Database Security Technique with Database Cache

Rashmi Parab
Yadavrao Tasgaonkar Institute of Engineering And Technology, Mumbai, Maharashtra
rashmi.9288@gmail.com

Nilima Nikam
Yadavrao Tasgaonkar Institute of Engineering And Technology, Mumbai, Maharashtra
nilima.nikam@tasgaonkartech.com

Abstract: Today people are depending more on the corporate data for decision making, management of customer service and supply chain management etc. Any loss, corrupted data or unavailability of data may seriously affect its performance. The database security should provide protected access to the contents of a database and should preserve the integrity, availability, consistency, and quality of the data in this paper, we analyze and compare five traditional architectures for database encryption. We show that existing architectures may provide a high level of security, but have a significant impact on performance and impose major changes to the application layer, or may be transparent to the application layer and provide high performance, but have several fundamental security weaknesses. We suggest a sixth novel architecture that was not considered before. The new architecture is based on placing the encryption module inside the database management software (DBMS), just above the database cache, and using a dedicated technique to encrypt each database value together with its coordinates.

Keywords: Authentication, Relational Database, Security, Integrity, Protection, DBMS, Blowfish.

I. INTRODUCTION

Typically, database management systems (DBMSs) protect stored data through access control mechanisms. However, an access control mechanism by itself is insufficient to protect stored data since it can be bypassed in a number of ways: by accessing database files following a path other than through the DBMS (e.g., an intruder who infiltrates the information system and tries to mine the database footprint on disk); by physical removal of the storage media; or by accessing the database backup files. Another source of threats comes from the fact that many databases are today outsourced to Database Service Providers (DSP). In such instances, data owners have no other choice than to trust DSP's who claim that their systems are fully secured and their employees are beyond any suspicion, assertions frequently repudiated by facts. Finally, a database administrator (DBA) may have sufficient privileges to tamper with the access control definition. An old and important principle called defense in depth involves multiple layers of a security control such that attackers must get through layer after layer of defense. In this context, encryption, which can complement and reinforce access control, has recently received much attention from the database community.

The purpose of database encryption is to ensure database opacity by keeping the information hidden to any unauthorized person (e.g., intruder). Even if attackers get through the firewall and bypass access control mechanisms, they still need the encryption keys to decrypt data. Encryption can provide strong security for data at rest, but developing a database encryption solution must take many factors into consideration. In this paper, we discuss the complexities associated with implementing a database encryption solution.

The new architecture is based on placing the encryption module inside the Database Management Software (DBMS), just above the database cache, and using a dedicated technique to encrypt each database cell value together with its coordinates. These two properties allow our new architecture to achieve a high level of data security while maintaining high performance and total transparency to the application layer. We also explain how each of the database-level architectures (i.e., SQL interface, storage engine, operating system and above cache) can be implemented in a commercial, open source, DBMS.

II.   LAYERS OF SECURITY

Database encryption solutions can be categorized based on their level of trust in the database server, encryption granularity and layer in which encryption takes place.

A.   Operating System

In this layer, pages are encrypted/decrypted by the operating system when they are written/read from disk. This layer has the advantage of being totally transparent, thus avoiding any changes to the DBMS and to existing applications. Furthermore, encryption in this layer is relatively resistant to information leakage and unauthorized modifications as a large number of database objects are encrypted in one chunk. However it suffers from several fundamental problems: (1) Since the operating system has no knowledge of database objects and their internal structure, it is impossible to encrypt different parts of the page using different encryption keys (e.g., when those parts belong to users with different authorizations) and thus cryptographic access control cannot be supported. (2) It is not possible to encrypt specific portions of the database and leave others in their plaintext form. Furthermore, not only relevant data is decrypted during a query execution since each access requires the decryption of an entire page. Therefore selective encryption is very limited. (3) The DBA cannot perform any administrative task (e.g., dropping a column) without possessing the encryption keys. (4) The database cache, which usually contains a large amount of disk page copies for improving performance, is kept in its plaintext form and is thus vulnerable to data-in-use attacks.

B.   Storage Engine

Similarly to the operating system layer, pages in this layer are encrypted/decrypted when they are written/read from disk. However, as opposed to the operating system layer, encryption/decryption operations are performed by the DBMS, at the cell-level granularity. In other words, each time a page is loaded from disk, all encrypted values in that page are decrypted (each one separately), and each time a page is stored to disk, all sensitive values in that page are encrypted (again, each one separately). However, although the use of cell-level encryption granularity allows different values within a page to be encrypted using different keys when a page is read from the disk into the database cache, the whole page must be decrypted, even if the initiating user does not have the authorization to access all values in that page. Moreover, the fact that each time a page is written/read from disk, multiple encryption/decryption operations is performed, may degrade performance substantially, compared to the single encryption/decryption operation per page in the operating system layer. Note that encryption in this layer is located beneath the query execution engine and is therefore transparent to the query execution engine and all layers above it (including the application).

