The application of cryptography for data base security

by EHUD GUDES and HARVEY S. KOCH
The Ohio State University
Columbus, Ohio

and

FRED A. STAHL
Columbia University
New York, New York

ABSTRACT

The application of cryptographic transformations for the purpose of enhancing the security in data base systems is discussed. These transformations have been recognized in the past as a valuable protection mechanism but their relation to data base security has not been identified. The major reason is the lack of a suitable data base model for investigating the questions of security and cryptography. A multi-level model of a data base is presented in this paper. This model helps to understand the connection between the data base structure and the cryptographic transformations applied to the data base. It is shown that cryptographic transformations can be applied between the different levels of the data base. Several types of these transformations are identified and the possible ways of using and controlling them are also discussed. The multi-level model can provide a useful framework for further research in the area of cryptography and data base security.

INTRODUCTION

The technical problems associated with providing protection for information in a shared computer environment have received considerable attention in recent years. Petersen and Turn discussed a broad spectrum of these problems in their article on "System Implications of Information Privacy." Subsequently, there have been numerous endeavors attempting to amplify and find reasonable solutions to these problems. Concurrent with the developments in the area of protection has been the very important research in data base systems. This is due principally to the growing size and complexity of existing data bases. Research in data security and research in data base systems have only been combined recently. Most data security models use a unified approach in which all components of the computer system (i.e., objects) are viewed as being on the same level. A data base system is viewed as much more complex than the traditional file system and therefore its security problems are more complicated. In a data base, protection may be required from the file and record level down to the field level. User protection requirements of a data base are more complex than protection of a full file (or segment) and complex protection specifications based on boolean expressions may be needed. A unified approach is, therefore, not suitable for dealing with the security problems in data base systems. Our view is that the security problems of data base systems have to be dealt with separately from the security problems of operating systems. (A similar view is held by Minsky.) However, since a data base system uses the operating system services, the two problems cannot be completely disconnected.

Since we are interested in data base systems, our assumption is that the hardware and the operating system are correct and secure. This is not as strong an assumption as it may seem at first glance. Since we have removed the problems of files and data bases from the operating system domain, the operating system becomes smaller and easier to verify. Clearly, the complexity of the protection problem in the data base system increases.

In this paper we concentrate on the security problems of data base systems. We develop a model of a data base which allows the incorporation of several protection mechanisms. The goal is to define a structural model of a data base in which known protection mechanisms can be applied, and their dependency on the structure of the data base can be understood. In this paper we are interested in one protection mechanism called cryptographic transformations. The value of these transformations as a means of protection has been well established and some research was done on their application in file systems. This model will help us to understand where and how to apply these trans-
formations, and how these transformations can be used in conjunction with other protection mechanisms, in a data base system.

CRYPTOGRAPHY AND DATA BASES

Cryptographic transformations have been recognized long ago to be an effective protection mechanism in communication systems. In the past they have been used mainly to protect information which is transferred through communication lines. However, as Peterson and Turn pointed out, they can be used as an effective counter-measure against some of the threats to security which exist in computer systems. Among them are: wiretapping, between lines entry, browsing files, physical acquisition of removable files. Although several other articles mention their existence and suggest their application in computer systems (Skatrud, Van Tassel), not much (public) research has been done in this area. In particular, there is no research on the problems of how to apply cryptographic transformations to permanent sharable data bases as opposed to their application to information transferred through communication lines. We call the first form of cryptography data base cryptography as opposed to the more popular form of communication cryptography.

There are large differences between the constraints put on communication cryptography (see Shannon) and between those put on data base cryptography. Turn enumerated some of these differences. The major ones are: (a) The problem of selective retrieval—because files are usually organized so that selective retrieval of records can be achieved, it is very desirable that enciphering (deciphering) of record i will not depend on another record j. This constraint prevents the use of the popular Vernam Cipher using a pseudorandom number generator of a very large period. Such a generator would be usually used for enciphering large quantities of data (e.g., the whole file) and would have to include more than one record in the enciphering process. (b) The long ‘life’ of the data—data in data bases usually resides there for relatively long periods. Therefore, the very popular method of changing the cryptographic keys very often cannot be applied, since it will require a complete reprocessing of the data base or a large part of it. (c) The processing problem—data in files and data bases is stored for processing purposes. It would be very desirable if we could process the ciphered data in the same way we process the ‘clear’ data. The reasons for this are that the system is more secure if only ciphered data is processed and the overhead of enciphering/deciphering every time we access the data is saved. We would like therefore to design ‘processable’ ciphers. Examples of such ciphers will be given later. It can also be shown that in the case of a ‘processable’ cipher, applying the cryptographic transformation on the data item level only, is not secure enough. Given the constraints above, it is clear that the subject of data base cryptography is strongly connected to the subject of data base organization, representation and accessing. None of the current models of data base systems addresses this problem directly. Furthermore, current data base models are not suitable for answering questions related to data base cryptography.

