To appear in: Advances in Cryptology; Proceedings of Crypto 83,
Plenum Press.

FIELD ENCRYPTION AND AUTHENTICATION

Dorothy E. Denning[1]

Purdue University
West Lafayette, Indiana

Abstract

Database encryption and authentication at the field level is attractive because it allows projections to be performed and individual data elements decrypted or authenticated. But field based protection is not usually recommended for security reasons: using encryption to hide individual data elements is vulnerable to ciphertext searching; using cryptographic checksums to authenticate individual data elements is vulnerable to plaintext or ciphertext substitution. Solutions to the security problems of field based protection are proposed.

1. Introduction

Database encryption and authentication at the field level is not usually recommended: using encryption to hide individual data elements is vulnerable to ciphertext searching; using cryptographic checksums to authenticate the integrity of individual data elements is vulnerable to plaintext or ciphertext substitution. These problems do not arise with record based protection, where each record is encrypted or authenticated as a unit.

---

[1]Research supported in part by NSF Grant MCS80-15484. Author's present address: Computer Science Lab, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025.

There are, however, disadvantages to record based protection. Projections cannot be applied before decryption or authentication to eliminate unneeded fields, and selections cannot be performed without decrypting up through the fields over which the selection is to be performed. With field based encryption, unneeded fields can be projected out and individual fields decrypted or authenticated. Moreover, record based protection is not suitable for applications that require keeping all but one or two short (e.g. one byte) fields in the clear for fast retrieval. An example of such an application is described later.

The objective of this paper is to propose techniques for secure encryption and authentication at the field level. The techniques we describe all use the Data Encryption Standard (DES) [6], though they are readily adapted to any conventional block encryption algorithm with the property that any one bit change to either the key or plaintext unpredictably affects each ciphertext bit.

For a given data element X, we let $E_K(X)$ denote the encryption of X under the secret key K. If X is less than 8 bytes long (the block size of the DES), then X will be replicated as many times as necessary to fill the block. If X is more than one block long, cipher block chaining (CBC) will be used during encryption to chain the blocks together [6] (also in [5]).

Section 2 discusses the secrecy problems of field encryption, and shows how these problems can be solved. Section 3 addresses the integrity problems of field authentication, and gives a solution to these problems. Section 4 describes a database application well suited to the field encryption and authentication techniques proposed here. Section 5 discusses an alternative approach that blends record and field based techniques, comparing it with the proposed techniques. Section 6 concludes.

2. Field Encryption

Consider a file of N records where each record has M fields. The objective is to conceal the data in some field j of every record. The obvious way of doing this is to encrypt the field under a secret database key K. Letting X_{ij} denote the plaintext value for record i, field j, the ciphertext value $C_{ij} = E_K(X_{ij})$ is thus computed and stored in record i (i = 1,...,N).

2.1. Security Problem

It is well known that this method of encrypting a field is not secure, especially when the field has low entropy relative to the total number of records (e.g., see [5, 3]). If $X_{ij} = X_{pj}$ for records i and p, then $C_{ij} = C_{pj}$; thus, an intruder may be able to deduce plaintext values by searching for records with identical ciphertext. In particular, if $C_{ij} = C_{pj}$ where X_{pj} is known, then one can infer that $X_{ij} = X_{pj}$. Or, if the distribution of values in the domain of the jth field is known, then the plaintext values can be inferred from the distribution of ciphertext values. For example, if the domain consists of two values 0 and 1, where 1 is expected to occur twice as often as 0, and the ciphertext "X:JT57.%" occurs about twice as often as the ciphertext "8N..64*#", then it is easy to deduce which records have 0 and which have 1. This method of attack, sometimes called *ciphertext searching*, can also be performed across fields within a record or among different records when the same key is used to encrypt all records and all fields. Because of the threat of ciphertext searching, field encryption has not been recommended for applications requiring a high level of cryptographic security.

2.2. Solution

The preceding method of field encryption is insecure because repetitions of data values are encrypted under the same key. Our solution is simply to use a distinct cryptographic key for each data element; that is, for each record, and for each field within a record. Letting K_{ij} denote the *element key* for record i, field j, the value X_{ij} is then encrypted as $C_{ij} = E_{K_{ij}}(X_{ij})$. Flynn and Campasano [8] proposed using a different key for each record; we are extending their approach to individual fields within a record.

