INDEXING OF XML-BASED ENCRYPTED MEDICAL DOCUMENTS IN WORM STORAGE

by

Naim Aksu

B. S., Computer Engineering, Boğaziçi University, 2003

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2006

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Prof. Taflan Gündem for his guidance and advice and also for the material support needed for this research.

I would like to thank Assist. Prof. Pınar Yolum Birbil and Assist. Prof. Ata Akın for their participations in my thesis committee.

I would like to thank Boğaziçi University Research Fund for supporting this thesis under grant number 04A108.

Special thanks to my wife Ozanay Aksu and to my family, for their endless love and support all throughout my study.

ABSTRACT

INDEXING OF XML-BASED ENCRYPTED MEDICAL DOCUMENTS IN WORM STORAGE

Since the medical records are increasingly stored in electronic forms, especially in XML based documents, the storage devices for these records must preserve their trustworthiness. The regularity requirements rely on storing the critical data in Write-Once-Read-Many (WORM) storage devices to prevent them from easy modification. The XML documents stored in the WORM storage may also be encrypted in order to protect the sensitive data, and satisfy the security requirements. Efficient access of the large volume of records requires the use of direct access mechanisms such as indexes. Relying on indexes for accessing records could, however, provide a means for effectively altering or deleting records, even those stored in WORM storage.

In this study, we propose a novel indexing structure and an encryption schema for encrypted medical XML documents stored in WORM storage structures. The proposed indexing method expedites projection, selection and join operations on encrypted medical XML records stored in WORM storages, and uses Generalized Hash Tree (GHT) data structure. Also, medical XML documents are stored with a novel encryption schema which combines traditional encryption techniques with order preserving encryption schema (OPES). In the literature so far, there are only some studies on indexing techniques for some ordinary documents such as electronic mail, financial statements, quality assurance documents which all stored in WORM structures, and these indexing structures only manage simple insert and search algorithms. Finally, we demonstrate that the proposed system gives satisfactory performance.

ÖZET

WORM DEPOLAMA AYGITLARINDA SAKLANAN XML TABANLI VE ŞİFRELİ TIBBİ DÖKÜMANLARIN İNDEKSLENMESİ

Tıbbi dökümanların elektronik ortamda özellikle XML tabanlı dökümanlar şeklinde saklanma oranının artması, bu amaçla kullanılan depolama aygıtlarının kayıt güvenirliliğini sağlamasını zorunlu kılmaktadır. Düzenleyici kurallar bu kayıtların 'bir kez yaz çok kez oku' (WORM) depolama aygıtlarında saklanmasını öngörmektedir. Ayrıca, hassas bilginin korunabilmesi ve güvenlik gereksinimlerinin karşılanabilmesi için WORM depolama aygıtlarında saklanan dökümanlar şifrelenebilir. Bununla birlikte, kayıt sayısının fazlalığı bu kayıtlara hızlı bir şekilde ulaşabilmek için indeks kullanımını gerekli kılmaktadır. Fakat, kayıtlara indeks yoluyla ulaşılması bu kayıtların başkaları tarafından kolaylıkla değiştirilmesi ve silinmesi için uygun bir ortam yaratmaktadır.

Biz bu çalışmada, WORM depolama aygıtlarında şifreli şekilde saklanan XML tabanlı tıbbi dökümanlar için yeni bir indeksleme yapısı ve şifreleme yöntemi önermekteyiz. Önerilen indeksleme metodu, WORM depolama aygıtlarında şifreli şekilde saklanan XML tabanlı tıbbi dökümanlar üzerinde izdüşüm, seçme ve birleştirme işlemleri yapabilen ve GHT veri yapısını kullanan bir indeksleme tekniğidir. Ayrıca, XML tabanlı tıbbi dökümanlar, geleneksel şifreleme yöntemlerini sıra korumalı şifreleme yöntemiyle (OPES) birleştiren yeni bir şifreleme algoritması kullanılarak saklanmaktadır. Literatürde şu ana kadar sadece elektronik postalar, mali raporlar gibi sıradan dökümanlar için indeksleme yöntemleri ile ilgili çalışmalar bulunmaktadır ve bu indeksleme yapıları sadece temel kayıt ekleme ve kayıt arama algoritmalarını yapabilmektedir. Son olarak, önerdiğimiz sistemin performans açısından tatmin edici sonuçlar verdiğini kanıtladık.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
ANSI	American National Standards Institute
DES	Data Encryption Standard
GHT	Generalized hash tree
HIPAA	Health Insurance Portability and Accountability Act
HL7	Health Level Seven
HMO	Health maintenance organization
OPES	Order preserving encryption schema
SDO	Standards developing organization
SEC	Securities and Exchange Commission
SGML	Standard Generalized Markup Language
XML	Extensible Markup Language
WORM	Write once read many