Very few models of security in data bases mention the use of cryptographic transformations. The CODASYL model provides the ENCODING/DECODING clause on the data item level. However, application on the data item level only may not be secure enough. The connection to other protection mechanisms is not clear. Hoffman mentions the SCRAMBLE/UNSCRAMBLE procedures, but does not give any details of their use.

We are then faced with the following questions: (a) To which level should the cryptographic transformations belong? To the physical structure, to the logical structure, or to the mapping between them? (b) Should the Data Base Administrator (DBA) have complete control on the cryptographic transformations? Similarly, should the keys for these transformations be part of the system, e.g., in its Data Definition Language (DDL), or should only the appropriate user have some of the cryptographic keys? (c) Should cryptographic transformations preserve or destroy the structure of the data base and what are the advantages and disadvantages of each case? (d) What is the relation with other protection mechanisms? Should they complement each other and how should it be done? The main goal of this paper is to answer the questions above. However, in order to answer them, we need a framework—a data base model in which the security problems and their relation to the data base structure are clearly identified. Such a model will be developed. Before we develop this model, we want to review some of the basic concepts in data security.

BASIC CONCEPTS

Looking at the literature on data security, we find some degree of confusion about the basic concepts. We will use the terms security, protection and access control interchangeably and assign them the following meaning by McCauley: ‘The process of determining the authorized users of the data base and of determining which access may be permitted and which would be denied.’ Graham and Denning made a very important distinction between the protection specification and the protection mechanism in computer systems. The protection specifications are the translation of management privacy views into exact specifications. The
that any protection mechanism is the mechanism to execute correctly the protection specifications and to assure that any protection violation of these specifications will be detected.

In different systems there exist different protection mechanisms. Most of these mechanisms are composed of two parts: the protection procedure and the protection data. The protection data is not to be confused with the protection specification. The protection data is data which is internal to the protection mechanism, for example—passwords in the case of the password protection mechanism or tags in Friedman's model. The protection procedure is analogous to the program or procedure in programming systems and it is the coded form of the protection mechanism algorithm. An important example of such a two part protection mechanism are cryptographic transformations. In this case, the transformation algorithm is the protection procedure while the cryptographic keys are the protection data. The analogy to programming systems can be carried further as follows:

<table>
<thead>
<tr>
<th>Programming Systems</th>
<th>Data Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure</td>
<td>Protection Procedure</td>
</tr>
<tr>
<td>Temporary Variables</td>
<td>Protection Data</td>
</tr>
<tr>
<td>Input Data</td>
<td>Protection Specifications</td>
</tr>
<tr>
<td>Output Data</td>
<td>Protection Decision (i.e., Grant or Deny)</td>
</tr>
</tbody>
</table>

As an example for the use of the concepts above, let us look at the protection mechanism in OS/360 file system. The protection mechanism is a password protection mechanism where a file can be accessed if and only if the right password is given by the user during OPEN. The protection procedure then is part of OPEN. The protection data is the list of passwords. The protection specification is the distribution of passwords between users according to the privacy decisions: which user has access to which file. So passwords here has a double role: as the protection data and as the way to express the protection specifications.

In the next section, we will use the concepts defined here in the discussion of the data base model and its relation to security.

A MULTI-LEVEL STRUCTURED MODEL FOR A DATA BASE

The existence of several data base levels is well recognized in the data base “community.” Usually a distinction is made between the logical level (structure) and the physical level (structure). A good discussion of these levels and the mapping between them is given in Sibley & Taylor. In the CODASYL model we can distinguish three levels: the Sub-Schema and Schema levels which define the logical structure and the storage level which defines the physical structure. A four level model known as the entity-set model was suggested by Senko, et al. Another four level model, similar in concept to the one we suggest here, was suggested by Sibley. We could have developed most of our notions in the framework of one of these models, however we preferred to develop our own terminology and structure for two major reasons: First it allows us to stress the security point of view which we are interested in. Secondly, our model differs from other models by its recognition of the existence of more than one physical level in a data base. In most data base models only one physical level is recognized and this is usually the secondary storage structure. However in most conventional systems (even with virtual memory), data exists physically in more than one medium and this fact is very important from the security and cryptographic points of view. The main idea in this model is that a data base is composed of several logical or abstract levels which describe data which physically resides in one or more physical media and therefore have one or more physical structures.

The existence of these physical media is recognized even in the conceptual model of CODASYL. In CODASYL three media are identified:

1. The user-working area
2. The system buffers
3. The secondary storage

Therefore we should have logical schema which describe the data structure in each of these levels. Actually in the CODASYL model the sub-schema describes the data in Physical Level 1 while the schema describes the data in Physical Levels 2 and 3.

