29. Daniel R. Ries and Michael R. Stonebraker, Locking granularity revisited. *ACM Transactions on Database Systems* **4** (2), 210–227 (1979).
30. C. S. Roberts, Partial match retrieval via the method of superimposed codes. *Proceedings of the IEEE* **67** (12), 1624–1642 (1979).
31. R. Sacks-Davis and K. Ramamohanarao, A two level superimposed coding scheme for partial match retrieval. *Information Systems* **8** (4), 273–280 (1983).
32. M. Schkolnick and P. Tiberio, Estimating the cost of updates in a relational database. *ACM Transactions on Database Systems* **10** (2), 163–179 (1985).
33. Kenneth F. Siler, A stochastic evaluation model for database organizations in data retrieval systems. *Communications of the ACM* **19** (2), 84–95 (1976).
34. V. Siskind and J. Rosenhead, Seek times for disc file processing: some results from probability theory. *The Computer Journal* **19** (4), 301–305 (1976).
35. Brad T. VanderZanden, Howard M. Taylor and Dina Bitton, Estimating block accesses when attributes are correlated. *Proceedings of the Twelfth International Conference on Very Large Databases, Kyoto, Japan, August 25–28, 1986*, pp. 119–127.
36. Brad T. VanderZanden, Howard M. Taylor and Dina Bitton, A general framework for computing block accesses. *Information Systems* **12** (2), 177–190 (1987).
37. S. J. Waters, File design fallacies. *The Computer Journal* **15** (1), 1–4 (1972).
38. S. J. Waters, Estimating magnetic disc seeks. *The Computer Journal* **18** (1), 12–17 (1975).
39. S. J. Waters, Hit ratios. *The Computer Journal* **19** (1), 21–24 (1976).
40. Kyu-Young Whang, Gio Wiederhold and Daniel Sagalowicz, Estimating block accesses in database organizations: a closed noniterative formula. *Communications of the ACM* **26** (11), 940–944 (1983).
41. S. B. Yao, Tree structures construction using key densities. *ACM 75: Proceedings of the Annual Conference of the Association for Computing Machinery, 1975*, pp. 337–340.
42. S. B. Yao, Approximating block accesses in database organizations. *Communications of the ACM* **20** (4), 260–261 (1977).
43. P. C. Yue and C. K. Wong, Storage cost considerations in secondary index selection. *International Journal of Computer and Information Sciences* **4** (4), 307–327 (1975).
44. John Zahorjan, Barbara J. Bell and Kenneth C. Sevcik, Estimating block transfers when record access probabilities are nonuniform. *Information Processing Letters* **16** (5), 249–252 (1983).

A Record-Oriented Cryptosystem for Database Sharing

The encryption/decryption scheme proposed in[8] is generalised in this paper. The new cryptosystem presented is record-oriented, i.e. each record is encrypted integratedly with different keys and each field is decrypted individually by separate keys, and has significant advantages over many conventional methods. Compared to the encryption system proposed by Davida, Wells, and Kam,[8] this new system has the advantages the previous one has and improves two drawbacks of theirs which will be stated below. Thus not only the security is increased but also the storage needed is reduced in the new cryptosystem.

Received August 1989, revised July 1990

1. Introduction

Generally, there are three methods for implementing data security: (1) physical security; (2) operating system security; and (3) data encryption.[9] Conventionally, people use physical security and operating system security to implement database security.[6,12] But it is not satisfactory for the following two reasons: (1) raw data exist in readable form inside a database; (2) it does not guarantee what we get from a database is the original data, which means the original data may have been modified by an intruder.

A database management system (DBMS) is the main line of defence for constraining users in traditional solution. By the use of current software techniques, it is not sure that illegal actions, such as Trojan horse programs,[4] that modify data can be avoided. Even hardware write locks and secure-periods processing are applied only to prevent unauthorised data modifications. There is no escape from unauthorised reading problems; however, read problems are more serious than write problem because read privileges are more frequently used than write privileges.[7] In the past decade, we have depended too much on DBMSs. Even if users are kept completely in legitimate channels for accessing data, the DBMS can lose raw data by omission, or the storage media may be physically stolen. Since raw data exist in readable form, the intruder can penetrate the DBMS and get or modify the data.