C.   SQL Interface

In this layer, data is encrypted using predefined stored procedures, views and triggers. While encryption in this layer is easy to implement and does not usually require significant changes to the application layer, it has the following limitations: (1) encryption takes place above the query execution engine, and thus some database mechanisms (e.g., indexes and foreign keys1) may not function properly; (2) the use of stored procedures entails a context switch from SQL to the stored procedure language which usually has a high negative impact on performance; (3) those mechanisms (namely: triggers, views and stored procedures) can be disabled by a malicious DBA.

D.   Application

In this layer, sensitive data is encrypted in the application layer before it is sent to the database and decrypted before usage. It supports the highest degree of freedom in terms of enforcing cryptographic access control. However, it suffers from the following disadvantages: (1) modifying legacy applications may require a lot of resources (i.e., time and money); (2) as encryption takes place above the query execution engine If different keys are used to encrypt the primary key and the foreign key. Different database mechanisms cannot function properly and need to be re-implemented by the application; (3) it re-invents the wheel for each new application that is being developed

E.   Client

This layer may promise the highest degree of security since the only one that is able to access the sensitive data is the legitimate client. However, it implies limiting the ability of the database server to process the encrypted data and in extreme cases, to use the database server for storage only

All of the discussed architectures (in above section) use either page-level or cell-level encryption granularities. Although these two granularities are the most commonly used, other granularities were also studied.

III.   CACHE LEVEL SECURITY

In this model, the pages in the DBMS cache are identical copies of pages on the disk. Since encryption in this layer is performed at the cell granularity level, accessing a single value in the encrypted cache requires only its decryption, without the need to decrypt other values.

A promising solution to this problem is using tamper-resistant hard-ware as means of protecting the database memory. However, developing and analyzing such a solution is out of the scope of this paper, especially since this problem is common to all database-level encryption solutions.

The position of a cell in the database is unique and can be identified using the triplet that includes its table, row and column IDs. We refer to this triplet as the \cell coordinates". In the suggested scheme, the cell coordinates are joined to the plaintext value before being encrypted.

Blowfish is an encryption algorithm that can be used as a replacement for the DES or IDEA algorithms. It is symmetric (that is, a secret or private key) block cipher that uses a variable-length key, from 32 bits to 448 bits, making it useful for both domestic and exportable use.

Advantage:
•   These two properties allow our new architecture to achieve a high level of data security while maintaining high performance and total transparency to the application layer.
•   Supports fine-grained cryptographic access control
•   Greatly improves performance since it allows to encrypt/decrypt only relevant portions of the pages when a query is executed.
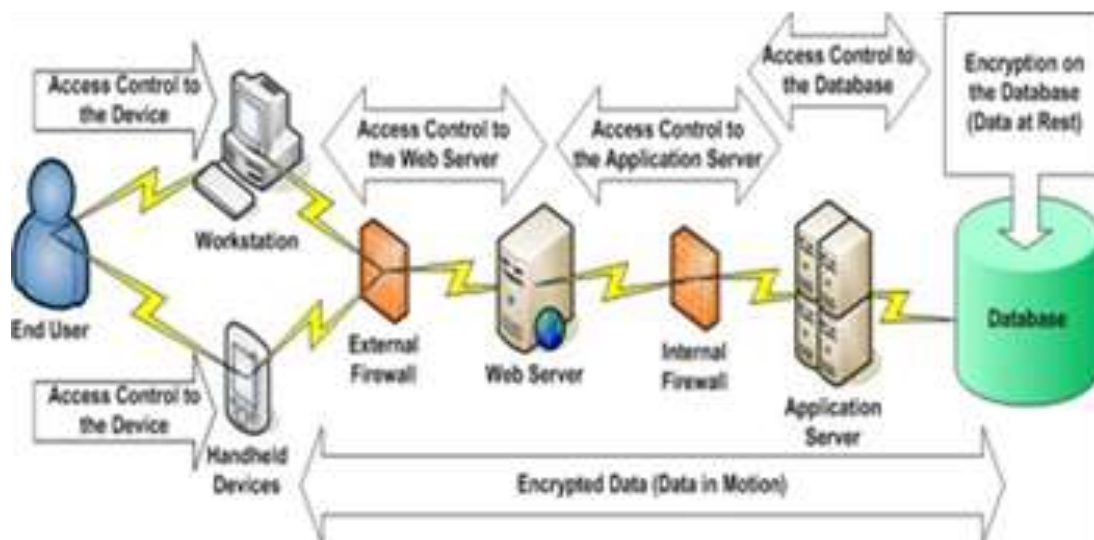

