experimental results described in the thesis. As mentioned in the respective chapters, each result is obtained from the execution of six rounds of experiments, referring to the following legend labels:

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
OPES	Data encrypted with OPES
Salsa20	Data encrypted with Salsa20/20
MOBAT AddCol	Data masked by MOBAT, where a column for masking keys has been added to the existing fact table
MOBAT CreateCol	Data masked by MOBAT, where a column for masking keys was added to the fact table, which has been completely recreated
MOBAT ColKey	Data masked by MOBAT, using a numerical column from the original fact table data structure as key $K_{3,j}$
SES-DW128	Data encrypted using SES-DW with 128 bit security
SES-DW256	Data encrypted using SES-DW with 256 bit security
SES-DW1024	Data encrypted using SES-DW with 1024 bit security

B.1. Data Masking Chapter Loading Time Results

Tables B-1 to B-3 show the results in seconds for the average (μ) and standard deviation (σ) of the data masking loading experiments, obtained using a Pentium IV 2.8 GHz CPU with 2GB RAM.

Table B-1. TPC-H 1GB Loading Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	310	9,86777	212	8,99475
AES128	899	46,94247	472	36,75193
AES256	958	44,63968	507	31,48409
3DES168	906	33,61551	485	21,38157
OPES	461	20,87444	305	22,10521
Salsa20	537	26,42794	361	26,65626
MOBAT AddCol	335	14,81949	227	12,39097
MOBAT CreateCol	323	14,70876	221	11,69447
MOBAT ColKey	318	12,81143	218	11,93016

Table B-2. TPC-H 10 GB Loading Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	3211	121,9969	2272	96,2474
AES128	10185	387,1303	5484	233,5230
AES256	11114	434,7008	6229	254,6556
3DES168	10424	508,4449	5635	257,1251
OPES	4943	222,8019	3325	160,8512
Salsa20	5881	185,4172	4088	180,0211
MOBAT AddCol	3597	181,0830	2550	155,7417
MOBAT CreateCol	3449	151,5198	2434	154,1759
MOBAT ColKey	3362	144,0208	2381	131,4362

Table B-3. Sales DW Loading Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	1195	74,2938	1247	70,9444
AES128	3574	155,6558	3232	111,3055
AES256	3699	162,8546	3381	117,3645
3DES168	3695	140,0080	3339	146,1417
OPES	1929	117,5107	1963	71,6937
Salsa20	2408	84,0577	2459	97,3811
MOBAT AddCol	1373	83,7072	1447	76,2599
MOBAT CreateCol	1308	79,9533	1367	80,6815
MOBAT ColKey	1260	80,7588	1318	78,5291

B.2. Data Masking Chapter Query Workloads Execution Time Results

Tables B-4 to B-6 show the results in seconds for the average (μ) and standard deviation (σ) of the data masking query workload execution experiments, obtained using a Pentium IV 2.8 GHz CPU with 2GB RAM.

Table B-4. TPC-H 1GB Query Workload Execution Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	625	50,6069	580	54,4009
AES128	1798	223,0013	1591	199,6768
AES256	1837	212,8436	1646	172,8946
3DES168	1895	175,8836	1712	186,4174
OPES	1813	158,0126	1629	137,4651
Salsa20	1727	163,8821	1523	154,9399
MOBAT AddCol	846	76,7923	813	82,0243
MOBAT CreateCol	809	79,0004	775	69,8340
MOBAT ColKey	763	86,0046	712	76,4791

Table B-5. TPC-H 10 GB Query Workload Execution Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	6155	481,3438	5301	406,6876
AES128	16927	1701,2962	13334	949,9173
AES256	17283	1767,3377	13846	1213,7299
3DES168	17973	1741,0874	15058	1266,3514
OPES	16889	1575,5657	13215	1172,8934
Salsa20	15704	1118,5171	12691	1054,1071
MOBAT AddCol	7527	762,7053	6420	715,2876
MOBAT CreateCol	7314	819,1865	6162	480,4649
MOBAT ColKey	7218	702,9792	5981	447,6100

Table B-6. Sales DW Query Workload Execution Time

	Oracle		SQL Server	
	μ	σ	μ	σ
Standard	2233	172,8706	2211	200,3533
AES128	17604	1399,6442	16923	1974,8563
AES256	18484	1619,3473	17827	1578,0671
3DES168	20425	1777,9447	18984	1827,5253
OPES	17465	1376,6070	16845	1497,5728
Salsa20	15582	845,2452	15212	1435,1688
MOBAT AddCol	5084	390,5519	4946	279,9171
MOBAT CreateCol	4435	462,5449	4313	240,3703
MOBAT ColKey	3966	283,0312	3637	264,4148

B.3. Encryption Chapter Loading Time Results

Tables B-7 to B-9 show the results in seconds for the average (μ) and standard deviation (σ) of the encryption loading experiments, obtained using a Core2Duo 3 GHz CPU with 2GB RAM.

Table B-7. TPC-H 1GB Loading Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	253	12,2420	171	9,6231
AES128	608	28,4159	382	14,3341
AES256	636	29,6265	407	19,2423
3DES168	617	31,9687	389	20,1096
OPES	353	17,3743	229	21,5238
Salsa20	419	24,6833	281	21,5931
SES-DW128	279	15,9888	191	15,8537
SES-DW256	294	20,3858	201	18,3346
SES-DW1024	451	21,4445	284	19,7159

Table B-8. TPC-H 10 GB Loading Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	2576	132,6468	1796	99,7148
AES128	6375	302,7141	4144	214,7684
AES256	6742	342,2266	4532	193,2705
3DES168	6527	384,4802	4290	245,6537
OPES	3766	153,7396	2542	102,1442
Salsa20	4481	190,5514	3106	129,0725
SES-DW128	3024	140,4549	2137	103,1846
SES-DW256	3216	153,7929	2320	109,6005
SES-DW1024	4844	200,5901	3516	133,9737

Table B-9. Sales DW Loading Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	994	38,4313	1013	47,0286
AES128	2676	125,6391	2416	97,7693
AES256	2889	89,9725	2573	111,7741
3DES168	2949	78,9573	2611	123,8752
OPES	1555	77,0835	1554	57,2072
Salsa20	1902	84,6333	1879	78,4652
SES-DW128	1124	46,8944	1161	54,5001
SES-DW256	1211	57,4479	1237	64,4903
SES-DW1024	1808	71,6928	1881	89,6482

B.4. Encryption Query Workloads Execution Time Results

Tables B-10 to B-12 show the results in seconds for the average (μ) and standard deviation (σ) of the encryption query workload execution experiments, obtained using a Core2Duo 3 GHz CPU with 2GB RAM.

Table B-10. TPC-H 1GB Query Workload Execution Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	492	48,4052	452	39,2937
AES128	1357	124,1525	1231	124,3141
AES256	1496	130,1163	1330	153,0616
3DES168	1702	167,9543	1362	159,4373
OPES	1535	136,5459	1326	99,6848
Salsa20	1268	95,7280	1131	98,4518
MOBAT AddCol	1015	93,4154	927	89,7789
MOBAT CreateCol	1251	126,5178	1140	106,7907
MOBAT ColKey	1453	117,9790	1325	96,2909

Table B-11. TPC-H 10 GB Query Workload Execution Time

	Oracle 11g		SQL Server 2008	
	μ	σ	μ	σ
Standard	5037	531,1588	4694	459,2833
AES128	15191	1358,3464	14063	993,7016
AES256	19073	1116,7794	16650	1276,2821
3DES168	22053	2105,4593	18821	1447,4942
OPES	17205	1205,4704	14155	1256,6578
Salsa20	14623	965,2504	13540	1080,3754
SES-DW128	9893	671,6570	9446	580,0519
SES-DW256	12056	973,8139	10289	916,5035
SES-DW1024	14976	1520,3692	13713	1153,3621

Table B-12. Sales DW Query Workload Execution Time

	Oracle		SQL Server	
	μ	σ	μ	σ
Standard	1766	143,4475	1690	181,5121
AES128	14101	1409,7929	13429	1117,5437
AES256	15490	1160,3142	14180	1013,8596
3DES168	15860	1645,6413	14898	1467,3108
OPES	14189	1272,7239	12381	1149,5012
Salsa20	11294	1078,3294	10019	868,6609
SES-DW128	6396	374,6025	5682	434,3993
SES-DW256	8998	512,0796	7806	612,9569
SES-DW1024	12546	1131,3574	10032	980,1660

Appendix C

Intrusion Detection Experimental Results

In this appendix we present the experimental results on intrusion detection described in Chapter 6 of the thesis. Tables C-1 to C-4 show the results for the average (μ) and standard deviation (σ) of the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN) generated by DW-DIDS in each scenario ("number of true users"-"number of intruders").

