Search Key Substitution in the Encipherment of B-Trees

Thomas Hardjono and Jennifer Seberry

Department of Computer Science, University College
University of New South Wales, Australian Defence Force Academy
Canberra, ACT 2600, AUSTRALIA

Abstract

This paper suggests an improvement to the scheme by Bayer and Metzger for the encipherment of B-Trees. Search keys are "disguised" instead of encrypted, and together with the data pointers and tree pointers which remain encrypted, prevents the opponent or attacker from recreating the correct shape of the B-Tree. Combinatorial block designs are used as a method to substitute the search keys contained within the nodes of the B-Tree. The substitution provides advantages in terms of the number of decryptions necessary to traverse the B-Tree, while the use of block designs are advantageous in terms of the small amount of information that needs to be kept secret. The method is aimed at enhancing the use of encryption for the nodes of the B-Tree, and not as a replacement of the encryption algorithm. Although in this paper it is used in the context of B-Trees, the method may be applicable to other record storage organizations.

Keywords: Data encryption, Security, Database systems.

CR Categories: E.3, H.2.0, H.3

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

1 Introduction

In recent years there has been a considerable interest in the area of database security, and computer security in general. One of the security mechanism that can be used to secure the records stored inside a database system is based on cryptographic techniques, more specifically on the use of encryption to prevent illegal users from reading the contents of the database [1,2,3,4].

In many database encryption schemes that have been proposed the encryption module is not a part of the database management system. Instead, the module is located outside the database system, such as in a trusted front-end or a trusted "filter" system. In general, this placement of the encryption module results in the reduced flexibility on the part of the database management system. Records are encrypted and passed-on to the database management functions which places them according to their cryptogram values, not according to their original plaintext value. This results in the "scrambling" of the shape of the data structure that is used for the organization of the records. The inflexibility becomes obvious when range searches and partial-match queries are performed over encrypted records. The only search that can be performed without having to decrypt every record in the database is that of exact-matching between encrypted query elements and the cryptogram records. Using exact-matching as the only possible method for database searches, the operations of the database system becomes very limited.

One possible solution was suggested by Bayer

and Metzger in [5] in the context of B-Trees [6]. Instead of having the encryption module external to the database system and the data encryption performed at "high-level" in the sequence of the handling of the records, the encryption module should be placed inside of the database management system and data encryption performed at the "low level" close to the disk-write stage of the B-Tree node blocks and data blocks. The placement of records in data blocks and the shape of the B-Tree would then be the same as the case when encryption was not performed. The B-Tree that organizes the records has no knowledge of the encryption software or hardware. Before a node block or data block is written to the physical disk from the main memory, it is encrypted first. The process of decryption is applied to any node block or data block which is read from disk into main memory. This provides security in that the blocks used to store the records are completely encrypted and the opponent or attacker cannot distinguish one block from the next. Bayer and Metzger [5] suggest the use of hardware encryption module to perform this "on-the-fly" encryption and decryption. Note that in many commercial database packages access to the low-level database functions is not provided. The library of functions for the database customization usually limits the access to the database mechanisms that manages records. Thus, the scheme presented by Bayer and Metzger, and the improvements presented in this paper, are only applicable in the case where low level access to the database management system components is possible. Database systems which are developed "in-house" or commercial database systems that provide the source code access would be a suitable candidate for this scheme. In the following section we will briefly discuss the B-Tree encipherment scheme presented by Bayer and Metzger.

2 Background: Encipherment of B-Trees

In [5] Bayer and Metzger proposed a *page key* scheme for paged file structures (such as B-Trees), whereby each page in a file F has a page number P_i $(1 \leq i \leq m)$ which is used to derive an encrypting key for that page. More specifically, two encryption functions were proposed, namely T for text encryption and PK for page key encryption. Each page number P_i has a corresponding *page id* which is P_{id}.

For any page P_i having page id P_{id}, the contents of that page is encrypted using T with the page key K_{P_i} which is calculated as follows:

$$K_{P_i} = PK_{K_E}(P_{id})$$

where K_E is the *file key* or *tree key*. Then the page contents M_{P_i} of page P_i is encrypted using T with K_{P_i} as follows:

$$C_{P_i} = T_{K_{P_i}}(M_{P_i})$$

and decrypted as:

$$M_{P_i} = T^{-1}_{K_{P_i}}(C_{P_i})$$

Two kinds of encryption systems were proposed for B-Trees, namely *block* ciphers and *progressive* (*stream*) ciphers. In this paper we only consider the use of block ciphers, which is more applicable to fields of records or whole records.

