System of end-to-end symmetric database encryption

V V Galushka, A R Aydinyan, O L Tsvetkova, V A Fathi and D V Fathi

Don State Technical University, 1, Gagarina Sq., Rostov-on-Don, 344000, Russian Federation

Abstract. The article is devoted to the actual problem of protecting databases from information leakage, which is performed while bypassing access control mechanisms. To solve this problem, it is proposed to use end-to-end data encryption, implemented at the end nodes of an interaction of the information system components using one of the symmetric cryptographic algorithms. For this purpose, a key management method designed for use in a multi-user system based on the distributed key representation model, part of which is stored in the database, and the other part is obtained by converting the user's password, has been developed and described. In this case, the key is calculated immediately before the cryptographic transformations and is not stored in the memory after the completion of these transformations. Algorithms for registering and authorizing a user, as well as changing his password, have been described, and the methods for calculating parts of a key when performing these operations have been provided.

1. Introduction
Almost all modern information systems use databases as the main tool for storing large amounts of information, which gives advantages both in fast standardized data processing using the SQL, and in the possibility of organizing effective shared access to them [1, 2]. At the same time, for information security, they usually rely on built-in security tools of some database management system (DBMS) [3]. They always include mechanisms for access control and user rights distribution [4], as well as, possibly, specific tools, for example, masking dynamic data [5] or stored procedures encryption in Microsoft SQL Server [6].

However, the listed methods require the presence of one or more administrators having both full access to the database and the ability to disable any configured security mechanisms. This operation feature is a potential vulnerability that can lead to data leakage, and therefore the actual problem is the development of an effective method of protecting databases based on cryptographic methods in order to reduce the number of ways for obtaining unauthorized access to information [7].

2. End-to-end encryption
End-to-end encryption with regard to databases is the method of data transfer and storage, in which only users who are involved in the exchange have access to them. Using cryptographic keys, end-to-end encryption technology ensures that data are controlled directly by the users, and neither interceptors, nor servers storing data, or their administrators can decrypt messages [8].

Encryption does not cancel the need to control access to data, but it increases protection due to reduction of possible data loss, even if the access control system is bypassed [9]. For example, if the
computer with the installed database was incorrectly configured and an attacker could obtain confidential data, the stolen information would be useless if it had been previously encrypted.

As is known, all encryption algorithms are separated into symmetric and asymmetric ones. Each of the groups has its advantages and disadvantages that determine the scope of the algorithm. For example, the main advantage of the symmetric encryption in comparison with asymmetric algorithms is the speed of operation due to the simplicity of the operations being performed, as well as the shorter key length to ensure comparable cryptographic strength.

Provided that the amount of data is processed by databases, a critical parameter is the speed of the encryption algorithm, and this means the necessity of using one of the symmetric algorithms. Next, the article describes the use of the AES algorithm, but in the practical implementation of a particular system, any other can replace it.

Symmetric algorithms are characterized by a number of shortcomings associated with the complexity of key management in a large network and their exchange between participants, which follows from the necessity to create a secret channel for the transfer of the keys. This drawback is a potential vulnerability of symmetric encryption algorithms. Thus, reliable and safe application of symmetric encryption methods to protect databases is only possible after the development of the appropriate key management system.

Today among the architectures of information systems, which include databases, the most common ones are client-server and three-tier architectures. Both of them assume that the database will be used at the base level of the system for storing data, adding them and issuing applications on demand, and data processing will be performed either by the application server or by a fat client. With regard to the end-to-end encryption scheme, these two types of software are the endpoints of data processing; therefore, they must perform encryption.

3. Key management technique
To ensure the correct interaction of users who exchange information with the database, that is, to allow one user to decrypt data that were encrypted not only by him but also by another user, they must be provided with one key.

Both in the presented diagram (figure 1) and in practice, there is no separate trusted communication channel for exchanging such information, and the only possibility of interaction of users (clients in the terminology of the client-server architecture) is inserting or selecting information from the database providing a multi-user mode of operation.

![Figure 1. The general scheme of end-to-end symmetric database encryption.](image-url)
However, direct exchange of a key through a database server is not a good solution, as it leads to additional difficulties in solving the problem of secure data transmission between the components of the information system. In general, any key transfer operations are potential sources of leakage and other threats to information security. The same goes for the storage process — as, with the transfer, additional security measures are required. The listed statements lead to the fact that the safest option excludes the storage and transfer of the key, which the database was encrypted with. Such scheme can be realized if the key is calculated "on the fly" as necessary, exists in the computer's memory for a limited period and is destroyed after the encryption or decryption procedure is completed.

Let $K$ be a key of $n$ bits length, and $K_i$ is the $i$-th bit of the key ($i = 0, ..., n-1$).

The encryption system should be transparent both for the user and for the database management system, and its work is fully implemented by the client software. As in any other system, each user must have his own password used to authenticate and determine access rights. Let $P$ be a password of length $m$ bits, and $P_j$ is the $j$-th bit of password $P$ ($j = 0, ..., m-1$).

It should be noted that the values of $n$ and $m$ in the general case are not related to each other in any way. For example, optional password length restrictions are usually defined by a minimum of 8 characters. Each character, depending on the encoding, is 8 or 16 bits, but the key length depends on the encryption algorithm, and for AES it can be 16/24/32 bytes or 128/192/256 bits respectively.