Table C-1. DW-DIDS ID Results for Profiles built from 5 "True" User Workloads

	TP		FP		TN		FN	
Scenario	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**
10-0	0	0	57	2,7358	1193	70,1898	0	0
9-1	62	3,3922	54	2,5322	1076	58,6154	38	1,0416
8-2	131	7,4332	76	4,2092	944	55,5715	69	2,9680
5-5	327	20,9846	282	15,3613	378	21,0100	173	10,0307

Table C-2. DW-DIDS ID Results for Profiles built from 25 "True" User Workloads

	TP		FP		TN		FN	
Scenario	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**
10-0	0	0	14	0,9442	1236	66,7567	0	0
9-1	81	4,8560	42	2,0296	1088	63,1729	19	1,9311
8-2	167	7,9252	54	2,6129	966	57,5426	33	1,4605
5-5	427	25,1637	221	12,8374	439	26,4010	73	3,9543

Table C-3. DW-DIDS ID Results for Profiles built from 50 "True" User Workloads

	TP		FP		TN		FN	
Scenario	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**
10-0	0	0	12	0,7048	1238	65,5332	0	0
9-1	85	4,4048	38	1,0416	1092	67,4688	15	1,1095
8-2	177	11,1384	48	1,9226	972	53,1624	23	0,9102
5-5	459	27,7095	204	11,5661	456	23,2852	41	2,3537

Table C-4. DW-DIDS ID Results for Profiles built from 100 "True" User Workloads

	TP		FP		TN		FN	
Scenario	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**	**μ**	**σ**
10-0	0	0	9	0,8625	1241	75,2322	0	0
9-1	88	5,0469	32	1,2429	1098	65,9825	12	0,7329
8-2	183	11,0307	43	2,0296	977	54,1515	17	1,4190
5-5	477	28,3424	193	11,2083	467	28,4039	23	1,8280

Appendix D

Intrusion Detection Benchmark

As current work under development, in this appendix we present a draft proposal for a DW Intrusion Detection Benchmark (DWID-Bench) for testing DIDS in DWs at the SQL level, given a controlled DW environment with mixed intrusion and non-intrusion SQL workloads.

The benchmark's main aim is to provide a feasible and objective mean of evaluating the efficiency of the intrusion detection processes and impact in database performance at the SQL level for DW DIDS. The proposed measures intend to produce insight for aiding developers in the improvement of their solutions and allow solution providers and clients to compare between different solutions.

To accomplish this, we consider the typical DW user workloads and intrusion detection techniques described in Chapter 2 and the SQL intrusion action type classification in Chapter 6. The chosen "intrusion" workload covers a broad scope of distinct types of SQL intrusion actions against DWs. The "intrusion" workload is executed concurrently with defined "non-intrusion" workloads, which are selected from the well-known TPC-DS benchmark to represent a typical decision support user workload, in order to simulate a scenario as close to reality as possible.

The remainder of this appendix is organized as follows. In Section D.1 we present the benchmark and describe its setup. In Section D.2 we present the database schema used in the benchmark. Sections D.3 and D.4 respectively explain the "non-intrusion" and "intrusion" workloads and how they are defined. Section D.5 describes the benchmark's execution rules and procedures, while Section D.6 describes its proposed metrics. In Section D.7 we discuss open issues regarding the development of the benchmark and finally, Section D.8 summarizes the benchmark proposal and points out future work.

D.1. DWID-Bench: Data Warehouse Intrusion Detection Benchmark

Figure D-1 shows the key components of the experimental setup required to run DWID-Bench. As in TPC-DS [TPC-DS], the main elements are the *System Under Test (SUT)* and the *Driver System*. The goal of the *Driver System* is to emulate the client applications and respective users and control all the aspects of each benchmark run. In the *Driver System* we include both the "non-intruder" and "intruder" users. Additionally, the *Driver System* also records the raw data needed to calculate the benchmark measures (which are computed afterwards by analyzing the data collected during each benchmark run).

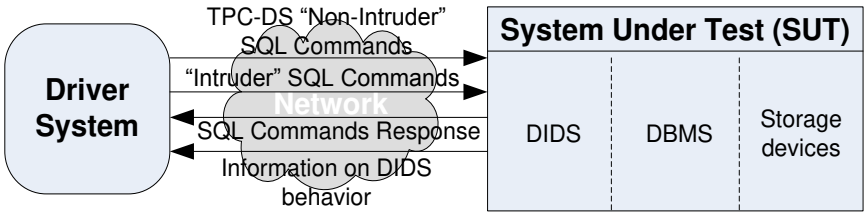


Figure D-1. DWID-Bench experimental setup

The *SUT* represents a client-server system fully configured to run both intruder and non-intruder workloads coming from the *Driver System* and includes the DIDS to be evaluated. From the benchmark point of view, the *SUT* is composed by the DIDS and the set of processing units used to run the workloads and to store all the data processed. In other words, the *SUT* can be any (hardware + software) system able to run the complete benchmark workload and execute the DIDS algorithms under the conditions specified by the benchmark procedure. The communication between the *Driver System* and the *SUT* may be executed through any type of LAN or WAN network infrastructures.

D.2. DWID-Bench Database Schema

In DWID-Bench, we partially use the data schemas proposed by TPC-DS. The TPC-DS has been released after we had partially executed the experiments presented throughout the thesis, and is the latest and probably the currently mostly used benchmark for measuring the throughput performance of Decision Support Systems (DSS). The TPC-DS

benchmark has been mapped to a typical business environment and claims to significantly represent DSS that:

- Examine large volumes of data;
- Give answers to real-world business questions;
- Execute queries of various operational requirements and complexities (*e.g. ad-hoc* instructions, reporting actions, data mining operations, etc);
- Are characterized by high CPU and I/O load;
- Are periodically synchronized with transactional source databases through database maintenance functions.

Assuming these features are common to a typical DW environment, as described in [Kimball and Ross, 2013], we accept the TPC-DS as representative of DSS and partially use its defined data schemas and workloads in DWID-Bench. The "intrusion" and "non-intrusion" DWID-Bench workloads focus on users with ETL and DW End User privileges, since these are the type of actions covered by the TPC-DS benchmark. We also define a set of actions for simulating DBA users as a mix of ETL + DW End User actions, plus DDL commands relating to the creation of tables, constraints and indexes belonging to the chosen schema.

The TPC-DS focuses on a generic retail business DSS for any industry that must manage, sell and distribute products. Its schema models the sales and sales returns process for an organization that employs three primary sales channels: stores, catalogs, and the Internet. Each of these channels has two fact tables, for storing the facts concerning sales and sales returns. There is also another fact table for modeling inventory for the catalog and Internet sales channels. Each fact table is linked with its respective dimensions in a star schema, which means the complete TPC-DS data schema is a set of seven star schemas, interlinked by their shared dimensions.

In DWID-Bench, we chose to use the TPC-DS store sales star schema, illustrated in Figure D-2. We chose this particular schema because it represents a common business DW scenario for many enterprises, within the set of proposed star schemas in TPC-DS. Moreover, the *Store_Sales* fact table is the biggest sized fact table of all generated tables in the complete TPC-DS database. As shown in Figure D-2, it is composed of one fact table

and ten dimension tables. In the following sections, we explain how the "intrusion" and "non-intrusion" workloads are defined.

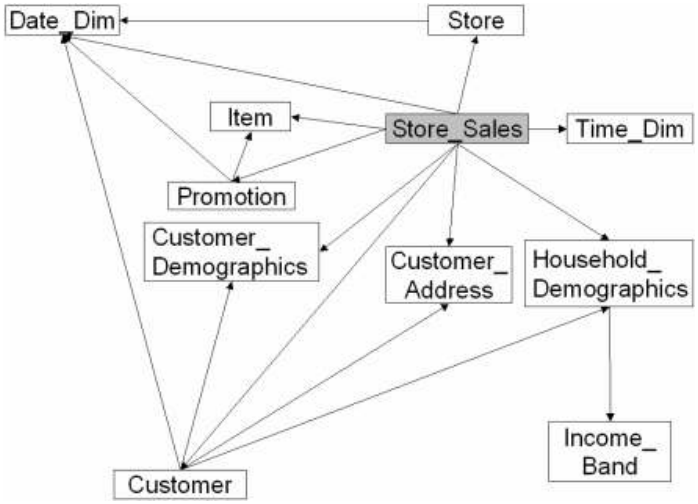


Figure D-2. TPC-DS store sales E-R diagram [TPC-DS]

D.3. DWID-Bench "Non-intrusion" Workload

The TPC-DS models a database that is continuously available 24 hours a day, 7 days a week, for data modifications against any/all tables and various types (*e.g. ad hoc*, reporting, iterative OLAP and data mining) of queries originating from multiple concurrent user sessions. This environment allows potentially long running and multi-part queries where the DBA cannot assume that the database can be inactive during any particular period. Queries and data maintenance functions may execute concurrently. Since we use the store sales star schema for our DW database, we use the predefined TPC-DS query and data maintenance workloads for the store sales star schema as our chosen "non-intruder" workloads in DWID-Bench.