3 Encryption of Node Blocks

In section 2 the units of storage used with respect to the encryption functions were pages within a given file. In the following sections we will adhere to the notation found in [7], in which the term *block* will be used to indicate the unit of secondary storage which holds a defined set of records of the database. The nodes of the B-Tree which do not hold any records are referred to as *node blocks* while those that do contain the actual records are referred to as *data blocks*. In the case where the records are stored in the node blocks the two terms are used interchangeably. Each node block will consist of the triplets [*search key, data pointer, tree pointer*]. The tree pointers of a node point to its children nodes, the search key is used to navigate through the nodes and the data pointer associated with a search key points to the block in secondary storage which contains the record with that search

key. To aid the discussion, the i-th triplet out of the n triplets in a node block b will be denoted as (k_i^b, a_i^b, p_i^b) or $(k, a, p)_i^b$.

From section 2 we see that one of the disadvantages of the scheme in [5] is that the contents of a given node of the B-Tree depends on some identifier (P_{id}) associated with that node. This dependency on the identifier ensures that each node has a unique encryption key, and that the encryption of two identical data items within two different nodes will result in two different cryptograms, making the attacks by an opponent harder. From the point of view of the insert and delete operations this dependency may result in too much overhead in the case of the B-Tree reconfiguration. Regardless of whether each node actually holds the data or holds a pointer to the data, when the contents of two child nodes are merged into one parent node each triplet (k_i^b, a_i^b, p_i^b) in the two child nodes must be decrypted, merged into the parent node and then re-encrypted. If the parent node occupies a new storage block, then a new identifier must be generated and a new encrypting key must be calculated. Out of the three elements of the triplet, the tree pointer p_i^b almost always changes when the triplet moves to a new node block. The search key k_i^b hardly ever changes, while the data pointer a_i^b only changes when the actual records are moved from one data block to another. Hence the scheme can be improved by removing the need to decrypt and re-encrypt static search keys within the triplets.

One result of keeping all the triplets (k_i^b, a_i^b, p_i^b) encrypted together is the need to decrypt more than one triplet within a given node block. In [5] Bayer and Metzger suggested a binary search-and-decrypt method to find the suitable tree pointer to follow. In the worst case this may take $log_2 n$ decryptions, where n is the number of triplets in a node block. An improvement to this scheme is to keep the search key k_i^b in node block b as plaintext, and disguise it in some way. Even if the search key is compromised, the fact that the tree pointer p_i^b and data pointer a_i^b are encrypted prevents the opponent from reconstructing the B-Tree. Hence a suitable format for the encryption of the triplet (k_i^b, a_i^b, p_i^b) is as follows:

$$f(k_i^b), E(b \parallel a_i^b \parallel p_i^b)$$

where E is the encryption function, f is the disguising function, b is the block number and the symbol "||" denotes concatenation. A node block with n triplets would have $n+1$ search keys k_i^b, n tree pointers p_i^b and n data pointers a_i^b [7]. The one tree pointer which does not have an accompanying tree pointer and data pointer should simply be disguised through the function f.

Having this configuration reduces the number of triplets that needs to be decrypted in a node block during a search. The function f can be a one way function, or even an encryption function. However, one of the motivations for not encrypting k_i^b is to reduce the number of necessary decryptions and reduce the amount of space required for storage of the cryptograms. The process of disguising the search keys and the encryption of the data and tree pointers should take place at the "low level", internal within the database management system after the shape of the B-Tree has been determined. In the following section we will describe a possible method of disguising the search key using combinatorial block designs.

4 Disguising Search Keys

In this section we discuss a possible method for disguising the search key k_i^b instead of encrypting it. It is important to realize that disguising the search key offers less security than encryption. However, the encryption of the tree pointer p_i^b and the data pointer a_i^b should prevent the opponent from reconstructing the B-Tree. Disguising the search key will make the effort to compromise the tree more difficult since the opponent cannot make an association between the disguised search key and the other two elements of the triplet. In this section the term *block* in *block design* should not be confused with *block* in *node blocks*. The first refers to constructs in combinatorial mathematics, while the later refers to database storage units. In discussing combinatorial block designs we will use the notations and definitions found in [8].