We first describe techniques for generating element keys K_{ij}, and then discuss encryption of the data.

2.3. Key Generation

We will assume that the first field in every record uniquely identifies the record; i.e., it is the primary key for the database (a primary key is not to be confused with a cryptographic key). We will also assume that this field is at most 8 bytes long, and that it is not encrypted. Let $R_i = X_{i1}$ be the

identifier for record i, and let F_j be an identifier for field j. An element key K_{ij} is defined by $K_{ij} = g(R_i, F_j, K)$, where g is a *key generating function*, and K is the secret database key. Note that key generation must be a cryptographic function of a secret key so that the element keys will be secret. Before describing possible functions for g, we state three security requirements that should be satisfied:

1. The probability of getting repetitions of keys should be low, especially within a field.

2. It should be computationally infeasible to obtain new information about an unknown data element X_{ij} from the ciphertext values, even if some plaintext elements are known, or the distribution of plaintext values is known.

3. It should be computationally infeasible to determine one element key from other element keys.

Property 1 is needed so that equal plaintext elements are encrypted under different keys, and therefore have different ciphertexts, with high probability. If none of the keys repeats or is "weak" [4, 10], then Property 2 will be satisfied as well -- at least to some extent. This is because equal (or unequal) plaintext elements have unpredictably different ciphertexts with the DES for even one bit key changes, thereby foiling ciphertext searching attacks. Note that some key repetitions can be tolerated as long as they do not reveal new information about the plaintext. Assuming that a repeating key is suspect only when $C_{ij} = C_{pq}$ for two ciphertexts, we require that the a posteriori probability $prob[K_{ij} = K_{pq} \mid C_{ij} = C_{pq}]$ be the same as the a priori probability $prob[K_{ij} = K_{pq}]$. Property 3 is needed so that if a cryptanalyst obtains the key for one data element, other data elements in the database remain protected.

In addition to satisfying these security problems, the key generator should be efficient. It should be possible to generate any key K_{ij} without generating other element keys in record i or field j. When evaluating different key generators, we will consider not only the effort required to generate a single K_{ij}, but also the effort required to generate all element keys in one record (e.g., to decrypt or authenticate an entire record), and the effort required to generate multiple element keys in a single field (e.g., to decrypt or authenticate a field in a multiple record access).

We will discuss five possibilities for the key generator g ("$\oplus$" is the exclusive-or operator):

1. $K_{ij} = E_{K_j}(R_i)$ where $K_j = E_K(F_j)$
2. $K_{ij} = R_i \oplus K_j$ where $K_j = E_K(F_j)$
3. $K_{ij} = E_{K_i}(F_j)$ where $K_i = E_K(R_i)$
4. $K_{ij} = K_i \oplus F_j$ where $K_i = E_K(R_i)$
5. $K_{ij} = E_K(R_i \oplus F_j)$.

In all five methods, we assume that unique identifiers are padded as necessary to fill 8 bytes. Figure 1 illustrates the different methods, where the dashed lines represent key flow into encryption, and the solid lines represent data flow. Because encryption is one way in the key, it is not possible to compute backwards along dashed lines (i.e., a known plaintext attack).

Method 1 first generates a *field key* K_j by encrypting F_j under K, and then generates the element key by encrypting R_i under K_j. Because both encryptions are one way in the keys, compromise of some element key K_{ij} will not compromise the field key K_j; because K_j is needed to compute element keys, other element keys cannot be determined from K_{ij}.

Because encryption is a one-to-one function, and R_i does not equal R_p for any two records i and p, $E_{K_j}(R_i)$ does not equal $E_{K_j}(R_p)$. This does not, however, guarantee that no keys for field j will duplicate, since only 56 bits of the output blocks are used (the 8 parity bits are discarded). We expect, however, duplicate keys to be rare because of the randomness of the DES (assuming the number of records is much smaller than the size of the key space, which is approximately 7 times 10^{16}).

The disadvantage of method 1 is that two encryptions are needed to compute each element key in a record. This potentially triples the effort required to encrypt or decrypt an individual record. On the other hand, for multiple record accesses to a particular field j, the element keys in field j can be obtained with one additional encryption each once K_j is computed.

