Research on the adaptive weighted mean algorithm for lightweight scheme of database encryption

Xiaoyan Wang[1], Rui Sun[2]

[1] *Henan Polytechnic, Department of Information Engineering, Zhengzhou, 450046, China*

[2] *Henan Polytechnic, Department of Information Engineering, Zhengzhou, 450046, China*

**Corresponding author's e-mail: wangxiaoyanhp@163.com*

Received 11 January 2013, www.cmnt.lv

Abstract

Database security is very important in an information system. Most database management systems (DBMS) use access control to protect database. But access control can't resist bypass threats. Database encryption can resist these threats well. Encrypting data can exist on different positions in a DBMS. This paper compares four database encryption schemes in DBMS and proposes six design criteria for database encryptions. According to these criteria, we implement a prototype in Postgre SQL compliance with the fourth scheme. Our design hasn't impact on other database functions and makes no changes on DBMS structure. So it is a good way to enhance database security especially existing DBMS. Finally, we compare performance of database between with encryption and without encryption. The downgrade of performance is little and tolerable.

Keywords: database encryption; key management: secure storage; adaptive weighted mean algorithm.

1 Introduction

With the development and popularization of computer technology, especially broad applications in important branches of national economy, the problem of computer security has been standing out in the information society.

Information is usually stored and managed in database system, so how to guarantee and strengthen the security and secrecy of database system has been the exigent problem. The security of database system lies on two layers: one is measure of user name/password identification, view, permission control and audit from database system itself, large database systems, such as Oracle, SQL server have these functions. The other is that application systems provide. Generally, basic secure technology from database system is adaptive in generic applications. For applications in important branches and sensitive fields, the above measures are not enough. Some users, especially interior ones can also obtain user name and password illegally, use other methods to enter database exceeding their authority and get or modify information. So it's necessary to encrypt important data in database system.

When more and more information systems are constructed in different application areas, the security of these systems becomes a very important aspect concerned especially in some critical systems. Database is center of most of these systems [1]. Database stores not only the permanent data, such as privacy data, person's credit card numbers, but also some important control data, such as some critical task states. So database becomes the hacker's main target and the main protection object in information systems [2-3]. But most information protection mechanisms are aimed to protect the perimeter of the network and to control the access to database [4]. In complicated real application environments, hackers can easily bypass the protection mechanisms of perimeter of the network and the DBMS to attack the underlying operating system (e.g. inside attacks). When hackers break into the underlying operation system, the datum stored in database would be accessed directly through the operating system's file management service [5]. In some circumstances, thefts maybe steal physical hard disks to get critical data. In front of these threats, defense of network perimeter and access control of DBMS are not enough to protect these sensitive datum stored in database [6-7].

Database encryption may occur in OS, DBMS and client. E. Goh etc. has proposed an architecture that used to encrypt P2P file systems [8]. They assume the network storage is untrusted. All the encrypting and encrypting operations are did in client. One of challenges on database encryption is that typical indexing techniques can't be used on encrypted data. Ernesto Damiani etc. [9] proposed an architecture in which the indexing information is stored in client. That requires change the client application. In some circumstances, it is not easy. Rakesh Agrawal etc. [10] proposed a new encryption scheme that could preserve the ordering for numeric data. Traditional indexing techniques can be used. But the encryption scheme only ensures numeric data comparison operations can be directly applied on encrypted data. Hakan etc. [12] proposed architecture to execute SQL over encrypted data. In Hakam's scheme, the query is partitioned two parts. One part is executed in and another is executed in client. This scheme is also a client mode. The indexing and changing client database service provider problems remains. Umesh Maheshwari etc. [13] propose to use a small amount of trusted storage in trusted platform to protect a scalable amount of untrusted storage. The scheme is based on trusted platform. In most systems, it is not applicable.

2 Adaptive weighted mean algorithm

System design should satisfy the following design criteria.

Criteria one: Adding encryption function into DBMS doesn't affect the DBMS structure.

Criteria two: Adding encryption function doesn't influence DBMS function, such as index.

