

# A Schematic Technique Using Data type Preserving Encryption to Boost Data Warehouse Security

M.Sreedhar Reddy<sup>1</sup> Prof.M.Rajitha Reddy<sup>2</sup> Prof R.Viswanath<sup>3</sup> Prof.G.V.Chalam<sup>4</sup> Prof.Rajya Laxmi<sup>5</sup> Prof.Md.Arif Rizwan

<sup>1</sup>Dept of CSE SSIETW,Hyderabad,AP,India

<sup>2</sup>Dept of CSE ,SSIETW,Hyderabad, AP,India

<sup>3&6</sup> Dept of CSE ,REC,Hyderabad, AP,India

<sup>4</sup>Dept of commerce&BA, A N University, Guntur , AP,India

<sup>5</sup>Dept of IT  
GITAM, VIZAG. AP,India

## Abstract

An ingenious data warehouse habitually contains information which must be painstaking enormously sensitive and proprietary. Protection of this information, as important as it is, is too often thorny by the presence of assorted computing environments, managerial issues, difficulties in controlling data distribution, and slipshod attitudes towards information security. We present a method of in progression fortification based on an encryption scheme which preserves the data type of the plaintext resource. We suppose that this method is particularly companionable for multifaceted data warehouse environments

**Key words:** Query, prevention, detection, encryption, decryption, datawarehouse, datamining

## 1. Introduction

Nowadays numerous IT security professionals it sounds as if believe that their restraint has three goals: confidentiality, integrity, and availability. This paper explores a different loom to the problem of IT security goals. Guided by the desire to anchor IT security in the most familiar features of ordinary human life, the approach is two pronged.

In terms of “security”; continues by analyzing them; and, based upon this analysis, proposes a basic definition of “Information Technology security.”

Data warehouse technology has, in modern years, provides its corporate executives and business planners with extraordinarily powerful decision support tools. Data warehouses can tell us **which** products to manufacture, **where** to locate factories and **how** to gain market share. They can be used to answer questions which weren't even contemplated when the data warehouse was built. Most remarkable of all—these feats can be accomplished using data extracted from existing operational systems.

It has become gradually more perceptible however, that the data warehouse is an alluring

goal for snoopers. To envisage that you are a annoyed employee, or industrial surveillance representative who manages to gain access to an organization's computer system. Now, would we likely to find out?

- . Data grouped into subject data areas so that you can quickly find the items of interest.
- . Accurate and complete information that has been painstakingly reviewed for correctness.
- . Time-indexed data so trends can be easily identified.
- . Summarized information with the ability to drill down for details.

Conceptually, the data warehouse process consists of three simple steps:

- 1) Extract data from the operational system,
- 2) Load the extracted data into data warehouse tables, and
- 3) Query the data warehouse to obtain decision support information.

Data is at risk during each of these phases. Several factors render data warehouses particularly susceptible to attack:

- 1) Extracted data is frequently transmitted of over insecure communication lines.

- 2) Extracted data is stored on a variety of computer systems and removable media which may have only minimal security.
- 3) The extraction process produces intermediate files and load files which contain sensitive information, but may not be well-protected.
- 4) Maintaining proper security attributes for the data warehouse tables is extremely time consuming in the face of constant organizational change.
- 5) Users often retrieve data from the data warehouse and create a "data mart," leading to widely distributed copies of sensitive data.
- 6) Sound security practice is often undermined because the data warehouse development effort is a "high visibility" project with a tight schedule.

Furthermore that research into the unique aspects of data warehouse security is still in the early stages, additional vulnerabilities will certainly be identified in the future

## 2. Problem Statement

When to develop a cryptographic approach to data warehouse security which could be practical in the complex, heterogeneous environments encountered in the business world, we recognized certain decisive objectives:

- 1) The approach should work with any combination of commonly used relational databases. (This rules out requiring binary data storage or other database-dependent features.)
- 2) It must function on multiple hardware platforms and operating systems.
- 3) It must appropriately encrypt and decrypt data on machines with different character sets (e.g., ASCII and EBCDIC).
- 4) The strength of the encryption algorithm should be comparable to widely used, state of the art technology.
- 5) To add encryption to an existing database should require no changes to the structure of the database. (Neither should any application changes be required to access non-encrypted fields.)
- 6) Encryption should occur as early as possible in the extraction process and decryption should occur at the last possible moment.
- 7) It could not be dependent on a particular programming language.
- 8) It should be "fail safe." (Any likely failure mode should be such that that access to the data is *denied*.) These requirements, based on our business requirements, constituted a formidable challenge. During the course of our research, however, it became apparent that the tractability of the problem could be improved significantly if

we could find a way to preserve the original data type across the encryption and decryption transformations.