Although the encryption method cannot make a database perfectly secure, it indeed solves some problems which cannot be solved by conventional methods. Encryption of data can solve the previous two problems in the following ways: (1) if data are not stored in readable form, a cryptoanalyst cannot know the message without proper decryption key(s); (2) if a DBMS does not know the value of data, the Trojan horse code does not work; (3) data authentication problems are solved since attackers cannot change the ciphertext without knowing the encryption keys.[2,5,11,15]

Davida, Wells and Kam[8] have first proposed an elegant cryptosystem named DWKC for database security. DWKC was based on the Chinese remainder theorem[14] and was a new milestone of database security implementation. However, this scheme has two drawbacks which will be described in the next section. This article proposes a generalised cryptosystem to DWKC and solves the two mentioned problems. The presented cryptosystem satisfies all requirements as DWKC does and fits many desired properties listed in Ref. 8. Our new cryptosystem appears in Section 3. Thereafter we analyse the cryptocomplexity in Section 4.[13] Finally, discussions and conclusions are stated in the last section.

2. DWKC for Database Encryption

In 1981, a cryptosystem named DWKC for database encryption was first proposed by Davida, Wells and Kam.[8] DWKC was based upon the Chinese remainder theorem.[14] Let X be the ciphertext of an encrypted record, q_i be the decryption key for field i and there be n fields in each record. The encryption procedure is described mathematically by Formula (2.1).

$$X = \sum_{i=1}^{n} e_i m_i \bmod Q \qquad (2.1)$$

where $Q = \prod_{i=1}^{n} q_i$, m_i is the value of field i, $e_i = (Q/q_i)b_i$ is the encryption key for field i, and b_i is the multiplicative inverse of Q/q_i with moduli q_i. The decryption can be done by the following formula:

$$m_i = X \bmod q_i \qquad (2.2)$$

that is, by conducting a modulo with q_i to the ciphertext X, each m_i is revealed.

However, this scheme suffers drawbacks as stated below. Let m_i and m_i' be the values of the ith field of two different records R and R'. From Equation (2.2), it is easy to see that $X - m_i$ is a multiple of the decryption key q_i, i.e. $X - m_i = c_1 q_i$ for some constant c_1. Besides, if a user knows the other plaintext value m_i' and the corresponding ciphertext X', then he can determine the read key q_i by solving the following simultaneous equations:

$$X - m_i = c_1 q_i$$

and

$$X' - m_i' = c_2 q_i$$

where c_2 is another constant. Thus q_i can be determined with high probabilities by computing the GCD$(X - m_i, X' - m_i')$. To prevent the above known-plaintext attack, each field is then concatenated with a random redundancy value x_i (each 32 bits or longer; larger redundancy leads to more security) before enciphering. Henceforth, the encryption procedure is replaced by Equation (2.3).

$$X = \sum_{i=1}^{n} e_i (x_i \| m_i) \bmod Q \qquad (2.3)$$

where $\|$ indicates concatenation. On the other hand, the decryption is modified as:

$$x_i \| m_i = X \bmod q_i \qquad (2.4)$$

By discarding the random bits x_i, one can get the ith field value m_i. Though this new encryption (decryption) procedure increases the difficulty of a known-plaintext attack, it has a large storage overhead. In the next section, we will propose a new cryptosystem to solve the above problems.

3. A Proposed New Encryption System

A new encryption scheme for database security is now proposed. The proposed cryptosystem satisfies all of the required constraints for an encryption mechanism.[8] To illustrate the scheme, we assume that there are n fields in each record of a database. Let $m_1, m_2, \ldots, m_n$ be the n raw data fields of a record. Let the decryption keys be (k, q_i) for the ith field. Essentially, the encryption process is to convert the field values of a record into a ciphertext form, say X, and later we can recover it to the original raw field values by using the decryption key. This encryption is done by the following formula

$$X = \left(\sum_{i=1}^{n} p_i N_i \right) \bmod Q \qquad (3.1)$$

where $N_i = \lfloor m_i q_i / k \rfloor$ is a transformed version of m_i, p_i is the combination coefficient of N_i and Q is a constant as described in Theorem 1 below.

Conceptually, for encryption, first each field value m_i is crushed to be N_i, then multiplied by a mixing coefficient p_i. Taking the summation and finally a moduli, the ciphertext X of a record is obtained. The diagram of encryption scheme is shown in Fig. 1.

On the other hand, each field value m_i can be decrypted by the formula

$$m_i = \lfloor X/q_i \rfloor \bmod k \qquad (3.2)$$

where $\lceil \rceil$ means a ceiling operator and $\lfloor \rfloor$ denotes a floor operator, while the decryption process is illustrated in Fig. 2.

We can now show that Formulas (3.1) and (3.2) are correct.

