Database Encryption – An Overview of Contemporary Challenges and Design Considerations

Erez Shmueli
Deutsche Telekom Laboratories; and the Department of Information Systems Engineering, Ben-Gurion University.
Beer Sheva, Israel
erezshmu@bgu.ac.il

Ronen Vaisenberg
School of Computer Science, University of California.
Irvine, CA, USA[1]
ronen@uci.edu

Yuval Elovici
Deutsche Telekom Laboratories; and the Department of Information Systems Engineering, Ben-Gurion University.
Beer Sheva, Israel
elovici@bgu.ac.il

Chanan Glezer
Deutsche Telekom Laboratories at Ben-Gurion University.
Beer Sheva, Israel
chanan@bgu.ac.il

ABSTRACT
This article describes the major challenges and design considerations pertaining to database encryption. The article first presents an attack model and the main relevant challenges of data security, encryption overhead, key management, and integration footprint. Next, the article reviews related academic work on alternative encryption configurations pertaining to encryption locus; indexing encrypted data; and key management. Finally, the article concludes with a benchmark using the following design criteria: encryption configuration, encryption granularity and keys storage.

Categories and Subject Descriptors
H.2.7 [**Database Management**]: Database Administration - *Security, integrity and protection.*

General Terms
Security

Keywords
Database Encryption, Security, Privacy.

1. INTRODUCTION
Conventional database security solutions and mechanisms are divided into three layers; physical security, operating system security and DBMS (Database Management System) security [1]. With regard to the security of stored data, access control (i.e., authentication and authorization) has proved to be useful, as long as that data is accessed using the intended system interfaces. However, access control is useless if the attacker simply gains access to the raw database data, bypassing the traditional mechanisms. This kind of access can easily be gained by insiders, such as the system administrator and the database administrator (DBA).

The aforementioned layers are therefore not sufficient to guarantee the security of a database when database content is kept in a clear-text, readable form. One of the advanced measures being incorporated by enterprises to address this challenge of private data exposure, especially in the banking, financial, insurance, government, and healthcare industries, is *database encryption*. While database-level encryption does not protect data from all kinds of attacks, it offers some level of data protection by ensuring that only authorized users can see the data, and it protects database backups in case of loss, theft, or other compromise of backup media.

In this survey, we focus on the academic work and propose a design-oriented framework which can be used by native and 3rd party DB encryption providers as well as DBAs and corporate IS developers.

2. ATTACK MODEL AND CHALLENGES
A database encryption scheme should meet several requirements. Among them are the requirements for data security, high performance, and detection of unauthorized modifications [2]. Inspired by that pioneer work in the field, we adopt these requirements and add several requirements that relate to the practicality of such an encryption solution. Each requirement will be discussed in details in the following subsections.

[1] Research performed while at the Department of Information Systems Engineering, Ben-Gurion University

2.1 Database Security – Models and Attacks

2.1.1 Database operational model

As with current database systems, when discussing the model for database encryption we assume a client-server scenario. The client has a combination of sensitive and non-sensitive data stored in a database at the server. Whether or not the two parties are co-located does not make a difference in terms of security. The server's added responsibility is to protect the client's sensitive data, i.e., to ensure its confidentiality and its integrity.

This model has three major points of vulnerability with respect to client data:

(1) Data-in-motion - All client-server communication can be secured through standard means, e.g., an SSL connection, which is the current de facto standard for securing Internet communication. Therefore, communication security poses no real challenge and we ignore it in the remainder of this paper.

(2) Data-in-use - An adversary can access the memory of the database software directly and extract sensitive information. This attack can be prevented using a tampered proof hardware for protecting the database server's memory, and therefore is also ignored in the remainder of this paper.

(3) Data-at-rest - Typically, DBMSs protect stored data through access control mechanisms. However, its goals should not be confused with those of data confidentiality since attacks against the stored data may be performed by accessing database files following a path other than through the database software, by physical removal of the storage media or by access to the database backup files.