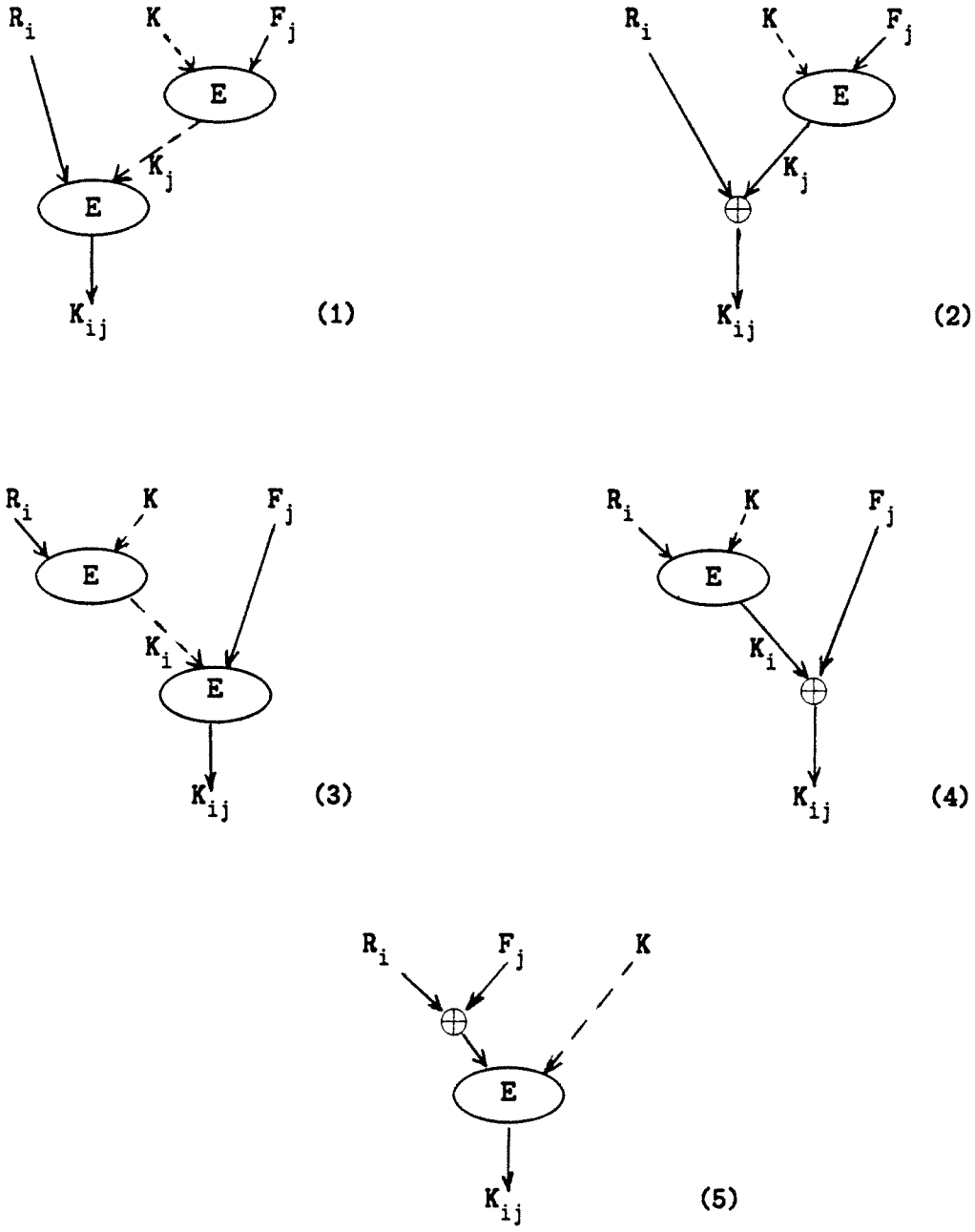

Figure 1. Key Generators.

Method 2 is similar to method 1 in that it first generates a field key. But the second encryption is replaced with an exclusive-or operation to speed computation of element keys. Because the element keys in a particular field are quickly derived from the field key, multiple record accesses to the field are extremely efficient. The problem with this approach is that if a key K_{ij} is compromised, then the field key K_j, and therefore every element key in field j, is easily computed (we assume record identifiers are known).