1. INTRODUCTION

Medical records contain sensitive information about medical procedures such as diagnosis of illnesses, medical treatments of patients, the medicines used, the treatment results of patients and so on [5, 9]. These sensitive records are increasingly stored in electronic form, which make them relatively easy to delete and modify without leaving a trace. This brings some important treats on the sensitive data. Thus, in some countries, medical records are subject to regulations that specify how they should be managed [4, 5]. These critical medical records have to be protected from any physical modification during their storage. Modification of records could result from user errors, such as issuing the wrong commands and replacing the wrong disks, and from software bugs [1].

There has been a growing importance to introduce Write-Once-Read-Many (WORM) storage devices to enable the effective preservation of records [11, 12]. The basic property of WORM storage is not to allow updates on the data inserted once. In this thesis study, we assume that the underlying storage supports reads from and writes to a random location, and ensures that any data that has been written cannot be overwritten, at least for a specified period. This assumption is in line with the current industry trend of providing WORM storage using magnetic disks [11, 12].

Due to the large volume of records and increasingly stringent query response time [10], some form of direct access mechanism such as an index must be available in order to access the records [3]. But if the index is not properly designed, the medical records stored in WORM storage can in effect be hidden or altered easily. The main requirement for this type of index is to ensure that once a record is preserved in WORM storage, the record will be quickly accessible in an unaltered form through the index. However, if the records are accessed through an index, even if they are stored in WORM storage, they will still be vulnerable to logical modification if the index can be suitably manipulated.

Possible attacks from an adversary to the indexed medical data stored in WORM storage can be explained by the following example. An adversary, possibly a system administrator, could write carefully crafted data to the WORM storage to subvert the index

and logically hide or modify selected records. He could fake an adjustment to the index structure such as tree rebalance, which would enable him to omit crucial records from the new structure. He could also prevent timely access to crucial records by causing the index to degenerate by inserting carefully chosen spurious records [1].

Today, XML (Extensible Markup Language) [6] is currently one of the most relevant standardization efforts in the area of document representation through markup languages and is rapidly becoming a standard for data representation and exchange over the Web. Especially, XML is becoming a common platform in medical records since it becomes essential to integrate separated medical information and allow them to be exchanged and retrieved through internet [16]. For example, Health Level Seven (HL7) is one of the several American National Standards Institute (ANSI) accredited Standards Developing Organizations (SDO's) operating in the healthcare arena. It has been actively working with XML technology since the formation of the SGML/XML Special Interest Group in September of 1996.

Another important concept for medical records is the privacy of critical medical data. In many countries, there are regulations for the privacy of medical records [15]. With the rapid advances in the computerization of medical data, the question of the protection of medical records privacy has begun to arise. Storing a large amount of sensitive information in databases could open the door to invasion of privacy. Storing the medical XML documents in WORM structures is not adequate for the security and privacy of the documents [5, 15]. Encryption is a commonly utilized method for the security of sensitive medical XML documents. Thus, the sensitive medical data must be encrypted to satisfy the security requirements.

In the thesis, we have designed a fossilized index structure and a hybrid encryption schema for the XML-based encrypted medical documents stored in WORM storages. The proposed index structure expedites the projection, selection and join operations for the medical XML data. The encryption schema combines the conventional encryption methods with the order preserving encryption schema (OPES) in [2].

Here, the term fossilized index represents an index structure which ensures that once a record is preserved in WORM storage, it is accessible in an unaltered form and in a timely fashion. In other words, the index structure is also stored in the WORM structure in an unaltered form like the original medical XML data. The index structure we have proposed handles not only simple insert and search operations, but also complex queries such as joins, projections and selections. Our index structure uses the generalized hash tree index structure in [1] to be able to query on the medical XML documents stored in the WORM structure.

All the element, attribute and node values are numbered using a preorder traversing algorithm. Then all the numbered values in the given XML document are inserted into double layer GHT by using the new extended insertion algorithm of GHT. The first layer of the GHT contains the path values of the XML documents, while the second layer of the GHT contains the leaf node values of the medical XML documents with an offset for a faster locating in a single XML document.