Theorem 3.1

Let $q_1, q_2, \ldots, q_n$ be pairwise relatively prime numbers. Let $m_1, m_2, \ldots, m_n$ and k be integers satisfying

$$\max\{m_i\}_{1 \leqslant i \leqslant n} < k < \min\{q_i\}_{1 \leqslant i \leqslant n}$$

Then

$$X = \sum_{i=1}^{n} Q_i b_i N_i \bmod Q$$

is the smallest constant such that $\lfloor X/q_i \rfloor \bmod k = m_i$, where

$$Q_i = k \prod_{j \neq i}^{n} q_j, Q_i b_i \equiv k \,(\bmod\, k q_i), N_i = \lfloor m_i q_i / k \rfloor$$

$$\text{for} \quad i = 1, 2, \ldots, n \quad \text{and} \quad Q = k \prod_{i=1}^{n} q_i$$

Proof

The proof is similar to that of Lemma 2.1 in Ref. 3.

Let k, q_i, m_i, N_i, Q_i, b_i, and X be symbols as stated in Theorem 3.1. Obviously, we have the following result.

Theorem 3.2

Let $m_1, m_2, \ldots, m_n$ be n field occurrences of a record. Let $K_i = (k, q_i)$ be the decryption key for the ith field. Let N_i be the crushing function for m_i; $p_i = Q_i b_i$, the combination coefficients. Then both Formula (3.1) and Formula (3.2) are the encryption and decryption algorithms, respectively.

Example

Let $m_1 = 3$, $m_2 = 2$, and $m_3 = 5$ be the three fields of a record R on some database. Let $K_1 = (11, 13)$, $K_2 = (11, 14)$, $K_3 = (11, 15)$ be the corresponding decryption keys of the three fields and $Q = 30030$. Then $N_1 = 4$, $N_2 = 3$, $N_3 = 7$ and $p_1 = 16170$, $p_2 = 27885$, $p_3 = 16016$. Therefore, we have $X = (16170 \times 4 + 27885 \times 3 + 16016 \times 7) \bmod 30030 = 20207$.

On the contrary, the three fields can be revealed correctly by using Equation (3.2.).

Clearly, if each record is encrypted by applying different k values, the problem stated previously as in Ref. 8 can be easily solved. This will be shown in the next section. The redundant bits appended in each field can be eliminated. Thus not only the security is increased but also the storage needed is reduced.

4. The Crypto-complexity

One criterion for implementing the database cryptosystem is crypto-complexity: the work factor or the number of operations required to break the cryptosystem by a cryptoanalyst. Now, let us consider the crypto-complexity. Our cryptosystem keeps Q, k, p_i and q_i as secret keys. Trying to get Q, k and p_i directly from the equation

$$X = \left(\sum_{i=1}^{n} p_i N_i \right) \bmod Q$$

appears to have very few possibilities indeed. A possible approach is first to find out all q_i's from the decrption algorithm with known plaintext–ciphertext pairs as stated in DWKC. Then it may be easier to determine Q, k and p_i's. Now, let us see how possible a known-plaintext attack is if we use different encryption keys, k's, for different records. Let X and X' be the ciphertexts of two different records R and R', respectively. If m_i and m_i' are fields in R and R', respectively, and both are known by a cryptanalyst, then from Equation (3.2) he has

$$\lfloor X/q_i \rfloor \bmod k = m_i \quad \text{and} \quad \lfloor X'/q_i \rfloor \bmod k' = m_i'$$

imply

$$\lfloor X/q_i \rfloor - m_i = c_1 k \quad \text{and} \quad \lfloor X'/q_i \rfloor - m_i' = c_2 k'$$

for some constants c_1 and c_2.

The above simultaneous equations have three unknown variables, q_i, k, and k'. Hence, they have infinite possible solutions for q_i. In general, if t corresponding fields of t records are known, there are $t+1$ unknown variables to be determined with t simultaneous equations. It will be much more difficult for a known-plaintext attack than that in Ref. 8. Besides, the random redundancies concatenated in each field are eliminated and thus our scheme significantly improves the storage efficiency.

5. Discussions and Conclusions

We have proposed a generalised cryptosystem which satisfies the required properties as listed in Ref. 8 for protecting a database. It has significant advantages over many conventional methods. Basically, each encrypted record is a ciphertext form over an integrated record; i.e., a record-oriented. Meanwhile, small changes made to the encrypted values will cause significant changes to the decrypted values, thus unauthorised modifications of data can be prevented. Also pattern matching attacks[13] cannot work for an intruder. Moreover, because it encrypts all fields together, the possibility of substituting attacks[13] is eliminated. A user can only read one field value according to the corresponding privilege key he has. It is not required that unnecessary parts of the data be decrypted to satisfy a read request.