Criteria three: Encryption is transparent to client.

Criteria four: Data manager can select one database to encrypt entirely or select one table of a database to encrypt.

Criteria five: System should provide multiple encryption algorithms.

Criteria six: Extending as little as possible SQL commands to providing encryption service and the encryption service don't change the standard SQL command specifications.

We add following SQL commands to provide encryption services:

(1) ENCRYPT ALL WITH ALGORITHM "crypto-algorithm" KEYLENGTYH klen BLOCKLENGTH blen

(2) ENCRYPT SELECTED WITH ALGORITHM "crypto-algorithm" KEYLENGTYH klen BLOCKLENGTH blen

(3) ENCRYPT TABLE tablename

(4) ENCRYPT KEYUPDATE

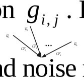
(I) encrypts the whole database. (2) sets the database partially encryption flag, then (3) encrypts a table. (4) Updates current user key. During encryption or decryption procedure, the system may break down for some reasons such as power fail. Prototype system provides a mechanism to recover the process when the encryption and decryption procedure is broken. The recovery process is automatic and needn't any people intervention.

The detected impulses will be removed by adaptive weighted mean algorithm. Let $f'_{i,j}$ be the value of the noise image at pixel location (*i,j*). For the corrupted pixel (i,j), the filtering window of size $(2L_f+1)\times(2L_f+1)$ is used. Starting with L_f =1, this filtering window iteratively extends outward by one pixel in its four sides until the number of noise-free pixels (denoted by $P_{i,j}$) within this window is not less than 1. Let $W'_{i,j}$ denote the values of noise-free pixels in the filtering window, i.e.,

$$W'_{i,j}=\{f'_{i+s,j+t}\,|\,b_{i+s,j+t}=0,b_{i,j}=1,\\ (s,t)\neq(0,0),-L_f\leq s,t\leq L_f\}. \tag{1}$$

The weighted mean value $g_{i,j}$ of the pixel values in $W'_{i,j}$ is defined as:

$$g_{i,j}=\frac{\sum\limits_{f_{i+s,j+t}\in W'_{i,j}} w_{i+s,j+t}\,f'_{i+s,j+t}}{\sum\limits_{f_{i+s,j+t}\in W'_{i,j}} w_{i+s,j+t}}, \tag{2}$$

where $w_{i+s,j+t}$ means the weight of $f'_{i+s,j+t}$. Let $m'_{i,j}$ be the median value of $W'_{i,j}$. Because the median value has the least probability to be the value of the corrupted pixels [1], $m'_{i,j}$ is utilized to determine $w_{i+s,j+t}$. It is easy to understand that the smaller the absolute difference between $f'_{i+s,j+t}$ and $m'_{i,j}$, the larger the weight $w_{i+s,j+t}$ should be to strengthen the influence of $f'_{i+s,j+t}$ on $g_{i,j}$. Based on extensive simulations which indicate that  is dependant on both above absolute difference and noise ratio, $w_{i+s,j+t}$ is chosen as:

$$w_{i+s,j+t}=\frac{R}{R+(1-R)\sqrt{\dfrac{\dfrac{|f'_{i+s,j+t}-m'_{i,j}|}{f'_{\max}-f'_{\min}}}{1-\dfrac{|f'_{i+s,j+t}-m'_{i,j}|}{f'_{\max}-f'_{\min}}}}}, \tag{3}$$

where $f'_{\max}$ and $f'_{\min}$ denote the maximum pixel value and the minimum one in the noise image, respectively.

The output of the DAWM filter is obtained by:

$$h_{i,j}=b_{i,j}\cdot g_{i,j}+(1-b_{i,j})\,f'_{i,j}. \tag{4}$$

3 Implementation

We implement a prototype based on PostgreSQL 7.4.2. In this section, we will not describe PostgreSQL but mainly point out the changes to PostgreSQL0.