Method 3 switches the order of the encryptions in method 1; that is, first a *record key* K_i is computed by encrypting the unique identifier R_i under K, and then the element key is computed by encrypting F_j under the record key. This approach also protects keys when an element key is compromised. Although method 3 does not guarantee distinct element keys within a field, we again expect duplicate keys to be rare. Because the element keys are obtained by one encryption from the record key, method 3 allows faster access to all fields within a record than method 1. On the other hand, multiple record accesses to a single field are slower.

Method 4 is similar to method 3 in that it first generates a record key. But to further speed access to data elements, it replaces the second encryption with an exclusive-or operation. With this approach, the time to encrypt or decrypt an entire record is competitive with record based encryption. Multiple record accesses to a single field, however, still require an encryption (of the record key) to obtain the element keys in the field. Like method 2, method 4 does not protect keys from exposure when an element key K_{ij} is compromised; this is because the record key K_i, and therefore every element key in record i, is easily computed. But for many applications this may be acceptable since the keys for other records are not exposed.

Method 5 computes each element key K_{ij} by encrypting R_i exclusive-ored with F_j under K. This method has the advantage of never requiring double encryption to compute an element key. Because it always requires an encryption, it is slower than method 2 for multiple record accesses to a single field, and it is slower than method 4 for single record encryption and decryption. With method 5, duplicate keys within a record or within a field should be rare and unpredictable. On the other hand, $K_{ij} = K_{pq}$ will occur whenever $R_i \oplus F_j = R_p \oplus F_q$ for two records. Because unique identifiers are not concealed, this property could be useful in ciphertext searching attacks. Moreover, if K_{ij} is compromised, then K_{pq} is also exposed.

Figure 2 summarizes the discussion of the five methods. Methods 1 and 3 provide the greatest security, but are the least efficient. Methods 2 and 4 are most efficient, and would be attractive for applications where the risk associated with multiple key exposures from key compromises can be tolerated. The reader is invited to think of other methods.

Method	Number of encryptions to compute each K_{ij} in record i	Number of encryptions to compute each K_{ij} in field j	Element keys exposed if K_{ij} compromised
1	2	1	0
2	1	0	All keys in field j
3	1	2	0
4	0	1	All keys in record i
5	1	1	All K_{pq} where $R_p \oplus F_q = R_i \oplus F_j$

Figure 2. Comparison of Key Generators

With all five methods, there is some danger of getting weak keys. If protection against weak keys is desired, unused bits in the identifier field could be randomly set to 0 or 1, and then flipped as needed. (This technique could also be used to change keys.)

2.4. How to Encrypt

Figure 3 illustrates the encryption and decryption of an element X_{ij}. If X is less than the 8-byte block size of the DES, then X is replicated as many times as necessary to fill the block. If X_{ij} exceeds the block size, then the encryption is performed using cipher block chaining with initialization block I. Although a distinct value of I could be chosen for every record, or even

every field within a record, this is unnecessary since keys are secret and do not repeat. We therefore propose to set I to the all zero block, so that encrypting a single block in standard block mode will be equivalent to encrypting that block as the first block in cipher block chaining (cipher block chaining begins by encrypting the first block exclusive-ored with I).