From the DWID-Bench perspective, each session with an open connection to the database refers to a given type of user (ETL, DW end user, or DBA, as described in Chapter 6 of this thesis). The benchmark expects each session to execute a stream of actions, which depend on the type of user and defined as the following:

- For sessions simulating "non-intrusion" users with ETL database privileges, data maintenance routines for all the tables of the store sales data schema are executed, exactly as defined in TPC-DS;
- For sessions simulating "non-intrusion" DW end users (*i.e.* typical business managers, analysts and decision makers), each session will execute a query stream with the complete set of SQL queries defined in TPC-DS that request processing data from the store sales star schema, thus totalizing 32 distinct queries for each stream, taken from the total of 99 queries defined in TPC-DS. The complete set of selected TPC-DS queries for composing the DWID-Bench "non-intrusion" workload is thus { Q3, Q6, Q7, Q8, Q13, Q19, Q27, Q28, Q34, Q36, Q42, Q43, Q44, Q46, Q47, Q48, Q52, Q53, Q55, Q59, Q61, Q63, Q65, Q67, Q68, Q70, Q73, Q79, Q88, Q89, Q96, Q98 }. For each benchmark run, each stream is expected to execute each distinct query once, in which their execution order is defined by the query ordering established in TPC-DS;
- For sessions simulating users with DBA privileges, the workload definition is very difficult to define, given the dynamic and huge scope of actions they can execute. The TPC-DS benchmark does not have any type of approach on actions coming from users with this profile, and to present an abstraction that strictly defines a finite set of particular actions for this type of user may risk the representativeness of the workload for this type of users in what concerns the benchmark. From this perspective, we consider the DBA user as someone that has privileges to execute any type of action that can be performed by ETL and DW end users (*i.e.*, DML commands – insert, update and select; delete is not considered, since typical DW maintenance involves only modifying or insertion of new data), plus common database object creation and maintenance actions such as creating, modifying and deleting tables and indexes (*i.e.*, any sort of DDL commands – drop, create, etc). Thus, in DWID-Bench we define the DBA workload as the mix of the ETL and DW end user workloads together, plus all the DDL commands needed for creating tables, constraints and indexes (*i.e.*, primary and secondary indexes, possible bitmap join indexes, key and referential integrity constraint instructions) for the store sales star schema.

The execution matrix for the "non-intrusion" workloads can be seen in Table D-1, displaying the query order for the maximum of 20 DW End User streams that can be executed in DWID-Bench. It shows the order in which each of the 32 queries chosen from the TPC-DS queries (identified by their number in TPC-DS) should be executed, depending on which user (1 to 20) it refers to. The assumptions and rules on how each user workload should be executed for each user stream in each benchmark run will be explained further in Section D.5.

Table D-1. "Non-Intrusion" DW End-User Workload – Query Ordering

Query Sequence Number	DW End-User Stream Number																			
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
1	96	98	98	89	79	73	34	70	98	88	43	7	68	61	46	27	48	61	42	47
2	7	96	59	52	8	98	88	53	59	52	48	43	88	53	42	47	63	8	19	55
3	44	13	88	53	89	88	44	6	70	13	53	13	44	98	79	73	61	67	47	13
4	19	36	6	7	46	19	53	34	44	7	96	36	28	89	6	46	34	42	53	44
5	43	63	27	63	48	65	7	13	73	27	63	98	3	68	96	13	36	88	7	73
6	27	3	28	13	59	3	73	28	3	34	70	8	19	13	8	59	52	70	48	28
7	36	6	68	19	19	79	36	98	7	65	36	88	53	43	89	88	53	13	79	27
8	46	28	8	96	6	13	89	79	36	28	7	28	52	63	88	68	28	28	70	67
9	63	27	63	8	28	6	28	48	61	42	98	59	27	36	28	36	88	46	36	34
10	59	8	19	36	44	52	6	47	65	67	47	96	98	96	55	44	73	89	73	52
11	98	52	55	43	88	7	65	46	67	36	28	55	67	65	48	19	42	6	59	36
12	70	61	42	47	36	36	42	55	6	48	55	63	61	42	36	79	89	53	96	7
13	67	88	53	3	61	61	47	52	47	6	44	3	8	19	47	53	67	34	65	6
14	28	68	67	46	55	53	59	44	55	59	27	52	7	7	59	61	8	47	55	96
15	47	67	44	59	52	68	8	3	42	61	59	47	55	44	27	3	59	79	44	88
16	3	79	61	55	27	28	19	19	34	68	67	46	48	67	44	28	65	19	88	53
17	89	43	73	27	63	42	61	42	79	47	52	53	63	3	65	6	6	48	63	8
18	6	47	96	42	47	67	3	43	28	98	42	67	43	48	98	65	46	27	89	65
19	52	19	36	34	70	48	67	61	89	70	65	27	42	55	19	52	68	52	27	79
20	42	53	43	68	7	63	98	7	46	73	88	70	65	8	53	63	44	55	46	3
21	8	55	52	44	96	8	48	73	19	3	34	61	59	88	68	98	43	96	28	63
22	88	46	70	67	68	46	79	89	8	79	73	34	36	47	43	55	3	36	52	46
23	65	65	7	6	43	55	46	36	96	89	79	73	70	52	3	96	98	59	68	59
24	34	70	79	28	65	89	96	8	88	19	6	6	34	28	7	67	70	68	43	43
25	48	59	65	65	98	96	68	63	43	96	68	89	79	70	34	43	96	98	61	61
26	73	48	34	61	42	27	63	68	53	46	89	65	89	73	73	48	79	3	34	89
27	55	34	3	73	34	44	27	67	52	53	61	44	46	79	70	7	55	73	3	42
28	53	89	13	98	3	43	43	88	68	55	46	79	96	34	67	70	13	43	13	70
29	79	7	46	70	13	47	52	65	13	44	13	42	73	27	13	34	19	63	6	68
30	13	73	89	48	73	70	13	59	48	63	8	48	47	6	63	89	27	7	8	48
31	68	44	47	79	67	59	70	96	27	43	3	19	13	59	52	8	47	44	67	19
32	61	42	48	88	53	34	55	27	63	8	19	68	6	46	61	42	7	65	98	98

In the following section we define the benchmark's "intrusion" workload.

D.4. DWID-Bench "Intrusion" Workload

The chosen intrusion actions intend to provide a wide coverage of the possible types of attacks described in Section 6.1 and most of the database threats discussed in published work [Schulman, 2007] that can be dealt with at the SQL level. Considering these threats, the types of attacks against DWs (described in Chapter 6), the classes of intruder actions presented in Table 6-1, and focusing on the specific business of the TPC-DS store sales data schema, we assume that the possible "intruder" profile is an attacker that has access to the database and pursues answers for the following generic questions:

- How are the store sales DW data structures (*i.e.* table, indexes and column names and types) implemented in the database, and how can they be reached? (SQL intrusion action class B defined in Table 6-1)
- How can the optimization data structures such as indexes be deleted so database performance is degraded? (SQL action class B and C)
- How can the existing data structures such as tables and views be deleted so that DW availability is affected and business information is lost? (SQL action class C)
- How to obtain the complete set of business values from the fact or dimension tables? (SQL action class D)
- How to obtain the full set of business values for a certain item, item brand, class or category, time period, city, county or state? (SQL action class E and F)
- How to obtain the grouped set (*e.g.* sum, average, count) of interesting business values for a certain item, item brand, item class, item category, time period, city, county or state? (SQL action class F)
- How to flood the database services with requests that can overwhelm them by creating CPU and I/O server and network bottlenecks? (SQL action class G)
- How can false data be inserted into the store sales fact table so that decision support may be compromised? (SQL action class H)

- How to modify or erase data so that decision support may become compromised? (SQL action class I and J)

In DWID-Bench we assume a set of instructions that are able to respond to these questions as the set of representative "intrusion" actions for the chosen database schema.