A FOUR LEVEL MODEL

In the model that we have developed there are four logical-abstract levels:

1. User-logical level
2. System-logical level
3. Access level
4. Storage level (also called structured storage level)

Each of these levels can be composed of several sub-levels. Corresponding to each logical level there is a physical level which is connected to a physical medium. We will now concentrate on the description of the logical levels and their relation to security.

The user-logical level corresponds to the way a user or a user group sees the data base. It is very similar to CODASYL's Sub-Schema with the exception that we do not have the constraint that the user-logical level must be a subset of the system logical level. On the contrary, it might be useful to have complex transformations between the two levels. Usually there are several user-logical level structures in a data base. The system-logical level describes the whole logical structure of the data base. It may correspond to CODASYL's Schema with the difference that indexes, directories, and access paths are not part of the system-logical level.
(they are part of the Schema in CODASYL.) The access level describes the directories, indexes and all possible access paths in the data base. The storage level is the result of applying the access level to a particular physical secondary storage device(s) and describes characteristics which are special for these devices. To each logical level corresponds one or more physical levels according to the number of physical media. An example is shown in Figure 1.

In this example the user logical level describes the data as it appears on the user site. The system logical level gives the interpretation to data which appears in memory. And the access and storage levels give the right interpretations to the data which reside on the secondary storage devices. It is conceivable that in the future the secondary storage hardware will do all the access calculations and therefore no access information has to be in memory or described in the system logical level. The main idea is the existence of more than one physical level corresponding to more than one physical media.

The relation of this model to security is discussed in Gudes. The model is shown to be general and to include other known data base security models as special cases. The main idea is the decentralization of protection mechanisms by the spreading of protection specifications and mechanisms through the different levels of the data base. In this paper we are interested with only one protection mechanism—cryptographic transformations. The model will be used as a framework for the application of cryptographic transformations in a data base. As is shown in the following sections, cryptographic transformations are a subset of the transformations between the physical levels of a data base, which is used for protection.

**FORMALISM**

In order to understand the relationship between the logical levels of a data base and their corresponding physical levels we need some notation. When one looks on a data base as it is represented on secondary storage one sees a sequence of 0's and 1's. These binary digits make sense only when one knows the right structure, coding and interpretation of the data. One starts with the simple division to data items and then starts to build the more complex blocks of the structure (repeating groups, records, files). The basic concepts, then, for describing a data base is the concept of the data item. A data item has the following properties:

- interpretation (attribute)—what the data item is and what it is used for.
- representation (coding)
- address

We denote data items as: \( d_i \)

A common case in data bases is that one data item contains one or more properties of other data items. We give two examples:

**Example 1**

<table>
<thead>
<tr>
<th>( d_i )</th>
<th>( d_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>interpretation:</td>
<td>Length of ( d_i )</td>
</tr>
<tr>
<td>value:</td>
<td>5</td>
</tr>
</tbody>
</table>

In this case \( d_i \) contains the property: length of \( d_j \).

**Example 2**

<table>
<thead>
<tr>
<th>( d_i )</th>
<th>( d_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>interpretation:</td>
<td>units distance</td>
</tr>
<tr>
<td>value:</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

If \( d_i = 0 \) then \( d_j \) is in miles
If \( d_i = 1 \) then \( d_j \) is in kilometers

Hence \( d_i \) contains the interpretation of \( d_j \). However we need also the interpretation of \( d_j \) in order to know what \( d_i \) is. The interpretation of \( d_i \) is probably documented in some manual which describes the system.

We see then that a set of data items may have several levels of interpretation. Some of them are in the data base itself and some of them are only implicit. Each physical level of the data is just a set of data items, where their interpretation is either in some of the data items themselves, or in the logical description of this level, or documented in some manual. More formally, a physical record \( j \) on level \( i \) is denoted as \( PR_{j,i} \) where this physical record has the address \( A_j \).

This physical record is an ordered tuple of data items

\[
PR_{j,i} = (d_{1,j}, d_{2,j}, \ldots, d_{n,j})
\]

The address of data item \( d_{k,j} \) is determined by its relative position and the length of all data items before it. These lengths can themselves be other data items or part of the logical description of this physical record. The order of data items within a physical record is then important for finding their addresses. (A physical record here is "continuous" by definition.) The physical data base on level \( i \) is the set of physical records on this level

\[
PB(i) = \{PR_{1,i}, PR_{2,i}, \ldots, PR_{n,i}\}
\]

In reality only part of a physical data base on some level will exist at any time (Except the fourth level—the storage level—in which the whole physical data base exists.)

The definition of physical records is very simple because we believe that the complex structure of the data base is connected to the interpretation we give to these data items. Most of this interpretation is in the logical levels of the data base. In order to describe these logical levels, we need to define more concepts.