Finally, let us discuss the problem of the management of keys (k). Essentially, there are three places to maintain these keys.

(1) We may append one field for k value in each corresponding record and encipher the entire record. The problem of this method is how we can encipher k so that any authorised users can reveal it freely.

(2) k's are managed by DBMS; that is, users keep q_i's privately and DBMS keeps k's privately; the decryption of any field is completed by DBMS and users cooperatively.

(3) k's are kept secretely by users; i.e. each user has a decryption key pair (k, q_i) to decrypt one field; however, the only drawback is that users may manage several keys.

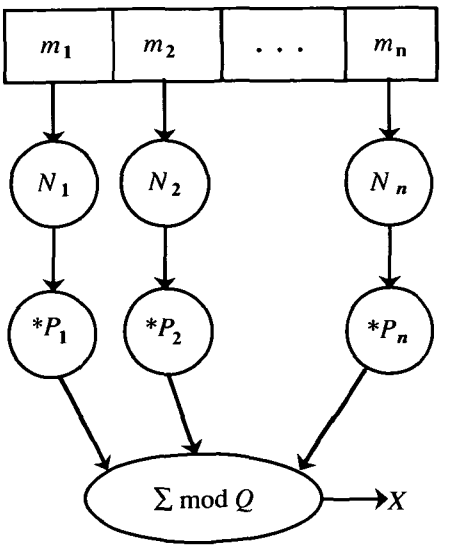


Figure 1. Encryption.

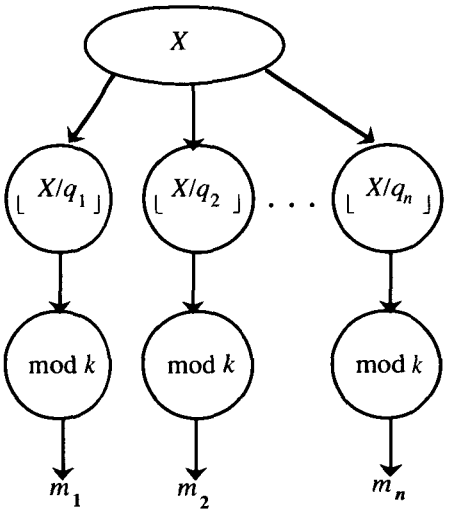


Figure 2. Decryption.

C. H. LIN[1], C. C. CHANG[2] and R. C. T. LEE[1]

[1]Institute of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30043, R.O.C.

[2]Institute of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 62107, R.O.C. (address for correspondence).

References

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass. (1974).

2. S. Berhovitz, J. Kowalchuk and B. Shanning, Implementing the public key scheme. *IEEE Communications Magazine* **17** (3), 2–3 (1979).
3. C. C. Chang and C. H. Lin, A reciprocal confluence tree unit and its applications. *BIT* **30**, 27–33 (1990).
4. D. E. Denning, *Cryptography and Data Security*. Addison-Wesley, Reading, Mass. (1982).
5. W. Diffie and M. Hellman, New directions in cryptography. *IEEE Transactions on Information Theory* **IT-22** (6), 644–654 (1976).
6. G. I. Davida, D. J. Linton, C. R. Szelag and D. L. Wells, Data base security. *IEEE Transactions on Software Engineering* **SE-4** (6) 531–533 (1978).
7. G. I. Davida and D. L. Wells, Microprocessors and data encryption. *Proceedings of IEEE COMPCON* Washington, D.C., 154–159 (1979).
8. G. I. Davida, D. L. Wells and J. B. Kam, A database encryption system with subkeys. *ACM Transactions on Database Systems* **6** (2) 312–328 (1981).
9. D. K. Hsiao, D. S. Kerr and S. E. Madnick, *Computer Security*. Academic Press Inc., ACM Monograph Series (1979).
10. J. Kam and J. D. Ullman, A model of statistical database and their security. *ACM Transactions on Database Systems* **2** (1), 1–10 (1979).
11. R. C. Merkle and M. E. Hellman, Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory* **IT-24** (5), 525–530 (1978).
12. N. Minskey, Intentional resolution of privacy protection in database system. *Communications of the ACM* **19** (3), 148–159 (1976).
13. K. H. Nam, Cryptographic models for computer communications. *Ph.D Dissertation*, University of Southwestern Louisiana (1985).
14. I. Niven and H. Zuckerman, *Introduction to the theory of numbers*. Wiley, New York (1966).
15. R. L. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **21** (2), 120–126 (1978).