Different security mechanisms can be categorized based on the level of trust in the database server, which can range from fully trusted to fully untrusted:

(1) Fully trusted - In this scenario, the server can perform all of the operations and no threat exists. Obviously this scenario is not of our interest, and is ignored in the remainder of this paper.

(2) Fully un-trusted - In this scenario, a client does not even trust the server with clear text queries; hence, it involves the server performing encrypted queries over encrypted data. This scenario corresponds to the Database as a Service (DAS) model.

(3) Partially trusted – The database server itself together with its memory and the DBMS software is trusted, but the secondary storage is not.

In our literature review we will categorize the different schemes based on their trust in the database server.

2.1.2 Attacks compromising security

An attacker can be categorized into three classes [3]:

(1) Intruder - A person who gains access to a computer system and tries to extract valuable information.

(2) Insider - A person who belongs to the group of trusted users and tries to get information beyond his own access rights.

(3) Administrator - A person who has privileges to administer a computer system, but uses his administration rights in order to extract valuable information.

2.1.2.1 Passive attacks

According to [4], a secure index in an encrypted database should not reveal any information on the database plaintext values. We extend this requirement, by categorizing the possible information leaks:

(1) Static leakage - Gaining information on the database plaintext values by observing a snapshot of the database at a certain time. For example, if the database is encrypted in a way that equal plaintext values are encrypted to equal ciphertext values, statistics about the plaintext values, such as their frequencies can easily be learned.

(2) Linkage leakage - Gaining information on the database plaintext values by linking a table value to its position in the index. For example, if the table value and the index value are encrypted the same way (both ciphertext values are equal), an observer can search the table cipher text value in the index, determine its position and estimate its plaintext value.

(3) Dynamic leakage - Gaining information about the database plaintext values by observing and analyzing the changes performed in the database over a period of time. For example, if a user monitors the index for a period of time, and if in this period of time only one value is inserted (no values are updated or deleted), the observer can estimate its plaintext value based on its position in the index.

2.1.2.2 Active attacks

In addition to the passive attacks that observe the database, active attacks that modify the database should also be considered. Active attacks are more problematic in the sense that they may mislead the user. Unauthorized modifications can be made in several ways [5]:

(1) Spoofing - Replacing a ciphertext value with a generated value. Assuming that the encryption keys are secure, a possible attacker might try to generate a valid ciphertext value, and substitute the current valid value stored on the disk. Assuming that the encryption keys were not compromised, this attack poses a relatively low risk.
(2) Splicing - Replacing a ciphertext value with a different cipher text value. Under this attack, the encrypted content from a different location is copied to a new location under attack.
(3) Replay - Replacing a cipher text value with an old version previously updated or deleted.

Note that each of the above attacks is highly correlated to the leakage vulnerabilities discussed before: static leakage and spoofing, linkage leakage and splicing and dynamic leakage and replay attack.

2.2 Encryption Overhead

Added security measures typically introduce significant computational overhead to the running time of general database operations. However, it is desirable to reduce this overhead to the minimum that is really needed, and thus:

(1) It should be possible to encrypt only sensitive data while keeping insensitive data unencrypted.
(2) Only data of interest should be encrypted/decrypted during queries' execution.
(3) Some vendors do not permit encryption of indexes, while others allow users to build indexes based on encrypted values. The latter approach results in a loss of some of the most obvious characteristics of an index - range searches, since a typical encryption algorithm is not order-preserving.
(4) In addition, it is desirable that the encrypted database should not require much more storage than the original one.