Conventional encryption schema is combined with the order preserving encryption schema in our proposed encryption algorithm [2]. Conventional encryption methods are used to encrypt the sensitive medical data and order preserving encryption schema is used to encrypt the number values of XML tree nodes in the proposed model. In our design, both the structure and content of the XML document is encrypted. Also, partial or full encryption of the given medical XML document is supported with a specified tag attribute in the XML file. Partial encryption schema prevents the system from encrypting all the data in the medical document. This operation results in performance gain by not encrypting the data which does not need encryption. Thus, we can decrease the time loss resulted from encryption and decryption operations of the medical XML documents.

Other important key issue in our encryption schema is that we use an order preserving encryption schema for encrypting the corresponding numeric values of tag names. This schema allows any comparison operation to be directly applied on encrypted data [2]. Also, in the encryption scheme, new values can be added without requiring changes in the encryption of other values. Thus, whole decryption is not needed for the medical XML document, and searching the numeric value of corresponding tag names in

the encrypted XML document is fastened. Thus, in our complete design, the projection, selection and join operations on specially encrypted medical XML documents are fastened.

In the literature, there is not any recent work directly about the indexing of XML-based encrypted medical documents stored in WORM storages. But, there is some research on efficient indexing of records stored in WORM structures. The most recent study on this type of indexing structure is a fossilized index that uses a generalized hash tree structure to handle insertion and search operations [1]. But this study only concern about ordinary documents such as electronic mail, financial statements, and quality assurance documents and performs basic key insertion and search facilities in these documents. Also, there are some order preserving encryption schemas and XML encryption methods in the literature. But, there is not any study which uses OPES for encryption of XML-based documents.

The rest of the thesis is organized as follows. We give an overview about the preliminary concepts for a better understanding of the proposed algorithms and structures in Section 2; also the related work about our proposed system is given in this section. We present the proposed index structure for the medical XML databases and the encryption schema for medical documents, and extend the index structure for encrypted medical XML documents in Section 3. In Section 4, we deal with the performance study of the proposed model and also simulation results are reported. Section 5 concludes the paper.

2. PRELIMINARY CONCEPTS AND RELATED WORK

In this section of the thesis, some preliminary concepts and the related work is explained for better understanding of the proposed algorithms and structures. First, trustworthy record keeping concept in medical databases will be discussed. Then, the WORM storage structures will be explained with its properties and the need for WORM storage in medical documents will be discussed. After that, the fossilized indexing technique will be explained, and the details about fossilized indexing will be presented with its properties and basic requirements. Also, some recent work for fossilized indexing in the literature will be given.

Generalized Hash Tree structure will be described with its advantages and disadvantages, and also the basic insert and search algorithms of GHT will be explained. After that, we discuss why we need an extension for this structure. Then, security issues in medical databases will be explained since a novel encryption schema is our second contribution in this study. The basic information about conventional encryption schema used in the proposed algorithm will be presented. Then, essential background about order preserving encryption schema will be explained with its key points which constructs the basic of our encryption schema. Also, recent work for order preserving encryption will be given.

2.1. Trustworthy Record Keeping in Medical Documents

The fundamental purpose of record keeping is to establish solid proof and accurate details of events that have occurred. Trustworthy records are, therefore, those that can be relied upon to achieve this purpose [1]. Especially, in medical databases, trustworthy record keeping is an important concept since very important data about public health are kept in these databases. Medical records contain sensitive information about medical procedures such as diagnosis of illnesses, medical treatments of patients, the medicines used, the treatment results of patients and so on [5, 9].

The storage structure is supposed to ensure that in an enquiry, all of the relevant records can be quickly located and retrieved in an untampered form. In other words, the records have to be protected from any physical modification during their storage. The suitable storage structure that satisfies the mentioned requirement is WORM storage structures. The basic property of WORM storage is not to allow updates on the data inserted once. In the thesis, we assume that the underlying storage supports reads from and writes to a random location, and ensures that any data that has been written cannot be overwritten, at least for a specified period. This assumption is in line with the current industry trend of providing WORM storage using magnetic disks [11, 12].

Trustworthy record keeping is an important concept in medical documents. Thus, there are regulatory requirements in many countries about medical data retention and storing the critical medical data in unalterable storage devices to prevent them from easy modification. With the increase of regulations governing the retention of data, companies and hospitals are challenged to find an efficient way to comply with requirements that files be stored for long periods of time and remain unchanged to protect data integrity. Such regulations include HIPAA, Sarbanes-Oxley, SEC 17a-3, SEC 17a-4, etc. HIPAA is the one which acts on the healthcare area. Healthcare, financial services, biotech, and government agencies are currently under pressure of these industry-specific regulations [28].