---

**Figure 1**—The four levels of a data base
Two data items are called similar if they have the same interpretation. A field is an abstract concept representing a set of similar data items. A field has no value but usually has a unique name or identifier. We denote field \( j \) on level \( i \) as \( F^i_j \).

The notation \( d_j - F^k \) means \( d_j \) is a data item occurrence of the field \( F^k \). A logical record is a set of fields with a unique name or identifier. The order of fields in a logical record is immaterial because each field can be identified by its name. Logical records on level \( i \) are denoted as \( LR^i \).

What is the connection between logical records and physical records? Very simple. A physical record is an occurrence of a logical record and a data item is an occurrence of a field!

The logical data base on level \( i \) is a set of logical records plus their interpretation which is contained in the corresponding description language \( DL(i) \). The logical data base on level \( i \) is \( LDB(i) \).

\[
LDB(i) = \{LR^1, LR^2, \ldots, LR^m\} + DL(i) + \text{some implicit interpretation.}
\]

The connection between logical data base level, physical data base level, logical records and physical records is seen in Figure 2.

In order to better understand the concepts above we give two examples for describing structures which are common in data bases.

**Example 1: Repeating Group**

Suppose we have a repeating group which gives information on a person's children. In this example we disregard the level notation. The logical record is:

\[
LRC = (F^1, F^2, F^3)
\]

The following interpretation will appear as part of the logical description

- \( LRC \) — information on children
- \( F^1 \) — number of children
- \( F^2 \) — name
- \( F^3 \) — age

Physical records which are occurrences of this logical record are:

\[
PR = \{d_1 - F^1, d_2 - F^2, d_3 - F^3, d_4 - F^3, d_5 - F^3\}
\]

where \( d_1 - F^1 \), \( d_2 - F^2 \), \( d_3 - F^3 \), \( d_4 - F^3 \), and \( d_5 - F^3 \) and value \( (d_1) = V(d_1) = 2 \), \( V(d_2) = 'AMIT' \), \( V(d_3) = 25 \), \( V(d_4) = 'ROMIT' \), \( V(d_5) = 22 \) and

\[
PR = \{d_1, d_2, d_3, d_4, d_5\}
\]

where \( d_1 - F^1 \), \( d_2 - F^2 \), \( d_3 - F^3 \), \( d_4 - F^3 \), \( d_5 - F^3 \), and \( V(d_1) = 3 \), \( V(d_2) = 'SMITH' \), \( V(d_3) = 29 \) etc.

**Example 2: Sets**

Suppose we want to describe the structure:

\[LRX, LRY, \text{ and } LRZ\] are names for logical records.

\[
LRX = \{F^1, F^2, \ldots, F^m, F^m+1\}
\]

\[
LRY = \{F^1, F^2, \ldots, F^n, F^n+1\}
\]

\[
LRZ = \{F^1, F^2, \ldots, F^k, F^k+1\}
\]

with the following interpretation

- \( F^m+1 \) — address of the physical record occurrence \( LRZ \) which is the "son" (if there is more than one son, we need another field to specify the number of sons).
- \( F^k+1 \) — address of son \( LRZ \)
- \( F^k+1 \) — address of \( LRX \) parent
- \( F^k+1 \) — address of \( LRY \) parent

We do not show here physical record occurrences of the above logical records. It should be clear now that any structure whether inter or intra record can be represented in this way. The important fact is that the complex structure is part of the logical description while the physical records are no more than ordered tuples of data items.

The user logical level is denoted as \( U(\text{DB}) \) and \( U_i (\text{DB}) \). We denote logical records on Level 1 as: \( UR^i \). Therefore

\[
U(\text{DB}) = \{UR^1, UR^2, \ldots, UR^n\}
\]

We denote a logical record on Level 2 as \( LR^i \). The system logical level is denoted as:

\[
S(\text{DB}) = \{LR^1, LR^2, \ldots, LR^n\}
\]

Figure 2—The connection between logical data base level, physical data base level, logical records and physical records.
We do not put any constraints on the transformations between these two levels. For example in the CODASYL model we have the following constraint:

\[ \forall UR_i \supseteq LR_o \text{ s.t. } UR_i = LR_o (*) \]

We will allow more complex transformations. In the access level we introduce concept of access record. An access record is simply another name for a logical record on Level 3. We denote access record i as ARi. The access level is then,

\[ A(DB) = \{AR_1, AR_2, \ldots, AR_n\} \]

The transformation between the system logical level and the access level is very important and we would like to give some examples of it:

Example 3:

Suppose we have a file—\( LR = (F_1, F_2, \ldots, F_n) \)—and two indexes. The first index specifies for each data item occurrence of field \( F_1 \) the record(s) which contain this data item. The second index does the same for field \( F_2 \). We say that the logical record \( LR \) is translated into three access records:

\[ AR_1 = (F_1, F_2) \]
\[ AR_2 = (F_3, F_4, \ldots, F_n) \]
\[ AR_3 = (F_1, F_3, F_5, \ldots, F_n) \]

where we have the interpretation \( F_1 = F_2 = \text{address of record containing a specific data item occurrence of } F_1 \). (If there is more than one such record we need another field to specify the number of these records.)