## 3. Proposed Solution

Cipher text bears roughly the same resemblance to plaintext as a hamburger does to a T-bone steak. A social security number, encrypted using the DES encryption algorithm, not only does not resemble a social security number but will likely not contain any numbers at all. A database field which was defined to hold a nine-character social security number would not be able to store the DES-encrypted version of the data.

A Visual Basic program would not read it. A graphical interface would not display it. There would be nothing that you could do with the encrypted social security number unless you had made extensive provisions for changes in data format throughout your application and physical database design.

### 3.1 Basic Data type Preservation

Our method reduces the need for changes to database structures and applications by preserving the data type of the encrypted field.

Data type preservation simply means that each cipher text field is as valid as the plaintext field it replaces. The key to our approach is defining an appropriate alphabet of valid characters and performing all operations within the constraints of the defined alphabet. Each different data type requires a judicious choice of alphabet. An alphabet consisting of numeric digits ("0123456789") could be used to encrypt most number data, such as social security numbers (e.g. 234-415-6978). (The dashes, not included the chosen alphabet, are copied unchanged to the corresponding positions in the cipher text output.)

Other alphabets, such as all printable ASCII characters, all characters shared by ASCII and EBCDIC, or all hexadecimal digits can be used to encode a variety of common data types.

### 3.2 The Approach

The first processing step involves replacing each plaintext character in the string by an integer that represents its position, or index, within the chosen alphabet. This number is between zero and one less than the total number of characters in the alphabet. If a plaintext character is not in the valid alphabet, it is copied to the output and removed from the string to be encrypted.

Example:

Plain text=" crazy"

Alphabet="abcdefghijklmnopqrstuvwxy z"

- Step1: Assign Index values  
 Index values=7,4,11,14,11  
 Step2: Add position sensitive offsets  
 Offsets=10,4,18,25,8  
 New index values=12,9,6,9,17  
 Step3: Shuffle the index value string  
 Shuffled values=3,18,17,10,9  
 Step4: Convert Back to desired data type  
 Cipher text-“crazy”

Figure 1 includes a worked example of the basic algorithm.:

After the alphabet index values have been assigned, we add a varying integer “offset” to each. We use modular addition to ensure that we generate only valid characters (i.e. characters which are contained in the alphabet). Remember that “modular” addition means adding two numbers and then determining the remainder after division by a constant “modulus” value. In the example above the alphabet size is 26 so, for example,  $18 + 11 \pmod{26} = 3$ . The actual offset values are generated based on a portion of the key being used to encrypt the data. This step ensures that long series of identical characters (such as 20 blanks at the end of a character field) will not encrypt identically. After adding the offsets, the entire string is shuffled. The shuffling method varies according to a permutation-invariant property of the index values, such as a sum or exclusive-or, of all values.2

The shuffling step helps to ensure that plaintexts with common prefixes or suffixes do not produce cipher text with common prefixes or suffixes. Once the encoding process is complete, each index value is mapped to the appropriate character in the alphabet. To recover the plaintext from the cipher text, one replaces the cipher text characters by their alphabet index values, “unshuffles” the string, regenerates the offset values, subtracts modularly on an integer-by-integer basis and substitutes the appropriate alphabet character.

Two enhancements to the above algorithms may be used to deal with certain data specific situations:

- First, in order to ensure that the encoded values of two single character strings with adjacent characters are not sequential (for example, we would not want “b” to encrypt as “y” whenever “a” encrypts as “x”), the alphabet itself can be shuffled based on a portion of the encryption key.
- Second, in order to inhibit guesses based on encrypted character

permutations, we can “ripple” the data from left to right and from right to left. This is done by hashing the key into a “starter-digit” and adding adjacent values pair wise. For example, the string of index values “1, 2, 3” might be rippled into “23, 5, 40” as follows (assuming a 55 character alphabet):

starter value = 72 (obtained by hashing the encryption key)

Adding left to right:

$$72 + 1 \pmod{55} = 18$$

$$16 + 4 \pmod{55} = 20$$

$$20 + 3 \pmod{55} = 23$$

Adding right to left:

$$23 + 72 \pmod{55} = 40$$

$$20 + 40 \pmod{55} = 5$$

$$18 + 5 \pmod{55} = 23$$

Applying this same method to the permutation “3, 2, 1,” on the other hand, ripples it to “27, 7, 40” and the fact that the two strings contain the same characters is disguised.

2 A variety of techniques could be used to generated the offsets and shuffling pattern, including the use of pseudo-random number generators.

### 3.3 Enhanced Encryption

While the encoding scheme presented above is sufficient to deter casual attacks, more substantial protection is required to protect sensitive data in the data warehouse. The approach described above can be combined with well-known encryption algorithms, such as DES or IDEA, to significantly increase the attacker’s burden. The basic idea is to use an established algorithm of known strength to produce the “offset” values.