A Short Note on Perfectly Balanced Binary Search Trees

We present a perfect balancing method for a binary search tree. During the updates the algorithm allows the structure to grow gracefully and maintains the optimal shape without degeneration. The algorithm uses swapping as the basic operation. Since the tree produced by the algorithm is optimal it can favourably be compared with that produced by other balancing algorithms. In worst case situation, the algorithm takes $O(n)$ time, n being the total number of nodes in the tree. This is an added significance when it is compared with the static optimal binary search trees.

Received September 1989, revised November 1990

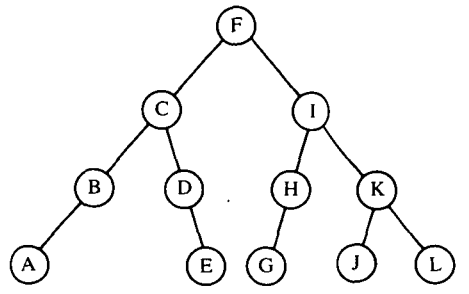
1. Introduction

Studies on the construction and maintenance of optimal binary search trees have received considerable importance in computer science literature. Knuth[9] and Wirth[13] describe algorithms producing static optimal binary trees. The complications involved in restoring the optimal shape of a binary search tree because of an update, lead to the formulations of 'permissive balance'. Such balancing strategies require only a little additional cost for the three reorganization process. Of the two balancing methods, global balancing algorithms[2,3,5,10,12] require either a traversal through the tree or a transformation of the input tree into some other structure, while the local balancing algorithms[1,8,11] keep the tree within the predefined balancing limit. A recent algorithm[7] produces a nearly optimal tree by displacing the data in an 'inorder' fashion until a vacant position is found in the lowest level of the tree. In the next section we present a strategy to create a perfectly balanced (optimal) binary search tree.

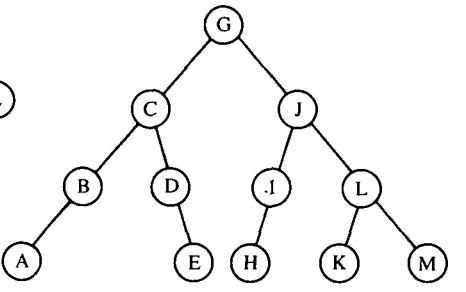
2. The insertion algorithm

Perfect balancing requires that at each node of the tree, the number of nodes in its left subtree nd right subtree differs at most by one. Hence perfectly balanced trees keep the optimal shape and they have the best worst case time in the tree operations. The standard tree data structure with an additional field for storing the number of nodes could be used to implement the tree.

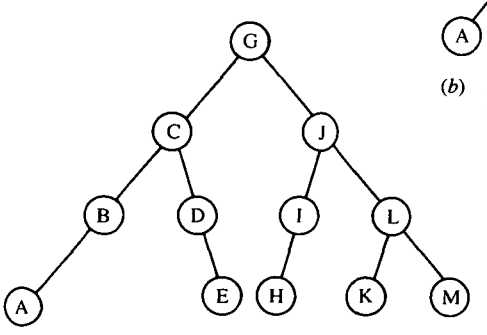
A subtree is defined as a heavy subtree if it has a brother subtree having less number of nodes. A perfectly balanced binary search tree



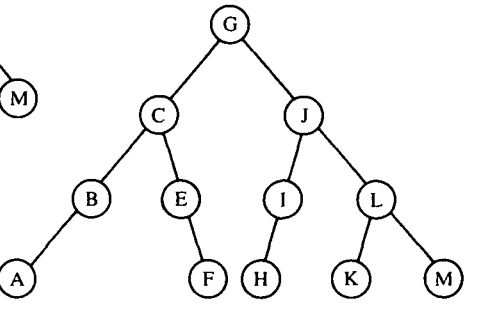
(*a*) Data to insert is say 'M'. Pivot node is 'C'. Number of nodes is 12.



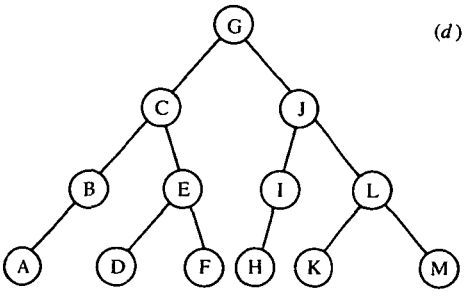
(*b*) After the first call (recursive). Data to insert is 'F'. Stepping down one level.



(c) After the second stage of the call. Data to insert is 'F'. Pivot is nil.



(*d*) After the third stage of call. Data to insert is 'D'.



(*e*) Final tree. A new node is inserted. Number of nodes is now 13.

Figure 1.