\( F_1 = F_2, F_2 = \text{address of record containing a specific data item occurrence of } F_2 \).

\( F_1, F_2, F_3, F_4, \ldots, F_n = F_n \)

We therefore have three access paths to physical occurrences of \( LR \):

\[ AP_1 = \text{access path 1} = (AR_1, AR_2) \]
\[ AP_2 = (AR_2, AR_3) \]
\[ AP_3 = (AR_3) \text{ which represents a sequential search through the file.} \]

With the concepts of access records and access paths, we can present now any structure of directories or indexes.

Example 4—Inverted File

\[ LR = (F_1, F_2, \ldots, F_n) \] (This is also \( AR_2 \))
\[ AR_1 = (F_1, F_2, \ldots, F_n) \] (directory entry)

with the following interpretation:

\[ \begin{cases} 
F_{11} = \text{name of field } F_1 \\
F_{12} = \text{value of a data item occurrence on field } F_1 \\
F_{13} = \text{number of records containing the corresponding keyword.} \\
F_{14} = \text{addresses of these records}
\end{cases} \]

Notice that \( F_{13}, F_{14} \) represent together a repeating group. The possible access paths are

\[ AP_1 = (AR_1, AR_2) \]—using the directory
\[ AP_2 = (AR_2) \]—a sequential search

Example 5: Splitting

\[ LR = (F_1, F_2, \ldots, F_n) \]
\[ AR_1 = (F_1, F_2, \ldots, F_k, A_k) \]
\[ AR_2 = (F_k, F_{k+1}, \ldots, F_n) \]

\( A_k \) is the address of the occurrence of the second half of the same logical record. The only possible access path is:

\[ AP = (AR_1, AR_2) \]

This situation is shown below:

\[ \begin{array}{c}
A_1 \\
F_1, F_2, \ldots, F_k, A_k
\end{array} \]

\[ \begin{array}{c}
AR_1 \\
\uparrow AP \\
F_{k+1}, \ldots, F_n \\
AR_2
\end{array} \]

We will use this example in the next section.

We feel that the concepts of access records and access paths are very significant since they clarify much of the confusion about the question: to which level do indexes and directories belong? We denote the logical records in Level 4 (storage level) as \( LP_r \). The storage level is then:

\[ P(DB) = \{LP_1, LP_2, \ldots, LP_n\} \]

In many cases this level is equivalent to the access level, i.e. \( LP_r = AR_r \). However it may differ in two important aspects:

1. Control information used by the device such as "gaps" or keys may be added.
2. If cryptographic transformations are used on
this level, they may destroy the whole data item structure. In that case we can speak on another level—Level 5—called unstructured storage level.

Until now we discussed mainly the logical abstract levels. As was explained before, the physical records are just occurrences of the corresponding fields. One comment is appropriate at this point. It may be that in reality we will have a smaller number of physical media, and therefore of physical levels, than the number of logical levels. In that case, some of the logical levels are abstract and no corresponding physical level exists. The importance of having these abstract intermediate logical levels is for understanding the structure. For example, it is usually the case that the access level and the storage level correspond to one physical medium—the secondary storage. In this case, the physical records on Level 4 are occurrences of the logical records on Level 4. No physical records on Level 3 exist. The existence of the access records is important because otherwise the transformation $L_A \rightarrow L_P$ would be very difficult to understand. In the case that we have only one physical medium, we come to the traditional view of a data base of several logical levels and one physical level! So our model contains the traditional view as a special case.

CRYPTOGRAPHIC TRANSFORMATIONS

Now, with the notation defined, we can return to the subject of cryptographic transformations. These transformations can affect both the structure of the data and its coding or representation. The importance of having several physical levels for a data base should now be clear. We can look at the cryptographic transformations as transformations between two consecutive physical levels. In general, cryptographic transformations are a subset of the set of transformations between the physical levels of the data base, which is used for protection. Cryptographic transformations can be a powerful protection mechanism. They can be used in two ways:

1. As part of the standard access control. That is, serving as a protection mechanism for implementing the protection specifications as they are defined in each level. In this case the user must know about the existence of these transformations since he probably holds some of the cryptographic keys.

2. As a system tool for protecting against illegal access paths such as browsing or wire tapping. In this case the cryptographic transformations are completely controlled by the system, they are not connected to a specific protection specification and the user does not have to know about their existence.

Of course, the combination of these two methods can be used. We leave the subject of how to use cryptographic transformations to the next section and concentrate in this section on describing the possible cryptographic transformations between the different physical levels of the data base.