For each intrusion action in which there are parameter variables (shown in brackets []), these should be given a value as defined in the list of random parameter variables. Each parameter value should be refreshed for each query in each intrusion action stream, so that the same parameters which are used in more than one action in the stream does not have its value repeated amongst the remaining actions (*e.g.* ITEM_K should have five distinct values for actions IA06, IA12, IA17, IA22, and IA34, shown further on).

The random generator is defined as a Mersenne Twister Pseudo-Random Number Generator[8] [Matsumoto and Nishimura, 1998], which is, by far, the most widely used PRNG [Marsland, 2011]. Its name derives from the fact that its period length[9] is chosen to be a Mersenne prime. The most commonly used version of the Mersenne Twister algorithm is based on the Mersenne prime $2^{19937}-1$ (alias MT19937). It has a period of $2^{19937}-1$ iterations ($\approx 4.3\times10^{6001}$), is proven to be equidistributed in (up to) 623 dimensions (for

[8] A pseudorandom number generator (PRNG), also known as a deterministic random bit generator (DRBG) is an algorithm for generating a sequence of numbers that approximates the properties of random numbers [Barker *et al.*, 2012]. The sequence is not truly random in that it is completely determined by a relatively small set of initial values, called the PRNG's *state*, which includes a truly random seed. Although sequences that are closer to truly random can be generated using hardware random number generators, *pseudorandom* numbers are important in practice for their speed in number generation and their reproducibility.

[9] A PRNG can be started from an arbitrary starting state using a seed state. It will always produce the same sequence thereafter when initialized with that state. The period of a PRNG is defined as the maximum over all starting states of the length of the repetition-free prefix of the sequence. The period is bounded by the size of the state, measured in bits. However, since the length of the period potentially doubles with each bit of 'state' added, it is easy to build PRNGs with periods long enough for many practical applications.

32-bit values), and runs faster than other statistically reasonable generators [Marsland, 2011].

For DWID-Bench, the following piece of pseudocode is assumed as the PRNG, generating uniformly distributed 32-bit integers in the range [0, $2^{32} - 1$] with the MT19937 algorithm (withdrawn from an original code listing written by Matsumoto and Mishimura and available at http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html):

```
/*
   Extracted from a C-program for MT19937, with initialization improved
   2002/1/26, coded by Takuji Nishimura and Makoto Matsumoto.

   Before using, initialize the state by using init_genrand(seed).

   Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
   All rights reserved.

   Any feedback is very welcome.
   http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
   email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)
*/

#include <stdio.h>

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0dfUL   /* constant vector a */
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */

static unsigned long mt[N]; /* the array for the state vector  */
static int mti=N+1; /* mti==N+1 means mt[N] is not initialized */

/* initializes mt[N] with a seed */
void init_genrand(unsigned long s)
{
    mt[0]= s & 0xffffffffUL;
    for (mti=1; mti<N; mti++) {
        mt[mti] =
            (1812433253UL * (mt[mti-1] ^ (mt[mti-1] >> 30)) + mti);
        /* See Knuth TAOCP Vol2. 3rd Ed. P.106 for multiplier. */
        /* In the previous versions, MSBs of the seed affect   */
        /* only MSBs of the array mt[].                        */
        /* 2002/01/09 modified by Makoto Matsumoto             */
        mt[mti] &= 0xffffffffUL;
        /* for >32 bit machines */
    }
}

/* initialize by an array with array-length */
/* init_key is the array for initializing keys */
/* key_length is its length */
/* slight change for C++, 2004/2/26 */
```

```
void init_by_array(unsigned long init_key[], int key_length)
{
    int i, j, k;
    init_genrand(19650218UL);
    i=1; j=0;
    k = (N>key_length ? N : key_length);
    for (; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1664525UL))
          + init_key[j] + j; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++; j++;
        if (i>=N) { mt[0] = mt[N-1]; i=1; }
        if (j>=key_length) j=0;
    }
    for (k=N-1; k; k--) {
        mt[i] = (mt[i] ^ ((mt[i-1] ^ (mt[i-1] >> 30)) * 1566083941UL))
          - i; /* non linear */
        mt[i] &= 0xffffffffUL; /* for WORDSIZE > 32 machines */
        i++;
        if (i>=N) { mt[0] = mt[N-1]; i=1; }
    }

    mt[0] = 0x80000000UL; /* MSB is 1; assuring non-zero initial array */
}

/* generates a random number on [0,0xffffffff]-interval */
unsigned long genrand_int32(void)
{
    unsigned long y;
    static unsigned long mag01[2]={0x0UL, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    if (mti >= N) { /* generate N words at one time */
        int kk;

        if (mti == N+1)   /* if init_genrand() has not been called, */
            init_genrand(5489UL); /* a default initial seed is used */

        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1UL];

        mti = 0;
    }

    y = mt[mti++];

    /* Tempering */
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return y;
```

```
}

/* generates an integer random number on [0, x[ */
long random(long x)
{
    return trunc(genrand_int32()*(1.0/4294967296.0)*x);
}



/* EXAMPLE OF USAGE - Generate first 10 random numbers in [0, 100[ */
/* 123456789 used as the initial seed */
int main(void)
{
    int i, x=100;
    unsigned long s=123456789;
    init_genrand(s);
    printf("10 random outputs in [0, 100[ \n");
    for (i=0; i<10; i++) {
      printf(random(x));
      printf("\n");
    }
    return 0;
}
```

The *random* function based on the Mersenne Twister should be used the following way:

- For each benchmark run, the PRNG should be reinitialized using seed 123456789 (execute `function init_genrand(123456789)`);
- Given *random(x)*, where *x* represents a fixed integer value, the function result should be a randomized number belonging to range [0…*x*-1];
- Given *random(x)*, where *x* represents a list of values, the function result should be one of those values randomly chosen from the list.

All random values should be generated sequentially for all random parameters of the previous user workload, before moving on to generate the random values for the random parameters of the next user workload, *i.e.*, the random values should be sequentially generated for the complete set of random parameters (R_TABLE, R_INDEX, ..., P_VALUE2) in the parameters' order, for user 1, and then moving on to user 2, and so on and so forth.

The complete list of defined used random parameter variable values is:

List of Random Parameter Variables

DEFINE R_TABLE = random('Store_sales', 'Time_dim', 'Date_dim', 'Customer', 'Item', 'Store ', 'Customer_address', 'Customer_demographics', 'Household_demographics', 'Promotion', 'Income_band')

DEFINE R_INDEX = random(select index_name from dba_indexes where table_name = [D_TABLE])

DEFINE ITEM_K = random(select max(i_item_sk) from item)

DEFINE ITEM_N = random(select distinct i_product_name from item)

DEFINE RD_DOM = random(14)+1

DEFINE RD_MOY = random(12)+1

DEFINE RD_YEAR = random(6)+1998

DEFINE CA_TYPE = random('ca_state', 'ca_county', 'ca_city')

DEFINE CA_VALUE = random(select distinct [SS_CATYPE] from customer_address)

DEFINE CA_STATE = random(select distinct ca_state from customer_address)

DEFINE I_TYPE = random('i_brand', 'i_class', 'i_category')

DEFINE I_VALUE = random(select distinct [SS_ITYPE] from item)

DEFINE SS_COLUMN = random('ss_wholesale_cost', 'ss_list_price', 'ss_salesprice', 'ss_ext_discount_amt', 'ss_ext_sales_price', 'ss_ext_wholesale_cost', 'ss_ext_list_price', 'ss_ext_tax', 'ss_coupon_amt', 'ss_net_paid', 'ss_net_paid_inc_tax', 'ss_net_profit')

DEFINE SS_VALUE = random(select max([SS_COLUMN]) from store_sales)

DEFINE SS_TICKET = random(select max(ss_ticket_number) from store_sales)

DEFINE SS_ITEM_T = random(select ss_item_sk from store_sales where ss_ticket_number = SS_TICKET)

DEFINE SS_SDATE = random(select max(d_date_sk) from date_dim)

DEFINE SS_STIME = random(select max(t_date_sk) from time_dim)

DEFINE SS_SITEM = random(select max(i_item_sk) from item)

DEFINE SS_SCUST = random(select max(c_customer_sk) from customer)

DEFINE SS_SCDEMO = random(select max(cd_demo_sk) from customer_demographics)

DEFINE SS_SHDEMO = random(select max(hd_demo_sk) from household_demographics)