First, consider a difference set with parameters $\{v, k, \lambda\}$. Such a difference set is also recognized as a $\{v, b, r, k, \lambda\}$ balanced incomplete block design, with $b = v$ and $r = k$. Here v is the number of *objects* or *treatments* in a finite set S, and a block design which is based on the set S consists of a collection of k-sets from S, having the property that each object appears in r of the k-sets. The number r is also referred to as the *replication number*. In order to be useful for the encryption of node blocks, we must have that $v \gg R$, where R is the number of records in the database. This is to ensure that the process of disguising the search keys cannot be easily inverted by an opponent. Note that here the symbol k (as in k-sets) has no relation to that in the search key k_i^b.

A useful way to illustrate the possible use of block designs for disguising search keys is to consider the blocks as lines in the finite projective plane of order n with $v = n^2+n+1$, $k = n+1$ and $\lambda = 1$. In this instance the blocks $\{B_0, \ldots, B_{v-1}\}$ can be treated as lines $\{L_0, \ldots, L_{v-1}\}$. The incidence matrix for this block design consists of v rows of 1's and 0's, with a 1 in row x and column y of the incident matrix indicating that the point P_x $(x = 0, \ldots, k-1)$ lies on line L_y $(y = 0, \ldots, v-1)$. To disguise the search key of a given triplet, we can employ a two-step "scrambling" of the set of integers that make up the search keys in the B-Tree. The first step consists of the mapping from the search keys to the "integers" that represents points P_x $(x = 0, \ldots, k-1)$ on lines L_y $(y = 0, \ldots, v-1)$. This step can take the form of a natural mapping between the search keys and the treatments in the block designs. Of more interest is the second step, in which the points P_x $(x = 0, \ldots, k-1)$ are mapped to other combinatorial "structures", hiding the original block design. In order to be useful for disguising the triplets, the two step mapping must be invertible to allow the recovery of the original block design and hence the original integer search keys in the triplets.

One possibility for the second step is to map the points P_x $(x = 0, \ldots, k-1)$ on lines L_y $(y = 0, \ldots, v-1)$ to points on *ovals* O_y $(y = 0, \ldots, v-1)$. In this case an *oval* is defined to be a set of k points no three of which are collinear [9]. In practical terms, an instance of the mapping of lines to ovals can be achieved by simply multiplying the set of "integers" that represent the points P_x $(x = 0, \ldots, k-1)$ on lines L_y $(y = 0, \ldots, v-1)$ by a suitable value t modulo v, resulting in the set of "integers" that represent the ovals O_y $(y = 0, \ldots, v-1)$. A very small example of such a multiplication modulo v, mapping lines to ovals, is given below. Here we have a $(13, 4, 1)$ block design. With $t = 7$ the points $\{P_0, \ldots, P_3\}$ on lines $\{L_0, \ldots, L_{12}\}$ are mapped to ovals $\{O_0, \ldots, O_{12}\}$:

0	1	3	9	0	7	8	11
1	2	4	10	7	1	2	5
2	3	5	11	1	8	9	12
3	4	6	12	8	2	3	6
4	5	7	0	2	9	10	0
5	6	8	1	9	3	4	7
6	7	9	2	3	10	11	1
7	8	10	3	10	4	5	8
8	9	11	4	4	11	12	2
9	10	12	5	11	5	6	9
10	11	0	6	5	12	0	3
11	12	1	7	12	6	7	10
12	0	2	8	6	0	1	4

In this example there are 13 lines whereby 4 points occur on every line. The rows of the left-hand block design consists of the points occurring on the lines, whereas the right-hand block design shows the same points on ovals.

4.1 Substitution using ovals

One way of using block designs to disguise search keys is simply to renumber the search keys based on the result of mapping lines to ovals. The first of the two-step "scrambling" of the search keys can be ignored, and the actual search keys are used in the second step of the scrambling.

The substitution of the search keys is as follows. First, the treatments that make up points $\{P_0, \ldots, P_{k-1}\}$ on lines $\{L_0, \ldots, L_{v-1}\}$ are used as the actual search keys, and they are "disguised" or replaced using the treatments that make up the same points on oval $\{O_0, \ldots, O_{v-1}\}$. In effect, the search key k_i^b is multiplied by t modulo v, resulting in a new integer used as the substitute for the search key. In doing so, the apparent shape of the

B-Tree as viewed by the opponent will not follow the expected shape of a normal B-Tree. Having access only to the B-Tree representation on a sequential set of disk blocks, the opponent will face difficulty in determining the most likely children node blocks of a given parent block. An example of a small B-Tree before and after the search key substitution based on the above set of points is given in Figure 1.