TRANSFORMATIONS BETWEEN PHYSICAL LEVELS

We start with the transformation between the user-logical and the system-logical levels. Clearly, the transformation between the corresponding physical levels is dependent on the transformation between the logical levels. In general, these transformations can be very complex. However, if we want to use system services to process and query on data items and we want the ability of sharing data with other users, the common data item structure has to be preserved. We identify three types of transformations between physical Levels 1 and 2:

Type 1: Data Item Substitution

$$d_1^f = f(d_2^j)$$

$$d_1^j = f^{-1}(d_1^f)$$

This is the common substitution transformation which does not destroy the data item structure. Its advantage is its simplicity and the fact that some processing, such as querying, can be done on data items even in their ciphered form. Its disadvantage is that it may not be secure enough. (Because the statistical properties of similar data items may be saved.)

Type 2: Data Item Expansion

$$d_1^f = f(d_1^j, d_1^j_1, \ldots, d_1^j_k)$$

A data item in the system logical is expanded to several data items and finer structure on the user logical level. As an example suppose we have the fields: NAME, AGE, SEX on the user logical level, while on the system logical level we have one field: PERSONAL DATA, whose occurrences are a scrambled form of occurrences of the three fields on the user logical level. This transformation can be made more secure than the former one but it has a major disadvantage which is: because the fine structure on the user logical is destroyed a user cannot ask queries, which require processing on the system logical level, about individual fields such as: NAME, AGE, SEX since these fields are not defined in the system logical level.

A third type of transformation is data item contraction

$$d_1^f = f(d_1^j, d_1^j_1, \ldots, d_1^j_k)$$

Clearly more complex transformations can be defined between Level 1 and 2. One comment is appropriate at this point. There may be some cryptographic transformations between the two levels that we are not
aware of. For example, if Level 1 and Level 2 are far apart and are connected by telephone lines, some enciphering device may be connected to these lines. We are not interested in this kind of enciphering which has nothing to do with the data base structure and as far as the representation of the data in the two levels is concerned does not have any effect. The same comment is true in the case of transformations between other physical levels of the data base.

**System Logical→→Access→→Storage**

We discuss the transformations between these three levels together since we consider the case of one physical medium to both Levels 3 and 4. We identify three types of transformations:

**Type 1: Substitution Oriented**

\[ d_i^1 = f(d_i^1), \quad d_i^2 = f^{-1}(d_i^2) \]

The importance of this transformation is that it allows us to define **processable ciphers**. Most of the processing of data items is done on the system logical level (in main memory). So our definition of processable ciphers must be connected to data in Level 2. The advantage of being able to process data in its ciphered form were discussed above. Informally, a processable cipher is a transformation from the system logical level to the storage level which preserves the data item structure and allows the type of operations which are done on the "clear" data to be done on the ciphered data items. Formally, if \( d_i^1 = f(d_i^1) \) and \( d_i^2 = f(d_i^2) \) and \( G \) is a defined operation on data items in Level 2, then \( f \) is processable if and only if:

\[ \exists G, G \rightarrow G(d_i^1, d_i^2) = f(G('d_i^1, d_i^2')) \]

Usually we also require that \( G = G^{-1} \). For example, if \( G \) is the compare operation, then we have

\[ d_i^1 = d_i^2 \leftrightarrow f(d_i^1) = f(d_i^2) \leftrightarrow d_i^2 = d_i^2 \]

Such a cipher \( f \) is called a **retrievable cipher**, since it allows the searching and retrieving of records in their ciphered form. (This is done by comparing the ciphered form of the query to the ciphered content of the record.)

**Type 2: Transposition Oriented**

We have explained before that the order of data items in a physical record is important in order to find their addresses. However, this order may change from one physical record to another. For example, suppose a logical record on Level 4 is: \( LP = (F_1, F_2, F_3) \) and following are occurrences of physical records:

\[
\begin{align*}
PR_1 &= (d_{11}, d_{21}, d_{31}) \\
PR_2 &= (d_{12}, d_{22}, d_{32}) \\
PR_3 &= (d_{13}, d_{23}, d_{33})
\end{align*}
\]

Where:

\[
\begin{align*}
d_{11} &\sim F_1, & d_{12} &\sim F_1, & d_{13} &\sim F_1 \\
d_{21} &\sim F_2, & d_{22} &\sim F_2, & d_{23} &\sim F_2 \\
d_{31} &\sim F_3, & d_{32} &\sim F_3, & d_{33} &\sim F_3
\end{align*}
\]

That is, the order of data items in a physical record changes from one record to another. Such a transformation is very effective against "browsing" since the starting address of data items is not known to the illegal "browser". Its disadvantages are that it increases the access time of finding a data item, and that another data item has to be added which contains information about the specific order in each record.