In the prototype, we add a shared relation pg_dbcrypt to record database encryption state and encryption parameters, pg_dbcrypt = (oid, mode, encrypt, decrypt, klen, blen, dbkey), where oid is the database oid (unique identifier of database), mode describes the database encryption mode, mode=A means the database is encrypted entirely, mode=s means the database is encrypted partially, mode=N means the database isn't encrypted. Klen records the length of key. Blen records the length of encryption block. Dbkey records the encrypted database key.

We add a field, isecnrypt, in pg_class relation to describe if the relation is encrypted. If isencrypt is true, the relation is encrypted. Otherwise it is not encrypted. When a user login, a postgres backend is started. Each postgres backend has a shared memory (PGPROC) to record some critical information about the database. We add several members into PGPROC structure to record the encryption parameters. When a postgres backend is started, the encryption parameters of database are head from pg_dbcrypt and filled into PGPROC. The database key is decrypted first then filled into PRPROC.

In our prototype, the database key is generated according to the DS password (DSP). DBMS generate the database encryption key (DBK) according to DSP, DBK = [HASH (DSP || RANDOM)]$_{\mathrm{klen}}$. DBK is encrypted and stoned in database, EDBK=E$_{\mathrm{DSPHV}}$(DBK), DSPHV=HASH (DSP) . The encryption algorithm is 3DES. Encryption key is same as decryption key, decryption key DBK=D$_{\mathrm{DSPHV}}$ (EDBK)= D$_{\mathrm{DSPHV}}$ (E$_{\mathrm{DSPHV}}$))=DBK.

In PostgreSQL, postmaster is the database service backend. Postmaster receives connecting request when authentication passes, postmaster generates a backend (postgres) for this connection. Postgres will process the user commands later. In precedent postmaster start, user need not input password. We add some code to prompt DS input start password and verify it. If the password is right, DSP is stoned in shared memory. During the initial of postgres, the encryption parameters is read from pg_dbcrypt and stored in PRPROC, the database key is decrypted and stored in PGPROC and check if the encryption or decryption procedure is crashed. If the last encryption or decryption is not integrity, it will redo the encryption or decryption procedure.

When postgres receives encryption database SQL command, call Encrypt DB function. Database encryption process constructs a failover file; the encryption operation is doing according to the failover file. Failover file head record the new secure parameters, the number of relations, and the number of relations that have been transformed. Failover file body records the array of relations and the encryption operations. Failover file head structure is {New Encryption Algorithm, New Decryption Algorithm, Key Length, Block Length, Command Type, New State, Redo, Completed Relation numbers, total relations, New Key}. Failover body is constructed by an array of relation node which structure is {relation, whether encrypt this relation (E), whether decrypt this relation (D)}. Database encryption according to the old secure parameters and the database encryption command construct failover file.

After constructing the failover file, database encryption will encrypt or decrypt according the records of failover file. The transform function first read failover file, find the "completed relation numbers"(CRIB, CRN+1 point to the next transform node. Transform function read next node then transform the node.

When database encrypt state is "encrypt partially", data manager can select a table to encrypt through executing "encrypt table table-name" SQL command. Table encryption procedure will first judge whether the table in command is a valid table. Valid means that the table is a table stored in disk, corresponds to a disk file, is not a view and the table is not encrypted. If the table is a valid table, encrypt the table with the database secure parameters.

In above describing, encrypting relations operation is did on files directly. It bypasses the DBMS storage manager. In standard query command executions, DBMS read or write data through storage manager. Storage manager provides two interfaces to read and write data form disk, which are md_read and md_write. Md_read and md_write functions have the same arguments, which are relation, block number and buffer pointer. Block number means operation on which block of the relation. Buffer pointer means memory address which data will read to or write from. In precedent md_read function, data block is read from relation file to buffer directly. In our prototype, data block is read from relation file, and then the block is decrypted with the secure parameters stored in PGPROC structure. After decryption operation, decrypted block is saved to buffer. In precedent md_ write function, block in buffer is written to disk file directly. In our prototype block in buffer is encrypted firstly with secure parameters in PGPROC structure, then write the encrypted block to the relation's disk file.