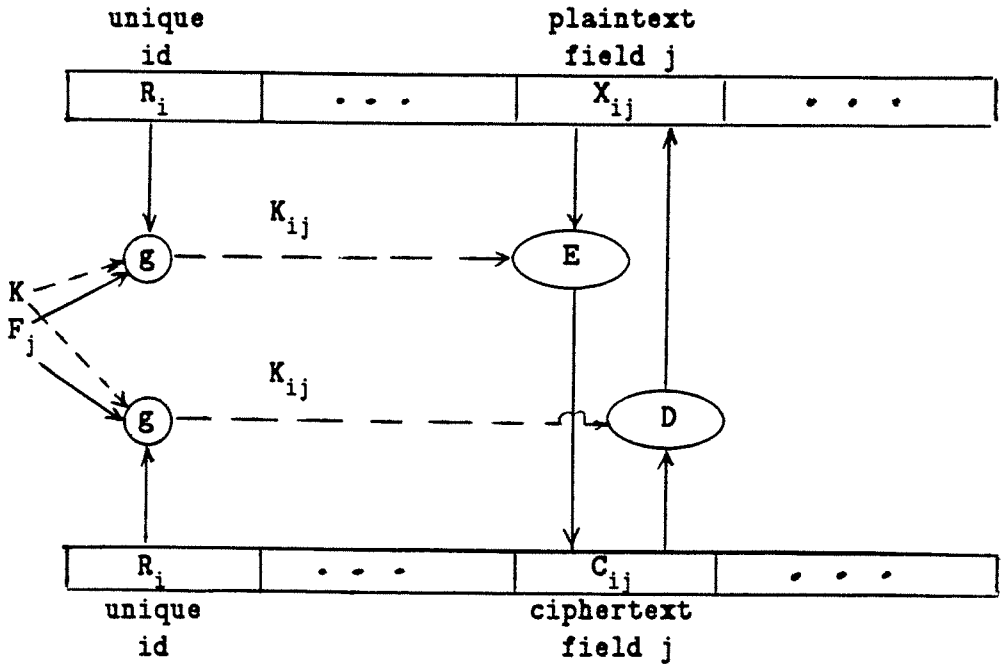

Figure 3. Field Encryption (E) and Decryption (D).

To retrieve information based on the value of an encrypted field j, all values in the field must be decrypted. For example, if employee salaries are encrypted, then it is not possible to retrieve the records of employees making more than 30,000 without decrypting the salary field of every record. It may be possible to speed retrieval by placing a secondary index on salary, and using field encryption to hide the pointers in the index; we do not yet know whether this can be done securely without negating the performance benefits gained by using the index.

One disadvantage of field encryption is that is causes expansion of short

fields. For example, using the DES would expand short fields to 64 bits. This message expansion could be avoided by using stream encryption (e.g., see [5]). Letting t be the length of X_{ij} in bytes ($t < 8$), X_{ij} is then encrypted by exclusive-oring it with the first t bytes of the element key K_{ij}. This approach does not, however, provide as much integrity as block encryption since there is less redundancy (see next section).

Stream encryption could also be used with longer fields, and would be more efficient than block encryption. If X_{ij} is exactly one block, then it would be much faster to compute $C_{ij} = X_{ij} \oplus K_{ij}$ than to compute $C_{ij} = E_{K_{ij}}(X_{ij})$. If X_{ij} is longer than one block, then output feedback mode could be used to generate a keystream from K_{ij} and seed I. With stream encryption, our method of field encryption would approximate a one-time pad, where each key K_{ij} (or stream generated by K_{ij}) is a separate pad.

Unfortunately, stream encryption is vulnerable to a known plaintext attack when the data is not constant and keys are reused [2, 5]. If some X_{ij} is known, it is easy to compute the element key K_{ij} (or key stream) by exclusive-oring the known plaintext with the stored ciphertext. Once K_{ij} is known, updated ciphertext for the field element is easily decrypted. Stream encryption is also vulnerable to a ciphertext only attack if keys are reused. Because keys cannot be reused securely, stream encryption should not be used with data that is potentially modifiable.

3. Field Authentication

The objective here is to verify that information retrieved from the database has not been changed. The usual way of doing this is to store a cryptographic checksum (authenticator) with each record. The checksum is a cryptographic function of the entire record, and is computed using a technique such as cipher block chaining with a secret key. When the record is retrieved, the checksum is recomputed from the data fields. If there is a match, then the probability is only $1/2^n$ that the record has been modified, where n is the length of the checksum in bits (64 if a full output block from the DES is used).

This strategy can degrade performance when some of the fields are long, and these fields are not used in a query. Because authentication requires

access to entire records, projections cannot be used to suppress unneeded fields. Cryptographic checksums computed at the field level would allow such projections.

3.1. Security Problem

Unfortunately, using checksums with individual fields introduces a security problem. Let S_{ij} be the checksum of value X_{ij} in record i, field j. If S_{ij} is a function only of X_{ij} (plus the secret key K), then the pair (X_{ij}, S_{ij}) in record i can be replaced by the pair (X_{pj}, S_{pj}) in record p without detection. (If field j is encrypted, then the replacement is performed by substituting ciphertext instead of plaintext.) Furthermore, the pair (X_{ij}, S_{ij}) can be replaced with the pair (X_{iq}, S_{iq}) in another field q of record i.

3.2. Solution

Our solution here is essentially the same as for encryption: make the key used to compute the checksum for an element X_{ij} a function of the record identifier R_i, the field identifier F_j , and the secret key K. Then replacing a pair (X_{ij}, S_{ij}) in record i with another pair should be detectable.