Fig. 1 System Architecture

IV.   IMPLEMENTATION
We describe our implementation of the four database-level encryption architectures in MySQL. The SQL interface architecture, as explained below, was implemented by adding dedicated view and trigger for encryption and decryption operations. The other three architectures were implemented in MySQL by modifying its source code. In many cases, implementing database encryption that way isn't possible since DBMSs are usually closed source. Modifying MySQL in order to support database encryption might seem like a \mega-project" at first. However, the modular design of MySQL enabled us to perform this task fairly quickly and with minimal changes. The sole part of the DBMS that has to be changed is the storage engine. All other layers above the storage engine continue to function as before. MySQL supports different storage engines and, for this research, we modified the InnoDB storage engine. Other storage engines can be modified

in a similar way. Implementing a database encryption based on the SQL interface architecture is described as. The main stages include: (1) renaming the sensitive table; (2) encrypting the values in the sensitive table; (3) defining an encryption trigger; and (4) defining a decryption view. In theory, this architecture should be transparent to the application layer. However, this is not the case in practice because: (1) some operations (e.g., insert, update, truncate) cannot be performed on the view and need to be rewritten to use the renamed table2. (2) range queries are not supported.

In this module is an important module to detect a super point in the network, First get all hash bits from the input string. Merge all the hash key one by one and then Use Random IP function to generate IP address of each node.
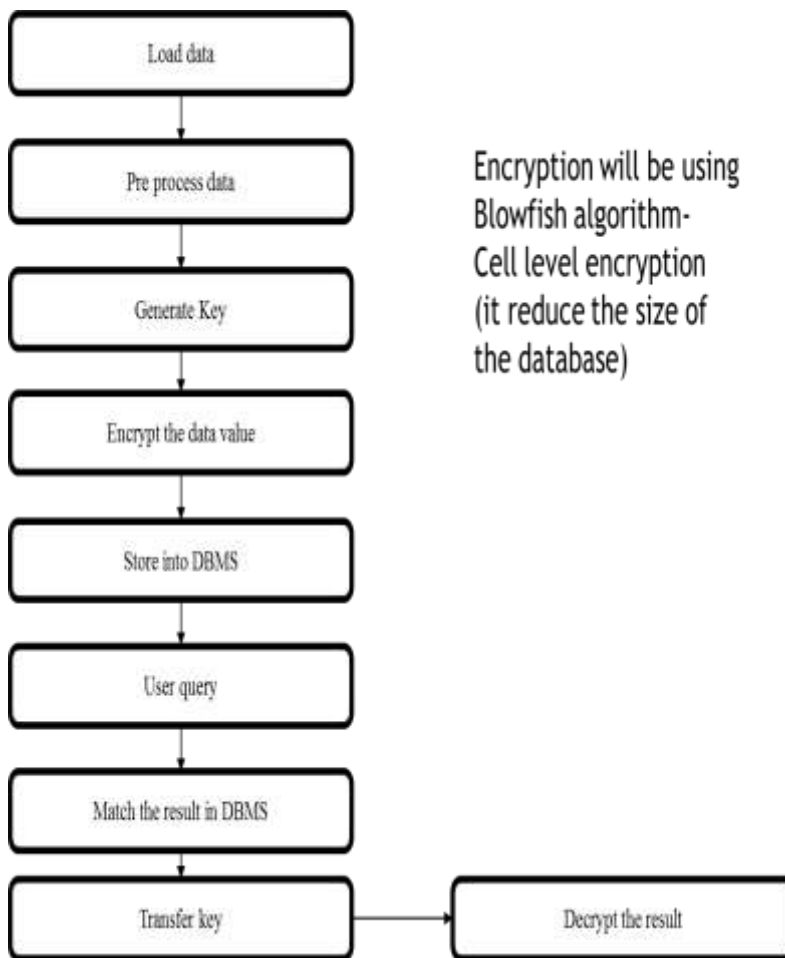
Fig. 2 Implementation Flow

A. Performance Analysis

We implemented the operating system, storage engine, and above cache architectures by modifying the source code of MySQL. All of our experiments also included the SQL interface architecture. However, since it produced extremely high degradation factors (i.e., higher than 100) compared to the no-encryption version of MySQL, which was included in MySQL source code, with 128 bits keys and the cipherblock chaining (CBC) mode. Key initialization was performed once for each encryption/decryption operation (i.e., either once for a cell or once for a page, depending on the architecture). Encryption of values in the above cache architecture utilized their cell-coordinates, where the μ function was implemented as SHA-1(tkrkc). For both of the above cache and the storage engine architectures, we padded sensitive values before encryption so that their size would become a multiple of 128 bits. The operating system architecture did not require any padding since the size of pages was already a multiple of 128 bits. All experiments were conducted on an Intel Core i7 CPU 2.67 GHz personal computer with 4 GB of RAM, running Ubuntu 12.04 LTS.