**Type 3: Access Oriented**

In this transformation we encipher only the data items which allow the transfer from one access record to another in a specific access path. For example, suppose we use the splitting example above. We have

\[ LR = (NAME, F_3, SALARY, F_4) \]

and

\[
\begin{align*}
AR_1 &= (NAME, F_3, A) \\
AR_2 &= (SALARY, F_4)
\end{align*}
\]

A— is the address of the second half of an occurrence of \( LR \). If we encipher the field \( A \), then matching the right salary to the right name without deciphering this field is very difficult. The reason is that we have used the notion of access path for enciphering. In this case we enciphered the only existing access path. This type of transformation can be very effective, since address fields do not have regular statistical properties. (Note also that fields NAME and SALARY do not have to be enciphered.)

**COMBINED TRANSFORMATIONS**

Shannon has shown the strength of combined cryptographic transformations. The simplest example of a combined transformation is the substitution—transposition transformation. In this case data items are first enciphered by substitution transformation and then transposed by the transposition transformation. It is very difficult now to get the statistical properties of data items in order to "break" the substitution because of the transposition.

Another effective combination is the access-substitution combination. Following are two examples of substitution access combination.

**Example 1: Hashing**

Suppose we have a hashing "file". We have one logical record

\[ LR = (F_1, F_2, \ldots, F_n) \]

and two access records

\[
\begin{align*}
AR_1 &= (F_1) \rightarrow F_1 \text{ is the hashing field} \\
AR_2 &= (F_2, F_3, \ldots, F_n)
\end{align*}
\]
We have two access paths. The legal access path—\((AR_1, AR_2)\) and the illegal one—\((AR_2)\) which is equivalent to a sequential search. Suppose we use the following transformation:

\[ k + d \rightarrow \text{hashing address} + K' \]

where \(d \sim F_i\), and \(K\) is the cryptographic key. The meaning of this transformation is that as a result of the hashing process we get an address and a key \(K'\). Key \(K'\) which is the "residue" of the hashing transformation is used for the substitution transformation of data item \(d\). The illegal path of \((AR_2)\) is now very difficult, since in order to guess what the substitution is, we need to go through the legal access path \((AR_1, AR_2)\) using the hashing algorithm and the cryptographic key \(K\).

Example 2: Inverted File

Suppose we have the "file" described above in Example 4. That is

\[
LR = (F_1, F_2, \ldots, F_n) \\
AR_1 = (F_n, F_n, F_{14}, F_{14}) \\
AR_2 = (F_n, F_2, \ldots, F_n)
\]

The two possible access paths are \((AR_1, AR_2)\) and \((AR_2)\). We add to access record \(AR_1\) a KEY field. That is

\[
AR_1 = (F_{13}, F_{13}, F_{13}, F_{13}, F_{13})
\]

where \(F_{13}\) is the key to encipher records with a specific keyword. All records which contain a data item occurrence of Field \(F_{13}\) (\(=\)keyword) are enciphered by specific occurrence of field \(F_{13}\). Therefore all records which have the same keyword are enciphered by the same substitution key. Again illegal path \((AR_2)\) is protected by the fact that for deciphering we need to know which records contain a particular keyword, but this information is contained in the directory only, which is part of the legal access path. (*) Many variations of the combined transformation technique are possible. One possible extension of this is to build the directory as a tree where the key for enciphering a node of the tree is contained in its parent node. The subject of the enciphering of search trees is discussed also in Bayer and Metzger.\(^\text{8}\)

Storage \(-\rightarrow\) Unstructured Storage

One can think of a general structure destroying transformation which can be used, for example, for backup or data migration purposes. Since usually another physical medium is involved (e.g., magnetic tape) we define another logical and corresponding physical level called: unstructured storage level. We do not give any special notation for this level. (It should be straightforward using the examples above.) An example for such transformation is to use a pseudo-random number generator to encipher a complete "file" or a large part of a data base. Such transformation can be made very secure. Its major disadvantage is that complete deciphering is necessary before any access is possible.

We summarize this section with the following schematic diagram of types of cryptographic transformations and their use in the multi-level data base model:

**Using Cryptographic Transformations**

As was pointed out above, there are two ways for implementing cryptographic transformations: (1) as part of the standard protection mechanism according to some protection specifications. These transformations are user oriented (since the protection specifications are user oriented) and the users should know about their existence, since they probably have some of the cryptographic keys, (2) as a system tool to protect against illegal access paths such as "browsing" or "wiretapping". In this case users may not be aware of the existence of these transformations and their keys are under system (or DBA) control.