In postgreSQL, there are two other storage interfaces and blind write and md_extend. In our portotype, the changes of and blindwrite and and extend are same as and write. In our prototype, we get the secure parameters from PGPROC not from pg-dbcrypt relation because we cannot access table in storage manager level through table access interfaces which will cause dead lock. So we must extend PGRPOC structure to store secure parameters for storage manager getting secure parameters.

When data manager think the database key is not secure and want to change the database key, he/she execute "encrypt key update" SQL command to update the database key. We encrypt the key stored in pg_dbcrypt with new key as following:

$$DSPHV' = HASH(DSP)$$

$$pg\_dbcrypy.key' = E_{DSPHV'}\left(D_{DSPHV}\left(pg\_dbcrypy.key\right)\right)$$

where pg_dbcrypt.key is the encrypted database key with DSP.

4 Experimental results

In this section, we test encryption and decryption effects on database performance. We insert 10 topples, 50 topples, 100 topples, 500 topples, 1000 topples, 5000 topples, 10000 topples respectively and continually into a relation which is not encrypted and record the used time. Then clear the test table and insert same content into a relation which is encrypted and record the used time. Compare the two groups of times.

Rc5 is a fast symmetric block cipher suitable for hard or software implementations. Rc5 is word-oriented; it has a variable word size, a variable number of rounds, and a variable-length secret key. A novel feature of Rc5 is the heavy use of data-dependent rotations-the amount of rotation performed is dependent on the input data, and is not pre-determined. While no practical attack on RCS has been found, the studies provide some interesting theoretical attacks, generally based on the fact that the "rotation amounts" in Rc5 do not depend on all of the bits in a register.

Rc6 was designed to thwart such attacks, and indeed to thwart all known attacks. Rc6 is an evolutionary improvement of Rc5, designed to meet the requirements of the AES. New features of RC6 include the use of four working registers instead of two, and the inclusion of integer multiplication as an additional primitive operation. The use of multiplication greatly increases the diffusion achieved per round, allowing for greater security, fewer rounds, and increased throughput.

According to the characteristics and restriction of the database encryption technology, the author puts forward a new block cipher suitable for database encryption which is named R encryption algorithm. R algorithm expansion routine of RCS, but the encryption algorithm of RC5 modified from RCS, RC6. It inherited key is modified. R inherits the data-dependent rotations of RCS, and also inherits integer multiplication of RC6, but it has only two working registers. The merits of RCS, RC6 are integrated in R algorithm, and small block is kept, so it is fit for database encryption. The security of R algorithm is higher than RCS, DES; exhaustive search for encryption key can be resisted due to the changeability of the length of key. The R encryption speed is faster. For R-32/16/16 on a 200MHz Pentium, a preliminary C++ implementation is compiled with the Borland C++ compiler, the encryption speed is 4.9M bytes/sec. It can fulfill the need of database encryption.

The comparison of adaptive weighted mean algorithm and RC5 can be seen from figure 1. The result shows that in the same decoding time, the adaptive weighted mean algorithm achieves better performance than RC5 in encryption complication.

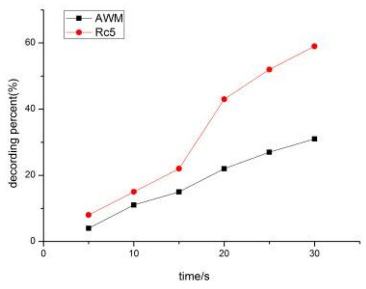


FIGURE 1 The comparison of adaptive weighted mean algorithm and RC5

The comparison of adaptive weighted mean algorithm and RC6 can be seen from figure 2. The result shows that in the same decoding time, the adaptive weighted mean algorithm achieves better performance than RC6 in encryption complication.