DEFINE SS_SADDR = random(select max(ca_address_sk) from customer_address)

DEFINE SS_SSTORE = random(select max(s_store_sk) from store)

DEFINE SS_SPROMO=random(select max(p_promo_sk) from promotion)

```
DEFINE SS_STICK = (select max(ss_ticket_number) from store_sales)+1

DEFINE SS_ITICK = random(select ss_item_sk from store_sales where ss_ticket_number =
[SS_STICK])

DEFINE SS_QUANTITY = random(99)+1

For i = 1 to 12
    DEFINE SS_VALUES[i] = random(9999999)/100
Next

DEFINE I_COLUMN = random('i_current_price','i_wholesale_cost')

DEFINE I_VALUE_2 = random(select max[I_COLUMN] from item)

DEFINE P_COLUMN_1 = random('p_start_date_sk', 'p_end_date_sk', 'p_item_sk',
'p_cost')

DEFINE P_VALUE_1 = random(select max([P_COLUMN_1]) from promotion)

DEFINE PROMO_K = random(select max(p_promo_sk) from promotion)

DEFINE P_COLUMN_2 = random('p_start_date_sk', 'p_end_date_sk', 'p_cost')

DEFINE P_VALUE_2 = random(select max([P_COLUMN_2]) from promotion)
```

The complete list of proposed intrusion actions that represent the "intruder" workload is as follows, composed by 34 SQL instructions (Intrusion Action IA01 to IA34.

Intrusion Action IA01. Query for retrieving information on the tables and columns of the database schema.

```
select table_name, column_name, data_type from user_tab_columns;
```

Intrusion Action IA02. Deleting an index from the database.

```
drop index [D_INDEX];
```

Intrusion Action IA03. Deleting the fact table.

```
drop table store_sales;
```

Intrusion Action IA04. Retrieving all data from the fact table.

```
select * from store_sales;
```

Intrusion Action IA05. Query flooding by requesting several joins on all data from the fact table to be processed and returned.

```
select * from
    (select * from store_sales) a, (select * from store_sales) b, (select * from store_sales) c,
    (select * from store_sales) d, (select * from store_sales) e, (select * from store_sales) f,
    (select * from store_sales) g, (select * from store_sales) h, (select * from store_sales) i,
    (select * from store_sales) j;
```

Intrusion Action IA06. Query retrieving all sales, date, item and customer data for all sales of a given item.

```
select * from store_sales, item, customer, date_dim
where ss_item_sk = [ITEM_K] and ss_item_sk = i_item_sk and
        ss_customer_sk = c_customer_sk and ss_sold_date = d_date_sk;
```

Intrusion Action IA07. Query retrieving all sales, item and date data for all sales in a random period of two weeks.

```
select * from store_sales, item, date_dim
where ss_sold_date_sk = d_date_sk and
        d_year = [RD_YEAR] and d_moy = [RD_MOY] and
        d_dom >= [RD_DOM] and d_dom <= [RD_DOM]+14
```

Intrusion Action IA08. Query retrieving all sales, customer address, date and item data for all sales in a given state, county or city.

```
select store_sales.*, customer_address.*, item.*, d_year, d_moy, d_dom
from store_sales, customer_address, item, date_dim
where ss_addr_sk = ca_address_sk and
        ss_item_sk = i_item_sk and  ss_sold_date_sk = d_date_sk and
        [CA_TYPE] = [CA_VALUE];
```

Intrusion Action IA09. Query retrieving all sales, item, date and customer address data for all sales of a given item in a given state in a random period of two weeks.

```
select store_sales.*, item.*, d_year, d_moy, d_dom, customer_address.*
from store_sales, date_dim, item, customer_address
where ss_item_sk = i_item_sk and
        i_product_name = [ITEM_N] and  ss_sold_date_sk = d_date_sk and
        d_year = [RD_YEAR] and d_moy = [RD_MOY] and
        d_dom >= [RD_DOM] and d_dom <= [RD_DOM]+14 and
        ss_addr_sk = ca_address_sk and ca_state = [CA_STATE];
```

Intrusion Action IA10. Query retrieving all sales, item and date data for all sales of all items of a given brand, class or category.

```
select * from store_sales, item, date_dim
where ss_item_sk = i_item_sk and ss_sold_date_sk = d_date_sk and
```

```
[I_TYPE] = [I_VALUE];
```

Intrusion Action IA11. Query retrieving the total quantity and total value of a given sales column, per item, for all items.

```
select ss_item_sk, i_product_name,sum(ss_quantity),sum([SS_COLUMN])
from store_sales, item
where ss_item_sk = i_item_sk
group by ss_item_sk;
```

Intrusion Action IA12. Query retrieving the total quantity and total value of a given sales column as well as the row count of those sales, for a given item.

```
select ss_item_sk, i_product_name, sum(ss_quantity), sum([SS_COLUMN]), count(*)
from store_sales, item
where ss_item_sk = i_item_sk and ss_item_sk = [ITEM_K];
```

Intrusion Action IA13. Query retrieving the total value of a given sales column as well as the row count of those sales, per day, in a given period of two weeks.

```
select d_year, d_moy, d_dom, sum([SS_COLUMN]), count(*)
from store_sales, date_dim
where ss_sold_date_sk = d_date_sk and
      d_year = [RD_YEAR] and d_moy = [RD_MOY] and
      d_dom >= [RD_DOM] and d_dom <= [RD_DOM]+14
group by d_year, d_moy, d_dom
order by d_year, d_moy, d_dom;
```

Intrusion Action IA14. Query retrieving the total value of a given sales column as well as the row count of those sales, per city per month, for a given state, county or city.

```
select ca_city, d_year, d_moy, sum([SS_COLUMN], count(*)
from store_sales, customer_address, date_dim
where ss_addr_sk=ca_address_sk and ss_sold_date_sk=d_date_sk and
      [CA_TYPE]=[CA_VALUE]
group by ca_city, d_year, d_moy
order by ca_city, d_year, d_moy;
```

Intrusion Action IA15. Query retrieving the total quantity and total value of a given sales column as well as the row count of those sales, for a given item in a given state, per city per day, in a given period of two weeks.

```
select ca_city, ca_county, ca_state, ss_item_sk, i_product_name, d_year,
      d_moy, d_dom, sum(ss_quantity), sum([SS_COLUMN]), count(*)
from store_sales, date_dim, customer_address, item
where ss_item_sk = i_item_sk and i_product_name = [ITEM_N] and
```

```
        ss_sold_date_sk = d_date_sk and
        d_year = [RD_YEAR] and d_moy = [RD_MOY] and
        d_dom >= [RD_DOM] and d_dom <= [RD_DOM]+14
        ss_addr_sk = ca_address_sk and ca_state = [CA_STATE]
group by ca_city, d_year, d_moy, d_dom
order by ca_city, d_year, d_moy, d_dom;
```

Intrusion Action IA16. Query retrieving the total value of a given sales column for all sales of a given brand, class or category, per city per month.

```
select [I_TYPE], ca_city, d_year, d_moy, sum(R_COLUMN)
from store_sales, item, customer_address, date_dim
where ss_item_sk = i_item_sk and [I_TYPE] = [I_VALUE] and
        ss_addr_sk = ca_address_sk and ss_sold_date_sk = d_date_sk
group by ca_city, d_year, d_moy
order by ca_city, d_year, d_moy;
```

Intrusion Action IA17. Modifying the values of a given sales column for all the sales rows of a certain item.

```
update store_sales set [SS_COLUMN] = [SS_VALUE] where ss_item_sk = [ITEM_K];
```

Intrusion Action IA18. Modifying the values of a given sales column for all the sales rows of a certain item belonging to a certain ticket number.

```
update store_sales set [SS_COLUMN] = [SS_VALUE]
where ss_ticket_number = [SS_STICK] and ss_item_sk = [SS_ITICK];
```

Intrusion Action IA19. Modifying the values of a given sales column for all the sales rows of a certain state, county or city.

```
update store_sales set [SS_COLUMN] = [SS_VALUE]
where (select count(*) from customer_address where
        ss_addr_sk = ca_address_sk and [CA_TYPE] = [CA_VALUE])>0;
```

Intrusion Action IA20. Modifying the values of a given sales column for all the sales rows of a certain brand, class or category.

```
update store_sales set [SS_COLUMN] = [SS_VALUE]
where (select count(*) from item where ss_item_sk = i_item_sk and
        [I_TYPE]=[I_VALUE])>0;
```