Let us define $D$ as the result of the bitwise exclusive-OR operation over the values of $K$ and $P$. The value of $D$ is the difference between the user's password and the database encryption key, and we will call this value as “difference”. To perform this operation effectively, it is necessary to convert the password and the key to the same length using some conversion method. By way of such transformation, it is proposed to use a hash function that has the property of outputting a result of a certain size regardless of the length of the input data. Since one of the acceptable length for the AES algorithm is 128 bits, the most suitable function for it is md5, the result of which also has a length of 128 bits, unlike, for example, sha1, which, although cryptographically stronger, generates hashes with a length of 160 bits. In this case, the cryptographic strength of the hash function does not matter, since it is used only to equalize the length of the source data.

4. Authorization task

When performing authorization, the password entered by the user is converted into a modified password with the length of 128 bits (16 bytes) according to the formula:

$$P^* = \text{md5}(P).$$

After that, bitwise exclusive-OR operation ($\oplus$) is performed, that is:

$$D_i = P^* \oplus K_i, i = 0, ..., n-1.$$

The result of the calculations is compared with the field “difference” of the current user and, in case of equality, the authentication is considered successful.

If it is necessary to extract (decrypt) the encrypted data from the database, the reverse operation is performed:

$$K_i = P^* \oplus D_i, i = 0, ..., n-1$$

and the resulting key is used for decryption (figure 2).
If one wants to add a new user or change a password to the existing user, one must perform on behalf of the authorized user, having the same rights, a number of computationally simple transformations given by the formula:

$$D'_i = \text{md5}(P') \oplus D_i \oplus \text{md5}(P), \ i = 0,...,n-1,$$

where $D'$ — the “difference” for the new user; $P'$ — the password of the new user; $D$ — the “difference” of the existing user, obtained from the database; $P$ — the password of the current user.

After performing the specified conversions, the value of $D'$ should be written to the cell “difference” of the line corresponding to the current database user in the “users” table.

The user can replace his password, relying on his current password by formula (1), where $D'$ is the future “difference” for the current user; $P'$ — the new password of the current user; $D$ — the current user “difference” obtained from the database; $P$ — current password of the current user.

Separately, it is worth considering the advantages that are manifested in a multi-user access to the database. The above methodology forms as a result a distributed decentralized key management system where the distribution occurs on several layers: 1 — a distributed representation of the key, when one of its parts is stored in the database, and the other one is known by the user, 2 — the main key is distributed among different users, that is, any of them can get it, knowing his password. Therefore, this technique inherits the advantages of distributed systems, such as reliability, local autonomy, and scalability [10]. This approach, unlike storing a fixed key, allows one to restore access to the database while at least one of its users remembers his password.

The other user can change the password of the existing user (e.g. in case when the password is lost), relying on his password using the formula (1), where $D'$ — the new “difference” for the user for whom the password is being changed; $P'$ — the new password of the user; $D$ — the user’s “difference”, based on the password recovery obtained from the database; $P$ — the password of the current user.

5. Conclusions
The proposed method of key management has a number of advantages.

1. The difficulty of gaining access to the encryption key due to the fact that it is possible only at certain stages of the client part of the information system by the way of deep analysis of the main memory dumps, which is a common vulnerability of any program.
2. Reliable protection of stored data provided by the cryptographic strength of encryption algorithms that meet modern standards.

The main drawback of the developed method is the inability to perform queries on the selection of data from the tables with a condition other than checking strict correspondence, over an encrypted column. It can be overcome by partially transferring the encryption/decryption process to the database server when using built-in features for encryption and key management in the SQL language, which is the subject of further research.

References
[1] Grund M, Schapranow M, Krueger J, Schaffner J and Bog A 2008 Shared Table Access Pattern Analysis for Multi-Tenant Applications AMIGE 2008. IEEE Symposium on (Tianjin: IEEE) pp 472–475
[2] Cattel R 2011 Scalable SQL and NoSQL data stores ACM SIGMOD Record vol 39 (New York: ACM) pp 12–27
[3] Samarati P and Capitani de Vimercati S 2001 Access Control: Policies, Models, and Mechanisms FOSAD 2000: Foundations of Security Analysis and Design (Bertinoro: Springer) pp 137–196
[4] Byun J-W and Li N 2008 Purpose based access control for privacy protection in relational database systems The VLDB Journal vol 17 (New York: Springer-Verlag) pp 603–619
[5] Santos R J, Bernardino J and Vieira M 2011 A data masking technique for data warehouses IDEAS ’11 Proceedings of the 15th Symposium on International Database Engineering & Applications (New York: ACM) pp 61–69
[6] Anjugam S and Murugan S 2014 Efficient Method for Preventing SQL Injection Attacks on Web Applications Using Encryption and Tokenization International Journal of Advanced Research in Computer Science and Software Engineering vol 4 (Advanced Research International Publication House) pp 173-177
[7] Goyal V, Pandey O, Sahai A and Waters B 2006 Attribute-based encryption for fine-grained access control of encrypted data CCS ’06 Proceedings of the 13th ACM conference on Computer and communications security (Alexandria, VA: ACM) pp 89–98
[8] Arasu A, Eguro K, Kaushik R and Ramamurthy R 2014 Querying encrypted data SIGMOD ’14 Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (Snowbird, UT: ACM) pp 1259–1261
[9] Ulf T T 2005 Database Encryption - How to Balance Security with Performance SSRN eLibrary
[10] Oleinikova S, Kravets O, Zolotukhina E, Shkurkin D, Kobersy I and Shadrina V 2016 Mathematical and Software of the Distributed Computing System Work Planning on the Multiagent Approach Basis International Journal of Applied Engineering Research vol 11 (Deli: ElDirect) pp 2872–2878