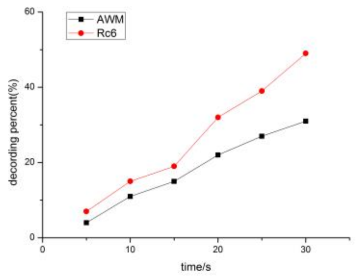


FIGURE 2 The comparison of adaptive weighted mean algorithm and RC6

The comparison of adaptive weighted mean algorithm and RCS can be seen from figure 2. The result shows that in the same decoding time, the adaptive weighted mean algorithm achieves better performance than RCS in encryption complication.

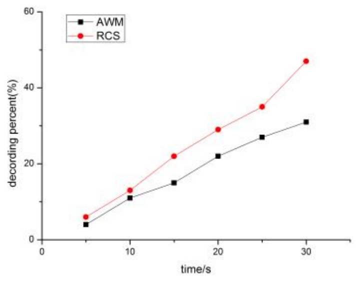


FIGURE 3 The comparison of adaptive weighted mean algorithm and RCS

5 Conclusions

Data protection plays an important role in computer security. The protection of database becomes very important in most information systems. Access control implemented in DBMS can resist most of threats on database except bypass threats. Data encryption can resist bypass threats well. Database encryption can be implanted into DBMS on different positions. Different schemes have advantages and disadvantages. We implement a prototype which implements encryption and decryption operations in a very low level of DBMS. This scheme has little effects on DBMS structure and performance. But in our prototype, the smallest encryption granularity is relation. The length of input block of cryptographic algorithm is the same as the length of output length. The encrypted blocks is independent each other. In future works, the prototype will be improved.

References

[1] Dorothy Elizabeth Robling Denning 1982 *Cryptography and Data Security* ADDISON-WESLEY Publishing Company 164-7
[32] Pitas I, Venetsanopou A 1990 *Nonlinear Digital Filters: Principles and Application* Norwell, MA: Kluwer 277-9
[33] Huang T S, Yang G J, Tang G Y 1979 Fast two-dimensional median filtering algorithm *IEEE Trans Acoust Speech Signal Process* **1** 8–13
[34] Umesh Maheshwari, Radek Vmgrdlek, Bill Shapiro 2000 How to Build a Trusted Database on Untrusted Storage *USENIX Symposium on QAerating Systems Design and Implementation* **32** 532–41
[35] Sun T, Neuvo Y 1994 Detail-preserving median based filters in image processing *Pattern Recognit Lett* **15** 341–7
[36] Wang Z, Zhang D 1999 Progressive switching median filter for the removal of impulse noise from highly corrupted images *IEEE Trans Circuits Syst-II: Analog Digital Signal Process* **46** 78–80
[37] Chen T, Ma K, Chen L 1999 Tri-state median filter for image denoising *IEEE Trans Image Process* **8** 1834–8
[38] Chen T, Wu H R 2001 Space variant median filters for the restoration of impulse noise corrupted images *IEEE Trans Circuits Syst-II:Analog Digital Signal Process* **48** 784–9
[39] Goh E, Shacham H, Modadugu N, Boneh D 2003 SiRiUS: Securing Remote Untrusted Storage *In Proceedings of the Internet Society (ISOC) Network and Distributed Systems Security (NDSS) Symposium* 133-45
[40] Eng H L, Ma K K 2001 Noise adaptive soft-switching median filter *IEEE Trans Image Process* **10** 242–51
[41] Rakesh Agrawal, Jeny Kiernan, Ramakrishnan Srikant, Yirong Xu 2004 Oider-Preserving Encryption for Numeric Data *SIGMOD Conference* 563-74
[42]Oprea and M Reiter 2005 Space efficient Block Storage Integrity *Proceedings of ISOC Network and Distributed System Security* 162-9.

Authors

Xiaoyan Wang, 1977.2.11, Shanghai

Current position, grades: Henan Zhengzhou, Master
University studies: computer science and its application
Scientific interest: Database technology
Publications: Numerous journal
Experience: For many years engaged in the work of computer teaching in University

Rui Sun, 1978.10.23, Hebi

Current position, grades: Henan Zhengzhou, Master
University studies: computer science and its application
Scientific interest: Database technology
Publications: Numerous journal
Experience: For many years engaged in the work of computer teaching in University