2.3 Integration Footprint

Incorporating an encryption solution over an existing DBMS should be easy to integrate, namely, it should have:

(1) Minimal influence on the application layer
(2) Minimal influence on the DBA work
(3) Minimal influence on the DBMS architecture

2.4 Handling Encryption Keys

The way encryption keys are being used can have a significant influence on both the security of the database and the practicality of the solution. The following issues should be considered:

(1) Cryptographic Access Control – Encrypting the whole database using the same key, even if access control mechanisms are used is not enough. For example, an insider who has the encryption key and bypasses the access control mechanism can access data that are beyond his security group. Encrypting objects from different security groups using different keys ensures that a user who owns a specific key can decrypt only those objects within his security group [6].
(2) Secure Key Storage – Encryption keys should be kept securely, e.g., storing the keys inside the database server allows an intruder access to both the keys and the encrypted data, and thus encryption is worthless.
(3) Key Recovery – If encryption keys are lost or damaged, the encrypted data is worthless. Thus, it should be possible to recover encryption keys whenever needed.

3. ALTERNATIVE CONFIGURATIONS

A large body of work exists in the field of database encryption. Related work can be generally categorized into four main classes: file system encryption, DBMS encryption, application level encryption and client side encryption. Related work also deals with indexing encrypted data, and keys' management.

3.1 File-System Encryption

The encryption scheme presented in [7] suggests encrypting the entire physical disk allowing the database to be protected. The main disadvantage of this scheme is that the entire database is encrypted using a single encryption key, and thus discretionary access control cannot be supported.

3.2 DBMS-Level Encryption

Several database encryption schemes have been proposed in the literature. The one presented in [8] is based on the Chinese-Reminder theorem, where each row is encrypted using different sub-keys for different cells. This scheme enables encryption at the level of rows and decryption at the level of cells. Another scheme, presented in [2], extends the encryption scheme presented in [8], by supporting multilayer access control. It classifies subjects and objects into distinct security classes that are ordered in a hierarchy, such that an object with a particular security class can be accessed only by subjects in the same or a higher security class. The scheme presented in [9] proposes encryption for a database based on Newton's interpolating polynomials.

The database encryption scheme presented in [10] is based on the RSA public-key scheme and suggests two database encryption schemes: one column oriented and the other row oriented. One disadvantage of all the above schemes is that the basic element in the database is a row and not a cell, thus the structure of the database is modified. In addition, all of those schemes require re-encrypting the entire row when a cell value is modified. Thus, in order to perform an update operation, all the encryption keys should be available. The SPDE scheme which [11] encrypts each cell in the database individually together with its cell coordinates (table name, column name and row-id). In this way static leakage attacks are prevented since equal plaintext values are encrypted to different cipher-text values. Furthermore, splicing attacks are prevented since each cipher-text value is correlated with a specific location, trying to move it to a different location will be easily detected. Further security analysis and fixes to this scheme can be found in [12].

3.3 Application-Level Encryption

In [13] a Web Data Service Provider Middleware (WDSP) application is suggested which translates the user queries into a new set of queries which execute of the encrypted DBMS. The model was implemented as the DataProtector[1] System which serves as an http-level rule-based middleman who regulates access to secure data stored on web service provider. The solution is attractive to public data storage, backup and sharing services which are very popular on the web nowadays.

3.4 Client-Side Encryption

The recent explosive increase in Internet usage, together with advances in software and networking, has resulted in organizations being able to easily share data for a variety of purposes. This has led to a new paradigm termed "Database as a Service" (DAS) [3, 14] in which the whole process of database management is outsourced by enterprises to reduce costs and to concentrate on the core business.

One fundamental problem with this architecture (besides performance degradation due to remote access to data) is data privacy. That is, sensitive data have to be securely stored and protected against untrustworthy servers. Encryption is one promising solution to this problem.

Defining the encryption scheme under the assumption that the server is not trusted, raises the question of how a query is evaluated if data are encrypted and the server has no access to the encryption keys [15].

3.5 Indexing Encrypted Data

The indexing scheme proposed in [16] suggests encrypting the whole database row and assigning a set identifier to each value in this row. The indexing scheme in [17] suggests building a B-Tree index over the table plaintext values and then encrypting the table at the row level and the B-Tree at the node level. The indexing scheme in [18] is based on constructing the index on the plaintext values and encrypting each page of the index separately. Since the uniform encryption of all pages is likely to provide many cipher breaking clues, the indexing scheme provided in [16] proposes encrypting each index page using a different key depending on the page number. However, in these schemes, it is not possible to encrypt different portions of the same page using different keys.