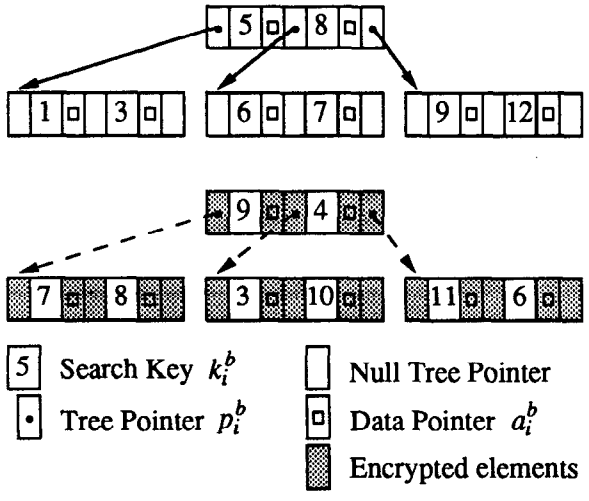

Figure 1: Example of the search key substitution using treatments on ovals

The substitution of a given search key is performed starting with line L_0. The k points on the line are compared with the search key. If none of the points on the line matches the search key, the next line L_1 is generated. The comparison is repeated again between the search key with the k points on the line. This process of comparison is repeated until a successful match is achieved, hence the requirement that $v \gg R$, where R is the number of records in the database. If one of the points on a line matches the search key, then an oval is generated corresponding to that line. The search key k_i^b, which matches the point on the line, is substituted by the equivalent point on the oval. In this instance, a "match" between the search key and a point means that their integer representations are equal. To prevent confusion, we will denote the "new" search key as $\overline{k_i^b}$. Thus, with respect to the previous block design and oval, the search key 1 is substituted by 7, 2 by 1, 3 by 8, 4 by 2 and so on.

In this scheme, the only information that has to be kept secret are the parameters $\{v, k, \lambda\}$ of the block design, the first line L_0 and the mapping from the lines to ovals. This presents a considerable advantage in terms of space used to hold cryptographic information. Conversion tables to maintain the correspondence between the actual and the disguised search keys are not required. In fact, the parameters of the block design and first block P_0 can be stored in "tamper-proof" devices, such as smartcards, which are carried by the legal users of the database system.

Note that we do not place triplets in node blocks based on the value of the disguised search key. The position of the triplets throughout the tree is the same as when the search keys are not disguised. This implies that the substitution of the search key k_i^b by $\overline{k_i^b}$ is done after the correct tree pointer p_i^b and data pointer a_i^b have been obtained and the triplet is ready to be written onto the physical disk. This ensures that range searches that require access to a whole subtree within the B-Tree can still be performed.

4.2 Substitution using exponentiation modulus

Although the encryption of the search keys provides the best security, it is disadvantageous in terms of the resulting cryptograms that have to be substituted for the search keys. Depending on the space available and other computer resources, the encryption of the search keys may be suitable. However, this will result in triplets that consume large storage spaces on the node blocks. Fewer triplets can be fitted onto a given node block, and the depth of the B-Tree would then increase substantially.

An alternative to the encryption of the search keys is to use the treatments in a block design as the exponents of a given integer modulo the number of objects v in the design. Another way of looking at this is to consider the treatments as exponents of a primitive element $g \in Z_N$, where N is prime. Here N should never be less than v. Sub-

stitution is performed as follows. First, a suitable treatment $t_{\alpha\beta}$ of point P_β on line L_α must be found such that $g^{t_{\alpha\beta}} \bmod N$ equals the search key k_i^b to be substituted. Once such a treatment is found, the corresponding treatment on oval O_α can also be found. The substitution is not done using the treatment on oval O_α. The treatment on the oval is instead used as the exponent of g modulo N, the resulting integer of which is used for the substitution. Here the value of g and N must be kept secret, in addition to the secret block design. A small example using the previous (13, 4, 1) block design with $g = 7$ and $N = 13$ is given below, where each exponentiation is reduced modulo N (Figure 2):