Consider first the case where the jth field is less than 8 bytes. Recall that our method of encryption replicates the field as many times as necessary to fill an 8-byte block. Because of this redundancy, the ciphertext $S_{ij} = C_{ij} = E_{K_{ij}}(X_{ij})$ can serve as the checksum. If X_{ij} is to be stored in the clear, then both X_{ij} and S_{ij} are stored in the record, and the probability is $1/2^{64}$ that a change to X_{ij} will go undetected. (The storage requirements for S_{ij} can be reduced by truncating it, say, to 32 bits, in which case the probability of not detecting a change is $1/2^{32}$.) If X_{ij} is to be stored as ciphertext C_{ij}, then C_{ij} serves the dual purpose of providing both secrecy and authenticity; no additional storage or processing time is required for authentication. If there are t bytes of data and 8 - t bytes of redundancy, then the probability is at most $1/2^{64-8t}$ that a change will go undetected. If this is not enough for large t (e.g., for t = 7 the probability is 1/256), then the field can be treated as a full block as described next.

Consider next the case where the jth field is exactly one block. Again,

if X_{ij} is to be stored in the clear, then $S_{ij} = E_{K_{ij}}(X_{ij})$ can serve as the checksum, and the probability is $1/2^{64}$ that a change will go undetected. But if X_{ij} is to be stored as ciphertext C_{ij}, C_{ij} cannot serve the dual purpose of providing both secrecy and authenticity unless X_{ij} has enough natural redundancy, e.g., as with English language text. If any 8-byte block of data is accepted as valid plaintext for the field, then there is no way of detecting changes when the ciphertext is decrypted. One simple solution is to define $S_{ij} = E_{K_{ij}}(C_{ij})$; that is, the checksum is the encrypted ciphertext. Both C_{ij} and S_{ij} are then stored in the record.

Finally, consider the case where the jth field is more than one block; thus, cipher block chaining is used during encryption. Let $last(C_{ij})$ be a function that returns the last block of ciphertext $C_{ij} = E_{K_{ij}}(X_{ij})$. If X_{ij} is to be stored in the clear, the checksum is defined by $S_{ij} = last(C_{ij})$; the probability is $1/2^{64}$ that a change will go undetected. If X_{ij} is to be stored as ciphertext, the ciphertext again cannot serve the dual purpose of secrecy and authenticity unless the data has enough natural redundancy. One method for computing the checksum is to encrypt C_{ij} using cipher block chaining and keep the last block; that is, $S_{ij} = last(E_{K_{ij}}(C_{ij}))$. This, however, has the disadvantage of requiring two encryption passes over the field. A more efficient method is to append a manipulation detection code to the plaintext, which is computed using noncryptographic means (e.g., by adding the blocks of plaintext modulo 2^{64}). Jueneman, Matyas, Meyer [11] discuss this and other approaches.

An attractive property of field checksums is that it is possible to tell which field (or fields) have been changed when validation fails (a checksum is also placed on the unique identifier so that when a validation error for record i, field j occurs, it is possible to tell whether the change occurred to X_{ij} or to R_i). With record checksums, it is not possible to pinpoint the changes to specific fields.

The main drawback with field checksums is message expansion. If there are M fields in the database, then M blocks are needed for checksums when the data is stored in the clear. By comparison, only 1 block is needed for a record checksum. On the other hand, field checksums allow fields (except for the unique identifier) to be projected out during query processing, so that the

total volume of data processed may decrease substantially.

4. An Application: Protecting Classified Data

Field encryption can be used to hide as little as one or as many as all fields of the database. It is most useful for applications where some fields must be kept in the clear to allow for fast retrieval or for retrieval by processes that do not have access to the secret key K. An example of such an application arose at the Air Force Summer Study on Multilevel Database Management Security [1]. The objective of the summer study was to make recommendations leading to the development of secure systems that handle classified data at different levels, where the users of the system are also cleared to different levels.

A study group headed by Clark Weissman proposed a near-term solution to this problem based on using a commercially available database management system. Because the database system may have security holes, all access to the database system is confined to a trusted (verified) interface. The trusted interface stores a label in each record (or field within a record) giving the classification level (Top Secret, Secret, Confidential, Unclassified) of the data in the record (or field). The classification labels are used by the trusted interface to determine what data a given user is allowed to access, based on the user's clearance. To ensure the integrity of the data and of the classification labels, the trusted interface uses cryptographic checksums, which are validated on retrieval.