The indexing scheme suggested in [19] enables the server to search for pre-defined keywords within a document using a special trapdoor supplied by the user for that keyword. The encryption function suggested in [20] preserves order, and thus allows range queries to be directly applied to the encrypted data without decrypting it. In addition it enables the construction of standard indexes on the cipher-text values. However, the order of values is sensitive information in most cases and should not be exposed. The encryption scheme provided in [15] suggests computing the bitwise exclusive or (XOR) of the plaintext values with a sequence of pseudo-random bits generated by the client according to the plaintext value and a secure encryption key.

In addition to table encryption, the SPDE scheme that is presented in [11] offers a novel method for indexing encrypted columns. However this method is very limited and is extended in [4] in order to solve elementary problems such as unauthorized modifications and discretionary access control. Further analysis and fixes to this scheme can be found in [13].

3.6 Keys' Management

Many techniques for generating encryption keys were mentioned in the literature; however, most of them are neither convenient nor flexible in the real applications. The scheme in [21] and its extension in [22] propose a novel database encryption scheme for enhanced data sharing inside a database, while preserving data privacy. In this scheme, a pair of keys is generated for each user. The key pair is separated when it is generated. The private key is kept by user at the client end, while the public key is kept in the database server.

[1] www.ics.uci.edu/~projects/dataprotector

4. CONCLUSIONS

Based on our review, Table 1 compares several database encryption deployment configurations. To summarize, the best flexibility is achieved when the encryption is made inside the DBMS. File-System encryption, even though being easy to deploy, does not allow using different encryption keys and does not allow choosing which data to encrypt/decrypt and thus have a significant influence on both data security and performance.

Table 2 summarizes the influence of encryption granularity on several aspects. Better performance and preserving the structure of the database cannot be achieved using page or whole table encryption granularity. However, special techniques can be used, in order to cope with unauthorized modifications and information leakage, when single values or record/node granularity encryption is used.

Table 1. Comparing Different Database Encryption Configurations.

	File-System Encryption	DBMS Encryption	Application Encryption	Encryption at the Client Side
Finest encryption granularity supported	Page	Cell	Cell	Cell
Support for internal DBMS mechanisms (e.g. index, foreign key...).	+	+	-	-
Support for cryptographic access control	-	+	+	+
Performance	Best	Medium	Low	Worst
Compatibility with legacy applications	+	+	-	-

Table 3 summarizes the dependency between the trust in the server and the keys' storage. If we have no trust in the database server, we would prefer to keep the encryption keys only at the client side. In cases where the database server itself is fully trusted, but its physical storage is not, we can store the keys at the server side in some protected region.

Table 2. Risk in Different Levels of Encryption Granularities.

	Information Leakage	Unauthorized Modifications	Structure Perseverance	Performance
Single Values	Worst	Worst	Best	Best
Record/ Nodes	Low	Low	Medium	Medium
Pages	Medium	Medium	Low	Low
Whole	Best	Best	Worst	Worst

Our survey indicates that sophisticated and robust database encryption features are available in both the academia and commercial worlds [23], however, their adoption by clients is still lagging because of practical constraints such as cost of deployment and performance overhead. In order for such advanced features to be widely adopted the aforementioned criteria need to be given top consideration by database encryption researchers and developers.

Table 3. Keys Storage Options and Trust in Server.

	Server Side	Keys per Session	Client Side
Absolute	+	+	+
Partial	-	+	+
None	-	-	+

5. ACKNOWLEDGMENTS

This research was supported by Deutsche Telekom AG.

6. REFERENCES

[1] Fernandez EB, Summers RC, Wood C (1980) Database Security and Integrity. Addison-Wesley, Massachusetts.