The actual implementation of the two methods depends on the data base structure and on the way the protection specifications and mechanism are spread out through the different levels of the data base.
SYSTEM CONTROLLED TRANSFORMATIONS

The following transformations are examples for system controlled transformations (method 2 above):

1. The transformation between the unstructured storage and the structured storage levels are always system controlled.
2. Combined transformations such as substitution-transposition or substitution-access can be system controlled for protecting against the illegal access path or browsing (e.g., in case of "stealing" a physical secondary storage device). In this case the cryptographic keys for these transformations have to be internal to the data management or access control methods.
3. Substitution oriented transformations, including processable ciphers under system control, can be used to protect against wiretapping or against dumping of main memory. Again, the keys for these transformations are either internal to the processing primitives of the query language or are part of the DDL for Level 2, but they must be protected by standard access control methods.

We see then that system controlled transformations are easy to implement and quite simple to control. They are controlled by the DBA and he can change the cryptographic keys at any time. They are very effective against illegal access paths such as: "browsing" or "wiretapping" but they must be complemented by standard access control methods for protection of the cryptographic keys. This method, of using both system controlled cryptographic transformations and standard protection mechanisms (e.g., passwords) such that each mechanism protects the "weak" areas of the other, can be very effective provided that adequate administrative security exists so that the probability of compromising both mechanisms is low.

USER CONTROLLED TRANSFORMATIONS

The situation with user controlled transformations is more complex, and the main reason is the existence of sharing and overlapping protection specifications. Clearly, if every user has access to a unique part of the data base, then this part can be enciphered by a unique key and only the appropriate user has the right key. But this case is unreal since the main purpose of a data base is sharing data. McCauley\textsuperscript{\textit{12}} has suggested a way to partition a data base using security atoms. The security atoms are data base dependent and not protection specification dependent! Every user has access to a set of security atoms according to the protection specifications. A way to use cryptography in this model will be to encipher each security atom with a unique key and distribute the keys between users according to the protection specification. (Somewhat similar to the passwords mechanism.)

Another possible use of user controlled cryptography is based upon the fact that the only "clear" data which is sent to the user is the data that he should have legal access to. (The fact that he gets a lot of "garbage" may discourage him from issuing illegal queries.) The main advantage of user controlled transformations versus system controlled transformations is concerned with the problem of key placement. In the system controlled case, the cryptographic keys must reside at all times in the system and therefore must be protected by the standard protection mechanisms. In the user controlled case there is the possibility that only the user(s) will have some of the cryptographic keys to their authorized data. These keys do not reside in the system except at the actual time the user is accessing his data. In this case even the DBA cannot "decipher" the user's data since he does not have the right key. This is certainly in accordance with the "need to know" concept, i.e., that the DBA does not have to know the content of every data item in the data base. Such a system, where only the authorized user has the cryptographic key was implemented by IBM\textsuperscript{\textit{2}} using the LUCIFER system to protect a "secure field." There are two disadvantages to this method. The first is the possibility of penetration of illegal cryptographic keys. Since the right key is not stored in the system, some checking algorithm (e.g., check digit) must be used. The probability that an illegal key will not be detected by the checking algorithm is not zero and its results in case of updating can be disastrous! The second disadvantage is that it is very hard to control the "user controlled transformations" case. For example, close cooperation between users which share the same data is needed if the cryptographic keys are to be changed. Also, close cooperation is needed between users and the DBA in some cases of error recovery and data base reorganization.

We note that cryptographic transformations which are controlled by the system are easier to implement but since the keys must reside within the system, they may be less secure. User controlled transformations allow the possibility that only the user will have the right cryptographic key, and therefore provide more security, but they require data base partitioning and probably a large number of keys, and therefore present quite a serious control problem.

FURTHER RESEARCH

The model presented in this paper is used as a framework for further research in the area of cryptography and data base security. This research is reported in Gudes.\textsuperscript{\textit{8}} The main areas of this further research are: (1) a detailed investigation into the problem of how to design an authorization scheme which is based on user controlled cryptographic transformations. Sev-
eral types of protection specifications such as: compartmentalized, hierarchical or data dependent are discussed and several cryptographic schemes are suggested for their implementation, (2) research into the problem of evaluating the security of data base enciphering. A "security measure" for data base enciphering is suggested and is computed for several types of ciphers, (3) an experiment using a simple file system is being performed. The goal of this experiment is to gain insight on the amount of security provided by the different ciphers and the overhead associated with them. The results of these areas of the research will be reported in future papers.

SUMMARY

In this paper we have investigated the problem of applying cryptographic transformations for enhancing the security of data base systems. These transformations have been recognized in the past as a valuable protection mechanism but their relation to data base security has not been identified. The major reason was the lack of a suitable data base model with which the problems related to security and cryptography can be analyzed. Such a model is developed in this paper. This is a multi-level model of a data base which helps us to understand the connection between the data base structure and the cryptographic transformations applied to the data base. We have identified several general types of cryptographic transformations, and have shown the two major ways in which they can be used: as system controlled transformations, or as user controlled transformations. More research is being conducted on the problem of designing a secure system based on cryptography and on the cost-effectiveness of the application of cryptographic transformations in a data base system.

REFERENCES