The DES algorithm takes as input a 64-bit input block and a 64-bit key (56 key bits and 8 parity bits) and uses these two values to produce a 64-bit output. The cipher text output can be decrypted using the same key. For all practical purposes, the only way to break the scheme is by an exhaustive search of the key space.

DES, like any block cipher, can be operated as a stream cipher in “cipher-feedback” mode. We use this mode to encrypt one index value at a time. At the end of each encryption pass, we also shift the plaintext data into the DES input register. This process is illustrated as follows:

Let the alphabet index values of the n character, plaintext input string be represented

By  $i_1 i_2 i_3 i_4 i_5 \dots i_n$  Let the 64-bit DES initial value required by cipher-feedback mode be constructed based on a portion of the encryption key

$$H(K) = a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 = A$$

Where each subscripted “a” value represents an 8-bit number (“0” to “255”). Let the output of the DES algorithm, using a key of “k” and an input of “A,” be represented by

$$E_k(a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8) = b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8.$$

The first transformed index value is the modular sum  $z_1 = b_8 + i_1 \pmod{l}$ . Where “l” represents the alphabet length.

At this point, a new DES input value, A, is constructed as  $A_2 = b_2 b_3 b_4 b_5 b_6 b_7 b_8 i_1$  and a new DES output is obtained

$$E_k(b_2 b_3 b_4 b_5 b_6 b_7 b_8 i_1) = c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8.$$

The second transformed index value is the modular sum  $z_2 = c_8 + i_2 \pmod{l}$ .

Note that the use of an addition operator is required, instead of the usual exclusive-or operator, is required to ensure that the data type is preserved.

After n such steps, during each of which a single input index value is transformed, we have an encrypted index-value string

$$Z = z_1 z_2 z_3 z_4 z_5 \dots z_n$$

We claim that recovering the string,  $i_1 i_2 i_3 i_4 i_5 \dots i_n$ , from the transformed string,  $z_1 z_2 z_3 z_4 z_5 \dots z_n$ , without knowledge of the key, K, is as difficult as breaking the DES algorithm itself. When using cipher-feedback mode, DES decryption, per se, is never invoked. Reversing the transformation is done by subtracting the low order DES output from the transformed index value. Below is a summary of the algorithm.

#### Setup responsibilities:

- 1) Select an encryption key with enough bits for the encryption algorithm key, encryption algorithm initial value and any basic processing stages.
- 2) Select a suitable alphabet to support the data type of the data to be encrypted.
- 3) Shuffle the alphabet according to a scheme based on the key. For each encrypted field:
- 4) Scan the input buffer for characters which are not included in the chosen alphabet. Move all invalid characters unchanged to their corresponding positions in the cipher text output buffer.
- 5) Move the index values of all valid characters to adjacent positions in a work buffer.
- 6) Add position-sensitive offsets according to a key-dependent scheme.
- 7) Scuffle the work buffer positions according to a data-dependent scheme.

8) “Ripple” the work buffer by calculating a key-based starter number and modularly adding pair wise from left to right then from right to left.

9) Set the cipher-feedback initial value using the chosen key.

10) Calculate the modular sum of the first work buffer position and the lowest order DES output byte. Store this value in a second work buffer.

11) Obtain a new DES initial value by moving the DES output to the input, shifted one byte to the left, and shifting the work buffer value into the lowest order position.

12) Repeat steps 9 through 11 using successive work buffer index values until all of the data is transformed.

13) Replace the transformed index values by their corresponding character equivalents and store them in the open cipher text positions.

Decryption is accomplished by performing the inverse of each transformation in the reverse order.

#### 4. Implementation Issues and Usage Constraints

Perhaps the most important  *caveat*  for anyone who wishes to implement our proposed encryption scheme is to guard against possible misinterpretation of encrypted data.

Scrambled text fields such as names and addresses are not likely to be mistaken for real information, but numeric fields may contain quite plausible values. A legitimate user who, through some administrative oversight, is erroneously presented with encrypted data may not recognize it as such and make bad decisions as a result. One approach to this shortcoming may be to include code in the query tool to fill in encrypted fields with a default value whenever the user has not been authenticated. Another approach may be simply to restrict the application of the technique to text fields. A revenue field may be quite useless without the corresponding product data or sales region information.

Another restriction on the use of this technique is that decryption must be performed before aggregate functions, such as minimum, maximum, sum, and average, are applied. This is not a serious inconvenience in the data warehousing environment because precompiled summary tables are usually available.

One must also bear in mind that this encryption scheme is  *consistent*  in that the same plaintext always results in the same cipher text. This has both positive and negative implications. On the positive side, the consistency of the encrypted

data allows for relational joins and *blind keys* (described later). On the other hand, consistent encryption exposes the data to the possibility of a statistical attack. If an attacker knows the relative frequency of specific data items, such as medical tests, he can deduce the corresponding encrypted values. This kind of attack can be stymied by using a value from another field (the table's primary key, for example) to modify the encryption key. This would, of course, preclude the use of this data in relational join predicates.