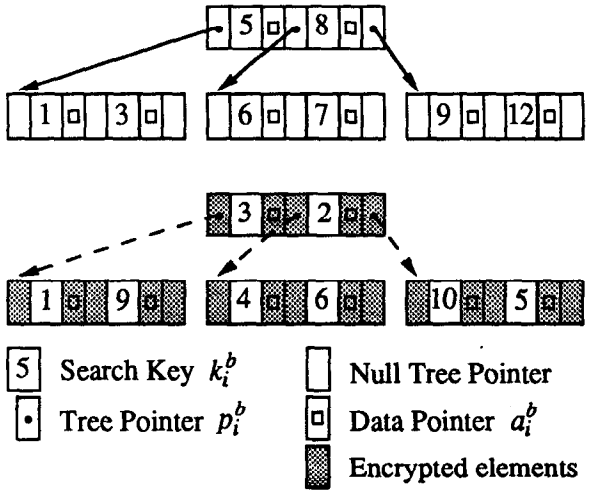

Figure 2: Example of the search key substitution using exponentiation modulus

7^0	7^1	7^3	7^9	7^0	7^7	7^8	7^{11}
7^1	7^2	7^4	7^{10}	7^7	7^1	7^2	7^5
7^2	7^3	7^5	7^{11}	7^1	7^8	7^9	7^{12}
7^3	7^4	7^6	7^{12}	7^8	7^2	7^3	7^6
7^4	7^5	7^7	7^0	7^2	7^9	7^{10}	7^0
7^5	7^6	7^8	7^1	7^9	7^3	7^4	7^7
7^6	7^7	7^9	7^2	7^3	7^{10}	7^{11}	7^1
7^7	7^8	7^{10}	7^3	7^{10}	7^4	7^5	7^8
7^8	7^9	7^{11}	7^4	7^4	7^{11}	7^{12}	7^2
7^9	7^{10}	7^{12}	7^5	7^{11}	7^5	7^6	7^9
7^{10}	7^{11}	7^0	7^6	7^5	7^{12}	7^0	7^3
7^{11}	7^{12}	7^1	7^7	7^{12}	7^6	7^7	7^{10}
7^{12}	7^0	7^2	7^8	7^6	7^0	7^1	7^4

This method of substitution can be made stronger by choosing N to be greater than v, and the only requirement is that g be a primitive element in Z_N. The larger the value of N, the more security is achieved. As mentioned previously, the only limit to N is the amount of space available on a node block. Similar to substitution using treatments on ovals, this substitution must be performed at a low-level, just before the actual disk-write stage of the node block.

4.3 Substitution using the sum of treatments in blocks

Instead of being associated with a given treatment in a block, the search keys can be associated with whole blocks in the design. First, a continuous subset of R blocks must be selected from the set of all blocks in a $\{v, k, \lambda\}$ block design. Similar to the block designs used in the previous sections, the blocks can be considered as a set of points $\{P_0, \ldots, P_{k-1}\}$ lying on lines $\{L_0, \ldots, L_{v-1}\}$. Then each search key can be assigned one line, either starting at L_0 or at L_w for some integer $w > 0$. If L_w is used as the starting line, then the number of lines v must be large enough to allow a continuous subset $\{L_w, \ldots, L_{w+R}\}$ $(w + R < v - 1)$ to be selected from the set of v lines. Thus, for R records in the database we must have $v \gg R$. The "integers" that form the treatments in a given line, which is associated to one search key, can be used as the input to another disguising function, the output of which is used as the substitute for the search key.

Given that there is a set of integers associated to a given search key, there are various possibilities in using the integers to diguise the search key. One possible method is to simply use the sum of the "integer" treatments of each line as the substitute for the search key associated with that line. More specifically, given that the starting line is L_w, where $w > 0$ and $w + R < v - 1$, and given that the search key k_i^b is associated with a given line L_x $(w \leq x \leq w + R)$ then k_i^b can be substituted

by $\overline{k_i^b}$, where

$$\overline{k_i^b} = \sum_{\alpha=w}^{x} \sum_{\beta=0}^{k-1} t_{\alpha\beta}$$

and $t_{\alpha\beta}$ is the "integer" treatment forming point P_β on line L_α. Here the summation is done without reducing modulo v. In simple terms, for a given line L_x ($w \leq x \leq w + R$), besides taking the sum of the integer treatments on the line L_x, all the other integer treatments starting at line L_w until line L_{x-1} is also added to the sum of treatments of line L_x. Using the (13, 4, 1) block design above, we have the following values for $\overline{k_i^b}$:

				$\overline{k_i^b}$
0	1	3	9	13
1	2	4	10	30
2	3	5	11	51
3	4	6	12	76
4	5	7	0	92
5	6	8	1	112
6	7	9	2	136
7	8	10	3	164
8	9	11	4	196
9	10	12	5	232
10	11	0	6	259
11	12	1	7	290
12	0	2	8	312

with Figure 3 showing the resulting B-Tree with the substituted search keys.