In order that the database system can perform operations such as select and join on the database, the data is stored in the clear. Doing so, however, admits the possibility of a Trojan Horse in the database system leaking classified information in its responses to queries (e.g., the response to a predetermined query may be an unclassified value equal to a classified one). To protect against such Trojan Horses, the classification labels should be concealed. This can be done with the field encryption scheme proposed here, where the secret key K is known only to the trusted interface, and all encryption and decryption is done in the interface. Even if there are only a few different plaintext classification labels, each record can be expected to have a different ciphertext classification label. If the plaintext labels are short (e.g., one byte), replicating the field for encryption will produce plenty of redundancy so that the ciphertext can provide authenticity as well as secrecy. An alternative method of concealing the labels, suggested by the study group,

is to store them in a separate database managed by the trusted interface. This approach imposes a penalty on performance, however, and increases the complexity of the trusted interface and the verification effort.

The field authentication scheme proposed here can protect the integrity of the data and its classification at the field level. This is done simply by concatenating each data element X_{ij} with its classification label before computing the checksum.

Figure 4 illustrates how classified data in record i is protected when classification is at the record level, and field M gives the classification label. The stored record shows the ciphertext classification $C_{iM} = E_{K_{iM}}(X_{iM})$, where $K_{ij} = g(R_i, F_M, K)$; and the checksum $S_{ij} = E_{K_{ij}}(X_{ij} \| X_{iM})$, where $K_{ij} = g(R_i, F_j, K)$, for some key generator g, and "||" denotes concatenation.

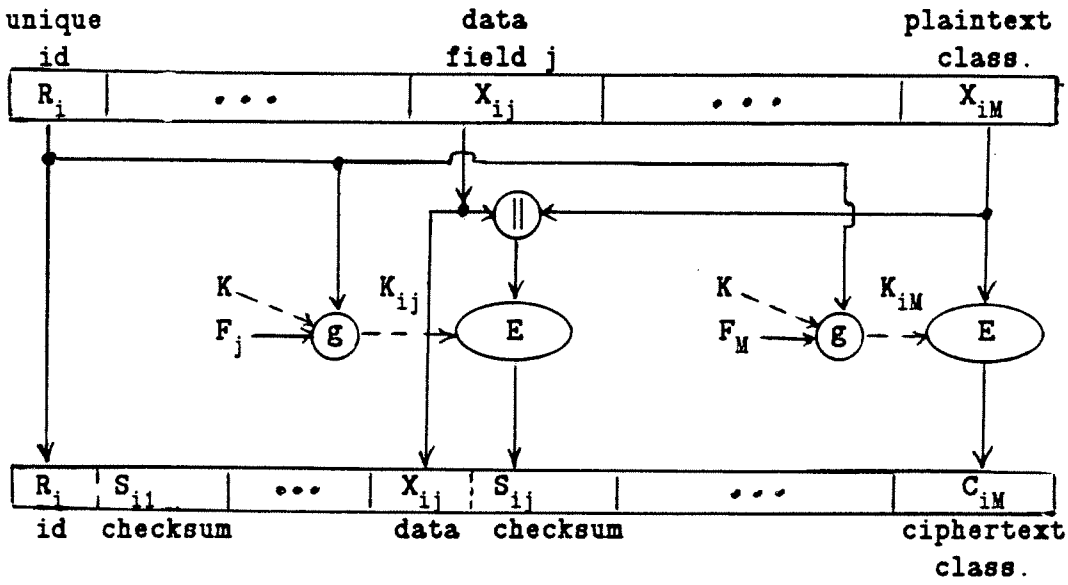

Figure 4. Integrity Protection for Classified Data.

Because the fields are individually authenticated, the database system can perform projections to suppress unwanted fields. The unique identifier

and classification label fields, however, are never projected out. The database system is also free to change the physical locations of records since the checksums are not a function of these locations.

When the data is returned to the trusted interface, only the returned fields are checked for changes. Changes to other fields are not detected until those fields are retrieved. Individual fields can be updated by the interface without the need to obtain the entire record.

5. An Alternative Approach

During our search for secure field based techniques, we considered an approach to database protection proposed by Davida, Wells, and Kam [3]. Their approach blends record and field based techniques in an intriguing way that allows an individual field to be extracted from the ciphertext (using a read subkey for the field), even though records are encrypted as a unit (to foil ciphertext searching). The trick to doing this comes from the Chinese Remainder Theorem. Like the scheme we have proposed, their scheme causes message expansion.