[2] Min-Shiang H, Wei-Pang Y (1997) Multilevel Secure Database Encryption with Subkeys. Data and Knowledge Engineering 22, 117-131.

[3] Bouganim L, Pucheral P (2002) Chip-secured data access: confidential data on untrusted servers. The 28th Int. Conference on Very Large Data Bases, Hong Kong, China, pp. 131-142.

[4] Elovici Y, Waisenberg R, Shmueli E, Gudes E (2004) A Structure Preserving Database Encryption Scheme. SDM 2004, Workshop on Secure Data Management, Toronto, Canada, August.

[5] Vingralek R (2002) Gnatdb: A small-footprint, secure database system. The 28th Int'l Conference on Very Large Databases, Hong Kong, China, August, pp. 884-893.

[6] Bertino E, Ferrari E (2002) Secure and Selective Dissemination of XML Documents. ACM Transactions on Information and System Security, 5(3), 290-331.

[7] Kamp PH (2003) GBDE – GEOM based disk encryption Source. BSDCon '03, pp. 57-68.

[8] Davida GI, Wells DL, Kam JB (1981) A Database Encryption System with subkeys. ACM Trans. Database Syst. 6, 312-328.

[9] Buehrer D, Chang C (1991) A cryptographic mechanism for sharing databases. The International Conference on Information & Systems. Hangzhou, China, pp. 1039-1045.

[10] Chang C, Chan CW (2003) A Database Record Encryption Scheme Using RSA Public Key Cryptosystem and Its Master Keys. The international conference on Computer networks and mobile computing.

[11] Shmueli E, Waisenberg R, Elovici Y, Gudes E (2005) Designing secure indexes for encrypted databases. Proceedings of Data and Applications Security, 19th Annual IFIP WG 11.3 Working Conference, USA.

[12] Kühn U (2006) Analysis of a Database and Index Encryption Scheme – Problems and Fixes. Secure Data Management.

[13] Merhotra S, Gore B (2009) A Middleware approach for managing and of outsourced personal data, NSF Workshop on Data and Application Security, Arlignton, Virginia, February 2009.

[14] Hacigümüs H, Iyer B, Li C, Mehrotra S (2002) Executing SQL over encrypted data in the database-service-provider model. The ACM SIGMOD'2002, Madison, WI, USA.

[15] Song DX, Wagner D, Perrig A (2000) Practical Techniques for Searches on Encrypted Data. The 2000 IEEE Security and Privacy Symposium, May.

[16] Bayer R, Metzger JK (1976) On the Encipherment of Search Trees and Random Access Files. ACM Trans Database Systems, 1, 37-52.

[17] Damiani E, De Capitani diVimercati S, Jajodia S, Paraboschi S, Samarati P (2003) Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. CCS03, Washington, pp. 27-31.

[18] Iyer B, Mehrotra S, Mykletun E, Tsudik G, Wu Y (2004) A Framework for Efficient Storage Security in RDBMS. E. Bertino et al. (Eds.): EDBT 2004, LNCS 2992, pp. 147-164.

[19] Boneh D, Crescenzo GD, Ostrovsky R, Persiano G (2004) Public Key Encryption with Keyword Search. Encrypt 2004, LNCS 3027. pp. 506-522.

[20] Agrawal R, Kiernan J, Srikant R, Xu Y (2004) Order Preserving Encryption for Numeric Data. The ACM SIGMOD'2004, Paris, France.

[21] He J, Wang M (2001) Cryptography and Relational Database Management Systems, Proceedings of IEEE Symposium on the International Database Engineering & Applications, Washington, DC, USA.

[22] Chen G, Chen K, Dong J (2006) A Database Encryption Scheme for Enhanced Security and Easy Sharing. CSCWD'06, IEEE Proceedings, IEEE Computer Society, Los Alamitos. CA, pp. 1-6.

[23] The Forrester Wave: Database Encryption Solutions, Q3 2005.