The summation method given in the example above does not have strong security when R is small. The subset of blocks ranging from w to $w + R$ is chosen to prevent the opponent from discovering the first block B_0 in the design. One advantage of the summation of the treatments is that is can be used for the substitution of search keys in high-level *Security Filters* [2,10] or front-ends retrofitted onto commercial "off-the-shelf" database management systems, which usually provide no access to low-level record routines except through a library of customization functions. For a given set of unique search keys k_i^b in an ascending order of size, the corresponding substitute search keys $\overline{k_i^b}$ derived through the summation of treatments is a set of integers maintaining

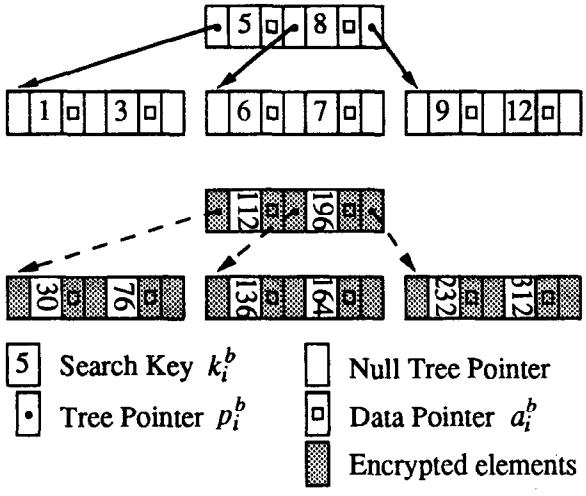

Figure 3: Example of the search key substitution using the sum of the treatments

that ascending order. Hence, whether substitution of the actual search keys is done at a "high level" before records are passed-on by the filter to the database management system, or at a "lower level" within the database management system components near to the disk-write stage, the shape of the B-Tree is still maintained. In the high level security filter, each record may have a plaintext search field which is included in the checksum calculation for that record. Instead of placing the actual plaintext search key in that field, a substitution can be performed before the security filter calculates the checksum. The database management system then creates its B-Tree based on the values in the plaintext substituted search key. Since the substitution using the sum of treatments preserves the ordering of the original search keys, the shape of the B-Tree would be the same as in the case when no substitution was performed. The summation of the treatments in the blocks or lines is by no means the only way to disguise search keys. With a search key being associated to a line, and thus to a set of "integer" treatments, many other substitution methods can be devised based on the treatments.

5 Encryption of Tree Pointers, Data Pointers and Data Blocks

There are a number of ways in which the triplet (k_i^b, a_i^b, p_i^b) can be encrypted. However, as mentioned earlier, increase in flexibility is obtained when the search key k_i^b is not encrypted The tree pointers p_i^b and data pointers a_i^b remain encrypted as in [5] together with the node block number.

Currently, two of the most commonly used cryptosystems are the Data Encrypted Standard (DES) [11] and the Rivest-Shamir-Adleman (RSA) [12] cryptosystem. The DES can be used to encrypt data segments or blocks of 64 bits, while the RSA cryptosystem can encrypt any data segments of length less than its modulus N. In this paper we consider the use of the RSA cryptosystem (or exponentiation modulus) for the encryption of the tree and data pointers. This choice is motivated by the fact that this cryptosystem is becoming increasingly available on hardware implementations, which may be suitable for low levels accesses to the disk blocks in the database system. In addition, the properties and weaknesses of the RSA cryptosystem are widely known and much research have gone into understanding the cryptosystem.

The RSA cryptosystem is known also as a "public-key" or "asymmetric" cryptosystem, and a number of possible attacks have been suggested and are discussed in [13]. These attacks are usually performed using the publicly known information, such as the modulus N and the public encryption (or decryption) key. Note, however, that the strength of the RSA cryptosystem is "increased" when it is used not in the usual public-oriented manner. More specifically, when the RSA cryptosystem is used to encrypt a message and none of the encryption parameters are made public, then the attacks by opponents are made considerably harder. The opponent may not even start to solve the factorization and discrete logarithm problems, since there is no public information available except for the cryptograms. A multilevel organization of encryption keys based on the RSA cryptosystem is presented in [14]. This multilevel scheme can be applied to the encryption of the tree and data pointers in the B-Tree, and also the data blocks. It may also allow each triplet in a node block to be assigned a security level, restricting access to data by users of lower security clearances.