Intrusion Action IA21. Modifying the values of a given sales column for all the sales rows of a certain day.

```
update store_sales set [SS_COLUMN] = [SS_VALUE]
```

```
where (select count(*) from date_dim where ss_sold_date_sk = d_date_sk
        and d_year = [RD_YEAR] and d_moy = [RD_MOY] and
        d_dom = [RD_DOM])>0;
```

Intrusion Action IA22. Deleting all the sales rows of a certain item.

```
delete from store_sales where ss_item_sk = [ITEM_K];
```

Intrusion Action IA23. Deleting all the sales rows of a certain item belonging to a certain ticket number.

```
delete from store_sales
where ss_ticket_number = [SS_STICK] and ss_item_sk = [SS_ITICK];
```

Intrusion Action IA24. Deleting all the sales rows of a certain state, county or city.

```
delete from store_sales where (select count(*) from customer_address where
ss_addr_sk=ca_address_sk and [CA_TYPE] = [CA_VALUE])>0;
```

Intrusion Action IA25. Deleting all the sales rows of a certain brand, class or category.

```
delete from store_sales where (select count(*) from item where
                ss_item_sk = i_item_sk and [I_TYPE] = [I_VALUE])>0;
```

Intrusion Action IA26. Deleting all the sales rows of a certain day.

```
delete from store_sales where (select count(*) from date_dim where
        ss_sold_date_sk=d_date_sk and d_year=[RD_YEAR] and
        d_moy=[RD_MOY] and d_dom=[RD_DOM])>0;
```

Intrusion Action IA27. Inserting false data in the store sales fact table.

```
insert into store_sales (*) values (SS_SDATE, SS_STIME, SS_SITEM, SS_SCUST,
    SS_SCDEMO, SS_SHDEMO, SS_SADDR, SS_SSTORE, SS_SPROMO, SS_STICK,
    SS_QUANTITY, SS_VALUE[1], SS_VALUE[2], SS_VALUE[3], SS_VALUE[4],
    SS_VALUE[5], SS_VALUE[6], SS_VALUE[7], SS_VALUE[8], SS_VALUE[9],
    SS_VALUE[10], SS_VALUE[11], SS_VALUE[12]);
```

Intrusion Action IA28. Retrieving all data from any table in the database.

```
select * from [R_TABLE];
```

Intrusion Action IA29. Retrieving the most sensitive customer data from all customer tables the database.

```
select * from customer, customer_address, customer_demographics
where c_current_addr_sk = ca_address_sk and c_current_cdemo_sk = cd_demo_sk;
```

Intrusion Action IA30. Retrieving a portion of sensitive customer data from all customers belonging to a given state, county or city.

```
select c_customer_sk, c_first_name, c_last_name, c_birth_day, c_birth_month,
       c_birth_year, c_email_address, customer_address.*, customer_demographics.*
from customer, customer_address, customer_demographics
where c_current_addr_sk = ca_address_sk and
    c_current_cdemo_sk = cd_demo_sk and [CA_TYPE] = [CA_VALUE];
```

Intrusion Action IA31. Retrieving the data of all promotions concerning a given item on a given month.

```
select promotion.*, item.*, d_year, d_moy, d_dom
from promotion, item, date_dim
where p_item_sk = i_item_sk and
       i_product_name = [ITEM_N] and p_start_date_sk = d_date_sk and
      d_year = [RD_YEAR] and d_moy = [RD_MOY];
```

Intrusion Action IA32. Modifying the current price or wholesale cost of a given item.

```
update item set [I_COLUMN] = [I_VALUE_2]
where i_product_name = [ITEM_N];
```

Intrusion Action IA33. Modifying the start date, end date, item or cost of a given promotion.

```
update promotion set [P_COLUMN_1] = [P_VALUE_1]
where p_promo_sk = [PROMO_K];
```

Intrusion Action IA34. Modifying the start date, end date, or cost of all promotions of a given item.

```
update promotion set [P_COLUMN_2] = [P_VALUE_2]
where p_item_sk = [ITEM_K]
```

Table D-2 resumes the user types that may execute each instruction, the action class and affected security dimensions, as well as the tables targeted to be affected by the instruction. From observing the table it can be seen that each *DBA "intrusion" workload* is composed by all 34 intrusion actions, the *ETL "intrusion" workload* is defined by 28 intrusion actions (all except IA03, IA22, IA23, IA24, IA25 and IA26), and the *DW end user "intrusion" workload* is defined by 18 intrusion actions (all intrusion actions that can be executed by "Any" user type). The definition of the number of streams each type of user should be running for each benchmark run will be described in the next section.

The chosen instructions that compose the intruder actions were guided by the assumption that each table has its own relative sensitivity, given the importance and business knowledge revealed by its contents. Obviously, the *Store_sales* fact table is much more sensitive (and therefore, more important from the intruder's perspective) than the *Date_dim* dimension table, since the first stores the operational secrets of the business and the second just serves as support for temporal definitions of the business. Thus, the majority of the defined intrusion actions were designed for targeting actions against the most important tables (which, for the store sales DW, concern the tables that store sales, items, promotions and customer information, namely tables *Store_sales*, *Item*, *Customer*, *Customer_address*, *Customer_demo* and *Promotion*).

Table D-2. "Intrusion" Workload

			TARGET TABLES										
Intrusion Action	**SQL Action Class**	**User Type**	**Store_ sales (facts)**	**Customer (dim)**	**Item (dim)**	**Promotion (dim)**	**Date_dim (dim)**	**Time_dim (dim)**	**Store (dim)**	**Customer_ address (dim)**	**Customer_ demo (dim)**	**Household _demo (dim)**	**Income_ band (dim)**
IA01	A	Any	X	X	X	X	X	X	X	X	X	X	X
IA02	B	ETL, DBA	X	X	X	X	X	X	X	X	X	X	X
IA03	B	DBA	X										
IA04	C	Any	X										
IA05	F	Any	X										
IA06	D	Any	X	X	X		X						
IA07	D	Any	X		X		X						
IA08	D	Any	X		X		X			X			
IA09	E	Any	X		X		X			X			
IA10	D	Any	X		X		X						
IA11	D	Any	X		X								
IA12	E	Any	X		X								
IA13	E	Any	X				X						
IA14	E	Any	X				X			X			
IA15	E	Any	X		X		X			X			
IA16	E	Any	X		X		X			X			
IA17	H	ETL, DBA	X										
IA18	H	ETL, DBA	X										
IA19	H	ETL, DBA	X										
IA20	H	ETL, DBA	X										
IA21	H	ETL, DBA	X										
IA22	I	DBA	X										
IA23	I	DBA	X										
IA24	I	DBA	X										
IA25	I	DBA	X										
IA26	I	DBA	X										
IA27	G	ETL, DBA	X										
IA28	C	Any	X	X	X	X	X	X	X	X	X	X	X
IA29	C	Any		X						X	X		
IA30	D	Any		X						X	X		
IA31	D	Any			X	X	X						
IA32	H	ETL, DBA			X								
IA33	H	ETL, DBA				X							
IA34	H	ETL, DBA				X							

The *Date_dim* dimension table is also often used in the "intrusion" action instructions; however, it is a static table, *i.e.*, it has fixed content and does not change over time. Furthermore, its content does not reveal any business information nor does it require external knowledge to be regenerated. Therefore, it can be easily and quickly rebuilt in case the content is damaged and is not so important as those previously mentioned.

Table D-3 shows the order in which each intrusion action should be executed for each user "intrusion" workload stream. The number of intrusion actions in each benchmark run ranges from 28+18+34 = 80 (for a setup composed by 1 "Intrusion" ETL User + 1 "Intrusion" DW End User + 1 "Intrusion" DBA User) to 28+180+34 = 242 (for a setup composed by 1 "Intrusion" ETL User + 10 "Intrusion" DW End Users + 1 "Intrusion" DBA User).