B. Analytic Results

In order to keep our analysis as simple as possible, we make the following three assumptions: (1) the size of a ciphertext database value equals its size before encryption; (2) all database values have the same size and (3) all database values are encrypted using Blowfish. Given a query Q, we denote its overall execution time when using the without encryption architecture as TQ we; the number of database values that were accessed during the execution of Q as N; the number of database values that required disk access as Ndisk; the number of values that were accessed directly from the database cache as Ncache = N - Ndisk; the time it takes to decrypt a single database value as Tvalue and the time it takes to decrypt a whole database page as Tpage.

The overhead of the operating system architecture compared to the without encryption architecture is in the decryption of each database page that is loaded from disk into the cache. Therefore the execution time of Q using the operating system architecture can be written as:

$$TQos = TQwe + Ndisk \text{ - } Tpage = TQwe + (N \text{ - } Ncache) \_ Tpage$$

Similarly, since the overhead of the above cache architecture compared to the without encryption architecture is in the decryption of each database value that is accessed during the query, the execution time of Q using the above cache architecture can be written as:

$$TQ\ ac = TQwe + N\ -Tvalue$$

Then, the difference between execution times is: TQos – Tqac = (Twe + (N - Ncache) _ Tpage) - (Twe + N - Tvalue)= N - Tpage - Ncache - Tpage - N - Tvalue > 0

Written differently, we have: N * (Tpage - Tvalue) > Ncache * Tpage

Ncache/N < (Tpage - Tvalue)=Tpage Ncache=N < 1 - Tvalue=Tpage

Meaning that for a values hit-ratio, Ncache/N, lower than 1-Tvalue/Tpage, the above cache architecture would outperform the operating system architecture. In order to quantify the ratio Tvalue/Tpage in our system, we measured the time taken to decrypt 109 single database values (assuming their size to be 16B) compared to the time taken to decrypt 109 whole pages (16KB in our MySQL configuration), and found the ratio between these two quantities to be roughly 1/315. (As explained earlier although 16B=16Kb = 1=1024, decrypting one large chunk of data is faster than decrypting it in parts.) Stated differently, as long as the values hit-ratio, Ncache/N, is lower than 1 − 1/315 = 0:9968, the above cache architecture should outperform the operating system approach. Since under these assumptions, all database values require encryption, the storage engine architecture would necessarily be outperformed by the operating system architecture.

V. CONCLUSION

We began by outlining the challenges that face any database encryption solution. Then, we showed how existing architectures may provide a high level of security, but have a high impact on performance and impose major changes to the application layer, or may be transparent to the application layer and provide high performance, but have several fundamental security weaknesses.

We also suggested a novel architecture for database encryption, which is based on placing the encryption module inside the Database Management Software (DBMS), just above the database cache, and using a dedicated technique to encrypt each database cell value together with its coordinates.

These two properties allow our new architecture to achieve a high level of data security while maintaining high performance and total transparency to the application layer. Finally, we showed through analytic and empiric evaluation of performance that in most realistic scenarios

REFERENCES

1. Abdulrahman Hamed Almutairi &Abdulrahman Helal Alruwaili,"Security in Database Systems", Volume 12, Issue 17, Version 1.0, the Year 2012, ISSN: 0975-4172.
2. Tarun Narayan Shankar, G. Sahoo, "Cryptography with Elliptic Curves", International Journal of Computer Science and Applications Vol. 2, No. 1, April / May 2009, ISSN: 0974-1003.
3. Erez Shmueli, Ronen Vaisenberg, Yuval Elovici, Chanan Glezer, "Database Encryption – An Overview of Contemporary Challenges and Design Considerations", SIGMOD Record, September 2009 (Vol. 38, No. 3).
4. H. Hacigumus, B. Iyer, C. Li, S. Mehrotra, Executing SQL over encrypted data in the database-service-provider model, Proceedings of the 2002 ACM SIGMOD International conference on Management of data (2002) 216-227.
5. Erez Shmueli, Ronen Vaisenberg, Ehud Gudes, Yuval Elovic, "Implementing a database encryption solution, design, and implementation issues". Computers & Security, Volume 44, July 2014, Pages 33-50