The encryption algorithm used for the encryption of data blocks can be different and independent to that used for the tree and data pointers in the node blocks. Having a different encryption algorithm or a different set of encryption parameters may provide better security. The compromise of node blocks will only provide the location of data blocks, and such a compromise may not necessarily aid the opponent in breaking the data blocks. Note, however, that the compromise of the B-Tree node blocks can lead to the compromise of the search key substitution parameters. The illegal reconstruction of the whole B-Tree may inevitably show the substituted search keys in the B-Tree, and the opponent can proceed to deduce the possible values of the original search keys.

6 Remarks and Conclusion

In this paper we have suggested a method to substitute search keys in the context of the encipherment node blocks in a B-Trees. This method is based on the use of combinatorial block designs. The main aim of the search keys substitution is to allow faster traversal of the B-Tree by reducing the number of decryptions required in a given node blocks. Search keys are substituted instead of encrypted, and in determining the path to follow down the tree, comparisons of substituted search keys is faster than decryptions. Given that an opponent or attacker is trying to reconstruct the encrypted B-Tree, the substituted search keys will not provide the correct shape of the original B-Tree. This is also due to the fact that the tree and data pointers are encrypted. Another advantage is in terms of storage space. Substituted search keys require less space than an encrypted search key. Combinatorial block designs have been employed as a method for substituting search keys. From the security point of view the main advantage of the method lies in the small amount of informa-

tion that needs to be stored, namely the block design parameters and the first block of the design. The method is aimed at enhancing the encryption algorithm employed for the encipherment of the B-Trees, and not as a replacement the encryption algorithm. The encryption algorithm that is used to encrypt the tree and data pointers must be a secure one. The encryption algorithm and parameters used for the encryption of node blocks can be different to that used for data blocks. The method of disguising search keys proposed in this paper is by no means restricted to B-Trees, and it may be applicable to other record maintenance schemes used in database management systems.

Acknowledgments

The authors would like to thank Dr. Mirka Miller for the valuable discussions concerning B-Trees and for the comments and suggestions regarding the paper.

References

[1] D. E. Denning, "Field encryption and authentication," in *Advances in Cryptology: Proceedings of CRYPTO 83* (D. Chaum, ed.), pp. 231–247, Plenum Press, 1983.

[2] D. E. Denning, "Cryptographic checksums for multilevel database security," in *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pp. 52–61, IEEE Computer Society Press, 1984.

[3] N. R. Wagner, "Shared database access using composed encryption keys," in *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, pp. 104–110, IEEE Computer Society, Apr 1982.

[4] G. I. Davida, D. L. Wells, and J. B. Kam, "A database encryption system with subkeys," *ACM Transactions on Database System*, vol. 6, pp. 312–328, Jun 1981.

[5] R. Bayer and J. K. Metzger, "On the encipherment of search trees and random access files," *ACM Transactions on Database Systems*, vol. 1, pp. 37–52, Mar 1976.

[6] R. Bayer and E. McCreight, "Organization and maintenance of large ordered indexes," *Acta Informatica*, vol. 1, no. 3, pp. 173–189, 1972.

[7] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems.* Benjamin-Cummings, 1989.

[8] A. P. Street and D. J. Street, *Combinatorics of Experimental Design.* Oxford University Press, 1987.

[9] P. Dembowski, *Finite Geometries.* Berlin: Springer-Verlag, 1968.

[10] D. E. Denning, "Commutative filters for reducing inference threats in multilevel database systems," in *Proceedings of the 1985 Symposium on Security and Privacy*, pp. 134–146, IEEE Computer Society Press, Apr 1985.

[11] NBS, "Data Encryption Standard DES." FIPS PUB46, US National Bureau of Standards,Washington, DC, January 1977.

[12] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–128, Feb 1978.

[13] J. Seberry and J. Pieprzyk, *Cryptography: An Introduction to Computer Security.* Sydney: Prentice Hall, 1989.

[14] T. Hardjono and J. Seberry, "A multilevel encryption scheme for database security," in *Proceedings of the 12th Australian Computer Science Conference*, (Wollongong), pp. 209–218, Feb 1989.