Table D-3. "Intrusion" Workload – Query Ordering

Sequence Order	ETL User	DW End Users										DBA User
		1	2	3	4	5	6	7	8	9	10	
1	IA02	IA05	IA07	IA31	IA10	IA16	IA28	IA07	IA29	IA05	IA28	IA01
2	IA05	IA16	IA28	IA13	IA04	IA06	IA14	IA12	IA01	IA10	IA08	IA28
3	IA09	IA15	IA06	IA15	IA15	IA31	IA29	IA29	IA04	IA08	IA09	IA14
4	IA33	IA07	IA04	IA09	IA09	IA08	IA30	IA09	IA08	IA31	IA13	IA03
5	IA17	IA06	IA13	IA16	IA08	IA07	IA13	IA14	IA15	IA12	IA29	IA22
6	IA34	IA01	IA05	IA30	IA28	IA04	IA04	IA05	IA12	IA29	IA01	IA19
7	IA14	IA10	IA30	IA05	IA13	IA13	IA01	IA08	IA11	IA01	IA15	IA05
8	IA28	IA12	IA14	IA28	IA30	IA28	IA08	IA28	IA31	IA14	IA04	IA25
9	IA16	IA28	IA08	IA06	IA06	IA11	IA09	IA13	IA30	IA06	IA06	IA34
10	IA04	IA29	IA12	IA12	IA29	IA30	IA12	IA06	IA09	IA07	IA31	IA06
11	IA31	IA31	IA01	IA08	IA01	IA15	IA06	IA31	IA13	IA30	IA11	IA12
12	IA01	IA30	IA31	IA29	IA12	IA01	IA10	IA10	IA05	IA11	IA07	IA24
13	IA21	IA09	IA16	IA01	IA11	IA05	IA05	IA04	IA14	IA16	IA12	IA31
14	IA13	IA04	IA10	IA04	IA31	IA10	IA07	IA11	IA06	IA09	IA05	IA16
15	IA06	IA14	IA11	IA11	IA16	IA09	IA15	IA16	IA07	IA04	IA16	IA15
16	IA18	IA11	IA09	IA07	IA05	IA12	IA16	IA15	IA28	IA15	IA30	IA20
17	IA30	IA13	IA15	IA10	IA14	IA29	IA31	IA30	IA16	IA13	IA14	IA23
18	IA07	IA08	IA29	IA14	IA07	IA14	IA11	IA01	IA10	IA28	IA10	IA33
19	IA27											IA32
20	IA08											IA17
21	IA32											IA21
22	IA15											IA09
23	IA20											IA11
24	IA29											IA30
25	IA12											IA08
26	IA11											IA10
27	IA10											IA27
28	IA19											IA18
29												IA26
30												IA29
31												IA02
32												IA04
33												IA07
34												IA13

D.5. DWID-Bench Rules and Execution Procedure

In this section we define the rules for implementing the DWID-Bench setup and its execution procedure. The rules for implementing the benchmark are the following:

- The store sales data schema should be implemented exactly as described in the TPC-DS benchmark;
- The database maintenance routines should run exactly as described in TPC-DS, representing the "non-intrusion" ETL workload streams. Each of these ETL streams may execute concurrently with DW End User streams or DBA streams, or alone. The "non-intrusion" ETL streams do not overlap; all operations need to have finished on "non-intrusion" ETL workload x before any procedure can start on behalf of "non-intrusion" ETL workload x+1. The first refresh data set can only start after 3**S* (where S represents the number of running "non-intrusion" DW end user query streams) "non-intrusion" queries have completed their execution. Each subsequent refresh set can start after completion of an additional 64 queries (the total number of instructions in two complete workloads). The purpose of linking data maintenance operations to completion of queries is so that the updates are interspersed among execution of queries in the benchmark runs, although concurrent execution of updates and queries is not required;
- Each "non-intrusion" query instruction should be exactly as described in the TPC-DS benchmark, while each "non-intrusion" instruction should be exactly as defined in Table D-1 (including instruction modification and the substitution of query parameters for both types of workloads);
- The same hardware and software should be used during the complete benchmark run without changes. The only allowed changes are those concerning the updating of both DW and DIDS databases and logs;
- The DIDS cannot be specifically optimized *a priori* for the set of SQL actions defined in the intrusion workload, *i.e.*, it may not know or take in account information regarding previous knowledge of the

intrusion workloads before the workloads' execution in the benchmark run;

- Each stream should be run only once, to avoid repeating instruction ordering;
- The driver system shall submit "intrusion" and "non-intrusion" workloads through one or more sessions on the SUT. Each session corresponds to one stream composed by a complete "intrusion" or "non-intrusion" user workload;
- If any of the workloads fails to execute, the benchmark results are invalid.

The DWID-Bench benchmark is defined by the execution of the *Training Phase*, followed by the *Testing Phase*. The *Training Phase* includes all activity required to bring the SUT to the configuration that immediately precedes the execution of the "non-intrusion" and "intrusion" workloads that will measure the intrusion detection and performance metrics of the DIDS, which composes the *Testing Phase*. For fairness of the database performance measures, the database server should be restarted before starting the *Testing Phase*, in order to reinitialize the database cache. The benchmark methodology is shown in Figure D-3. The *Training Phase* includes:

1) The execution of all SQL DDL commands that create the store sales DW data schema (datafiles, tables and views) and constraints, as well as any performance optimization objects (*e.g.* indexes);
2) The execution of all data loading procedures to populate the DW with the initial data defined by TPC-DS for the chosen scale factor as defined in that benchmark;
3) During the execution of the two previous steps, the DIDS can access and analyze the executed operations to build the "normal" ETL and/or DBA user profiles, in any way, if needed;
4) The execution of one to five "non-intrusion" ETL data maintenance workload streams as the first one to five refresh sets as defined in TPC-DS and following the rules previously presented in this section, and one to ten DW End User "non-intrusion" workload streams, for allowing the DIDS to build the "normal" non-intruder ETL and DW end user profiles, in any way.

The *Testing Phase* includes:

1) The execution of the same number of "non-intrusion" ETL and "non-intrusion" DW End User workload streams as those used in the Training Phase;

2) The execution of one "intrusion" DBA stream, one to ten "intrusion" DW End User streams, and one "intrusion" ETL stream, concurrently with the "non-intrusion" workloads.

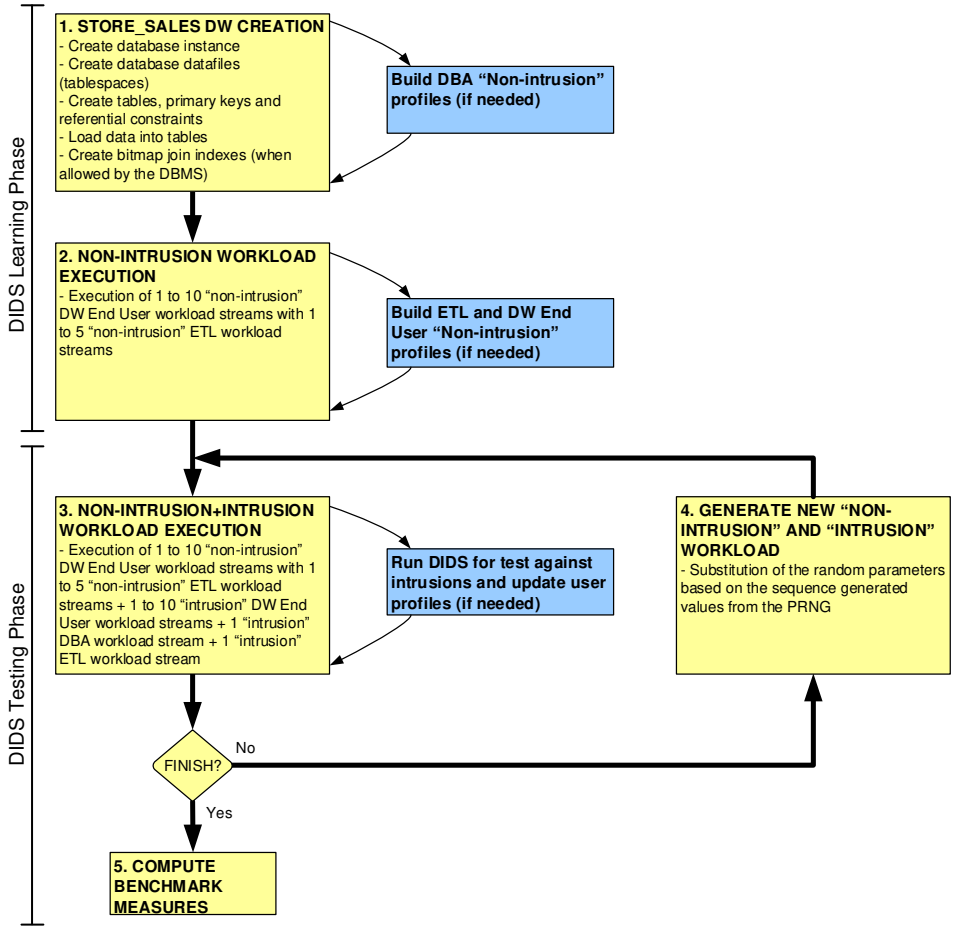


Figure D-3. DWID-Bench benchmark methodology

Figure D-4 illustrates the execution sequence of the *Testing Phase*. Note that the "non-intrusion" ETL workload is executed as defined in TPC-DS, with the only difference that it refreshes the database after completing the

processing of a group of 64 queries instead of 192 (because the complete DWID-Bench "non-intrusion" workload has 32 queries, instead of 99 as defined in TPC-DS; 64 is an approximate proportional number).