## 5. Co-existence with Other Security Controls

We do not propose data type-preserving encryption as the ultimate solution to all data warehouse security concerns. It is presented, rather, as one of several mechanisms to be employed in a more comprehensive security strategy. Specifically, we see our technique as a *containment* device which limits potential damage in the event of a successful bypass of other security controls. In general, there are at least five categories of security controls:

**Prevention-** Preventative measures include anything which can be done to prevent an attack or to keep it from succeeding. This includes strengthening vulnerabilities and providing disincentives to the attacker.

**Detection-** Detective measures include anything which alerts the support staff to the fact that an attack is in progress or has been recently attempted.

**Containment-** Containment measures include anything which can serve to limit the damage of a successful attack.

**Recovery-** Recovery measures include anything which is done to restore normal operation and user access after an unscheduled interruption.

**Investigation-** Investigative measures include anything which is done to identify a malefactor and collect evidence which will be used in a disciplinary process or criminal prosecution.

A good data warehouse security plan will include multiple countermeasures for each identified threat ideally, at least one from each of these categories.

## 6. Applications to Other Areas

In addition to data warehouse security, there may be several other areas in which this technique may prove useful, such as providing an additional check on data integrity.

By adding a check character to the beginning of each plaintext field any alteration would be immediately obvious during the decryption process. The decryption routine could be

modified to perform this check and return an error code is tampering is suspected.

Another possible application is in the use of *blind keys*. In certain situations, one needs to know that two quantities are equal without actually knowing the quantities themselves.

One may wish to match bank account numbers from multiple sources, for example, in credit check applications but not use the actual numbers themselves because of the potential for fraudulent activity.

It may also be possible to control access to commercially available data through the use of this technique. A master database could be distributed to subscribers with individually licensed components encrypted using different keys. Access to the individual components could be made available by distributing keys following payment of the proper license fees.

**Acknowledgement:** I really thankful our Chairman Madam Smt.R.Usharani, Director Sri R.RajiReddy and my kids Mr.M.Crazy and Kum M.Lucky who encouraged lot directly or indirectly to present this research paper

## References

- [1] H. Akaike. On entropy maximization principle. *Applications of Statistics*, pages 27{41, 1977.
- [2] M. R. Anderberg. *Cluster Analysis for Application*. Academic Press, 1973.
- [3] P. S. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Proc. 4th International Conf. on Knowledge Discovery and Data Mining (KDD-98)*. AAAI Press, August 1998.
- [4] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Society*, 64:1183{1210, 1969.
- [5] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Tras. Math. Software*, 3(3):209{226, 1977.
- [6] C. L. Giles, K. D. Bollacker, and S. Lawrence. CiteSeer: An automatic citation indexing system. In *Digital Libraries 98 { Third ACM Conference on Digital Libraries*, 1998.
- [7] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD*, May 1995.
- [8] H. Hirsh. Integrating multiple sources of information in text classification using whirl. In *Snowbird Learning Conference*, April 2000.
- [9] J. Hylton. Identifying and merging related bibliographic records. MIT LCS Masters Thesis, 1996.
- [10] B. Kilss and W. Alvey, editors. *Record Linkage Techniques/1985*, 1985. Statistics of Income Division, Internal Revenue Service Publication 1299-2-96. Available from <http://www.fcsm.gov/>.
- [11] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet

portals with machine learning. *Information Retrieval*, 2000. To appear.

- [12] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. <http://www.cs.cmu.edu/~mccallum/bow>, 1996.
- [13] A. Monge and C. Elkan. The edit-matching problem: algorithm and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, August 1996.
- [14] A. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *The proceedings of the SIGMOD 1997 workshop on data mining and knowledge discovery*, May 1997.
- [15] A. Moore. Very fast EM-based mixture model clustering using multiresolution kd-trees. In *Advances in Neural Information Processing Systems 11*, 1999.
- [16] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954{959, 1959.
- [17] S. Omohundro. Five balltree construction algorithms. Technical report 89-063, International Computer Science Institute, Berkeley, California, 1989.
- [18] K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11):2210{2239, 1998.
- [19] G. Salton and C. Buckley. Term weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513{523, 1988.
- [20] M. Sankaran, S. Suresh, M. Wong, and D. Nesamoney. Method for incremental aggregation of dynamically increasing database data sets. *U.S. Patent 5,794,246*, 1998.
- [21] D. Sankoff and J. B. Kruskal. *Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, 1983.
- [22] J. W. Tukey and J. O. Pedersen. Method and apparatus for information access employing overlapping clusters. *U.S. Patent 5,787,422*, 1998.
- [23] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103{114, 1996.