Our analysis of the approach revealed several disadvantages compared to the scheme we have proposed. First, it is more costly to decrypt a single field element because a divide operation (specifically a reduction modulo the read key for the field) must be performed over the complete record. Second, it is not possible to project out fields without performing a computation over complete records, and to do such a projection requires knowing the field keys. Thus, an untrusted database system could not do such projections, as in the classified database application. Third, it is not possible to update an element without recomputing the ciphertext for the complete record. Fourth, it is not applicable for concealing a single short field, such as a classification label.

6. Conclusions

We have proposed techniques for overcoming the security problems of field based protection. The principle idea behind these techniques is to generate a different key to encrypt (or authenticate) each data element in the database. This principle is similar to that behind a stream cipher, where different portions of the plaintext are exclusive-ored with different portions of a nonrepeating key stream. Indeed, with Diffie's and Hellman's [7, 9]counter

method of stream encryption, the ith key character in a stream is generated from i, making random access into an encrypted file possible. One difference between their method and ours is that we generate an element key K_{ij} from both the ith record identifier and the jth field identifier, which permits records to be physically moved or even new fields added without reencrypting the entire file. Another is that because we propose to use block encryption, the data can be modified and the same element keys reused. (As noted earlier, with stream encryption, portions of the key stream can never be reused.) Moreover, with block encryption, each element key can encrypt as little as one bit (though the ciphertext will be 64 bits) or as much as several blocks.

There are two disadvantages to the techniques we have proposed over record based techniques: message expansion, and the overhead of computing element keys. These are compensated by two advantages; a single low entropy field can be concealed within a record, and fields can be projected out before decryption and authentication. The best strategy may be a hybrid approach that groups certain fields together for protection purposes. For a given application, the optimal groupings would be determined by the physical representation of the data and the access patterns to the database.

This is a preliminary report, and we leave unresolved many issues relating to the security and cost of the proposed techniques. We are exploring these issues, and welcome comments and suggestions.

Acknowledgments

Thanks to Clark Weissman for stimulating me to think about the problem, and to Peter Denning for helpful discussions.

References

1. Multilevel Data Managment Security. Committee On Multilevel Data Managment Security, Air Force Studies Board, National Research Council, 1982.

2. Bayer, R. and Metzger, J. K. "On the Encipherment of Search Trees and Random Access Files." *ACM Trans. on Database Syst. 1*, 1 (March 1976), 37-52.

3. Davida, G. I., Wells, D. L. and Kam, J. B. "A Database Encryption System with Subkeys." *ACM Trans. on Database Systems 6*, 2 (June 1981).

4. Davies, D. W. Some Regular Properties of the 'Data Encryption Standard' Algorithm. In *Advances in Cryptology: Proc. of CRYPTO 82*, D. Chaum, R. Rivest, A. Sherman, Ed.,Plenum Pub. Co., 1983.

5. Denning, D. E.. *Cryptography and Data Security*. Addison-Wesley, Reading, Mass., 1982.

6. Data Encryption Standard. National Bureau of Standards, Washington, D.C., Jan., 1977. FIPS PUB 46

7. Diffie, W. and Hellman, M. "Privacy and Authentication: An Introduction to Cryptography." *Proc. IEEE 67*, 3 (Mar. 1979), 397-427.

8. Flynn, R. and Campasano, A. S. Data Dependent Keys for a Selective Encryption Terminal. Proc. NCC, Vol. 47, AFIPS Press, Montvale, N. J., 1978, pp. 1127-1129.

9. Hellman, M. E. On DES-Based, Synchronous Encryption. Dept. of Electrical Eng., Stanford Univ., Stanford, Calif., 1980.

10. Jueneman, R. R. Analysis of Certain Aspects of Output Feedback Mode. In *Advances in Cryptology: Proc. of CRYPTO 82*, D. Chaum, R. Rivest, A. Sherman, Ed.,Plenum Pub. Co., 1983.

11. Jueneman, R. R., Matyas, S. M. and Meyer, C. H. Authentication with Manipulation Detection Code. Proc. 1983 IEEE Symp. on Security and Privacy, IEEE, Apr., 1983.