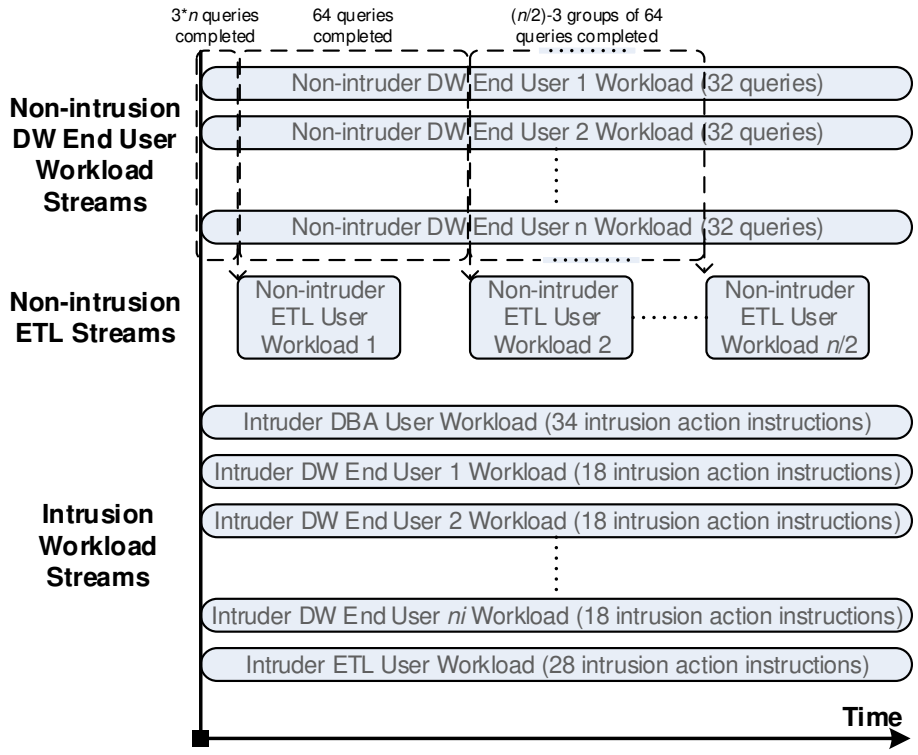


Figure D-4. Benchmark Testing Phase execution flow for *n* "non-intrusion" DW End Users and *ni* "intrusion" DW End Users

The following section defines the benchmark's metrics.

D.6. DWID-Bench Metrics

To evaluate the overall efficiency of a DIDS in a data warehousing environment, we propose focusing on the following aspects concerning intrusion detection in DWs:

- The efficiency of the intrusion detection processes themselves, *i.e.*, their ability to effectively detect intrusion actions (true positives) and minimize the number of false alarms (false positives), and minimize the number of intrusions that pass undetected (false negatives);

- How quickly after an intrusion action occurs is the DIDS able to produce an alert, given that in many cases it is critical to detect an intrusion as quickly as possible, before it may damage the DW;
- The ability of the DIDS to evolve by improving its intrusion detection efficiency through time.

Given this, in DWID-Bench we define the Data Warehouse Intrusion Detection Benchmark Coefficient ($DWIDB_{Coef}$) metric, which involves two main components that respectively measure a DIDS' efficiency and speed in intrusion detection time, where *ne* represents the number of benchmark runs, $F\text{-}score_i$ the F-score[10] obtained by the DIDS in each benchmark run, $\Delta t_{QWorkloads}$ the total execution time (in seconds) of the "non-intrusion" and "intrusion" workloads of all benchmark runs, and $\Delta t_{IDProcesses}$ the total execution time (in seconds) of the DIDS of all benchmark runs:

$$DWIDB_{Coef} = \underbrace{\frac{\sum_{i=1}^{ne} i * F\text{-}score_i}{\sum_{i=1}^{ne} i}}_{\text{Evaluates the intrusion detection efficiency through time, giving higher weight to the most recent F-scores}} * \underbrace{\frac{\Delta t_{QWorkloads}}{\Delta t_{QWorkloads} + \Delta t_{IDProcesses}}}_{\text{Evaluates the impact of the time taken to execute the intrusion detection processes}} * 100$$

Given its expression, $DWIDB_{Coef}$ will output a real value in the range [0...100]. A higher benchmark value indicates a better DIDS. To illustrate the outcome of the proposed metric, consider the following values shown in Table D-4 as fictional examples of three DIDS to be evaluated by DWID-Bench.

Table D-4. DWID-Bench DIDS benchmarking examples

	1st Benchmark Run (*ne* = 1)				2nd Benchmark Run (*ne* = 2)			
	$F\text{-}score_1$	Δt_{QWork}	Δt_{IDProc}	$\boldsymbol{DWIDB_C}$	$F\text{-}score_2$	Δt_{QWork}	Δt_{IDProc}	$\boldsymbol{DWIDB_C}$
DIDS 1	60%	1000	200	**50.0**	80%	2000	400	**61.1**
DIDS 2	70%	1000	200	**58.3**	70%	2000	400	**58.3**
DIDS 3	70%	1000	250	**56.0**	60%	2000	500	**50.7**

[10] The F-score measure was explained in Chapter 6, Subsection 6.6.2.

Observing the table, it can be seen that after the first benchmark run, DIDS 2 and DIDS 3 are those presenting the highest intrusion detection efficiency, *i.e.*, they have higher F-score than DIDS 1, but since DIDS 2 takes less time in its intrusion detection processes than DIDS 3 it outputs a higher benchmark value, making it the best DIDS after the first benchmark run. Moreover, although DIDS 1 executes its intrusion detection processes faster than DIDS 3, this last DIDS presents a higher intrusion detection efficiency with an F-score that overcomes the fact that it is slower.

However, after the second benchmark run, and assuming that they all take the same time in execution as the first benchmark run, DIDS 1 improves its intrusion detection efficiency to an F-score of 80%, which allows it to improve its benchmark value to a measure that makes it the best solution. And DIDS 1 is in fact the best solution after both benchmark runs, since its F-score average and running times are the same as DIDS 2, but its most recent intrusion detection efficiency has the best results of all DIDS. On the other hand, the fact that DIDS 3 presented worse results in the second benchmark run has made it the worst DIDS.

Therefore, the $DWIDB_{Coef}$ results shown in Table D-4 demonstrate that the benchmark metric is indeed able to track the efficiency of the intrusion detection processes and its evolution, along with the ability to also measure the impact of the required time spent by those processes.

D.7. Discussion

The proposed benchmark abstracts the diversity of the described classes of possible intrusion actions, while retaining custom normal user activity and DW environment requirements. As it is necessary to execute a large number of queries and data maintenance operations to completely manage any business analysis environment, no benchmark can succeed in exactly mimicking a particular environment and remain broadly applicable. We acknowledge that the definition and implementation of benchmarks is not a trivial task and that there are always discussable issues concerning the objectivity and effectiveness of each proposal. However, in DWID-Bench we have tried to provide a wide coverage of possible intrusion activity in DWs, while simulating their execution in a realistic-like data warehousing environment. Given the importance of intrusion detection in DWs and the

lack of both DIDS at the SQL level as available packages supplied by DBMS vendors as well as standard benchmarks to test them, we believe that the issues presented in this appendix are worthy of notice and hope that our work may drive the discussion around the subject in both the benchmarking and intrusion detection research communities, and possibly make way for a standardized benchmark for this purpose.

D.8. Summary and Future Work

In this appendix we have proposed a novel benchmark that focuses on evaluating DIDS at the SQL command level in DW environments. The proposed metrics provide an objective and comprehensive mean of evaluating the intrusion detection efficiency and ability to improve, as well as the impact on database response time, of proposed DIDS for DWs. The benchmark's implementation procedures and metrics also comply with the principles of comprehensibility and reproducibility required in benchmarking proposals.

While this benchmark offers a representative scenario of possible intrusion attacks on DWs, it does not reflect the entire range of possibilities. As future work, we intend to increase and develop the "intrusion" workload for widening the coverage of possible intrusion actions and therefore produce more thorough tests.