HIPAA is the United States Health Insurance Portability and Accountability Act founded in 1996. HIPAA includes an administrative simplification section which deals with the standardization of healthcare-related information systems. In the information technology industries, this section is what most people mean when they refer to HIPAA. HIPAA establishes mandatory regulations that require extensive changes to the way that health providers conduct business [4].

HIPAA establishes standardized mechanisms for medical data retention, electronic data interchange, security, and confidentiality of all healthcare-related data. The Act mandates standardized formats for all patient health data, medical data retention for some period of time different for each type of medical documents, unique identifiers (ID numbers) for each healthcare entity, and security mechanisms to ensure confidentiality and

data integrity for any information that identifies an individual. The privacy regulations are in effect now, and security regulation enforcement also began in 2005 [28].

The complete data retention requirements are:

- Medical records: Children, birth to 21 years of age; and adults, five years, continuing until two years after death;
- Records of information disclosures: six years;
- Compliance standards, implementations, policies, procedures: six years [28].

Thus, the regularity requirements mandate a compliant electronic storage medium which prevents easy modification, and support integrity protection and accessibility. However, in efforts to secure the data to be retained, WORM functionality is a strong ally in the effort to maintain the integrity of that data [29].

2.2. Indexing in Rewritable Storage

Direct access mechanisms such as indexes are needed to be maintained in order to ensure that all of the records relevant to an enquiry can be discovered and retrieved in a timely fashion. Thus, if an index is used and can be suitably manipulated, they will still be vulnerable to logical modification. As mentioned in the above section, an adversary, possibly a system administrator, could write carefully crafted data to the WORM storage to subvert the index and logically hide or modify selected records.

He could fake an adjustment to the index structure such as tree rebalance, which would enable him to omit crucial records from the new structure. He could also prevent timely access to crucial records by causing the index to degenerate by inserting carefully chosen spurious records. Thus, the index itself must be maintained in the WORM structure in order to prevent it from altering or hiding the contents. Figure 2.1 and Figure 2.2 show how the records are vulnerable to logical modification if an index is used [1].

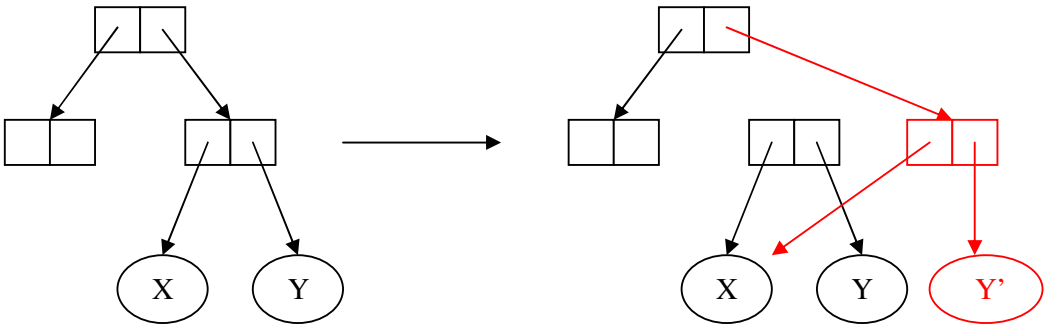


Figure 2.1. Indexes in rewritable storage allow altering the content

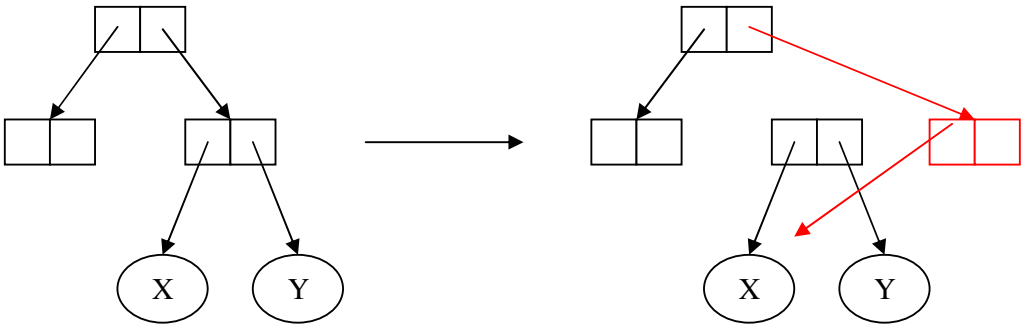


Figure 2.2. Indexes in rewritable storage allow hiding the content

2.3. Fossilized Indexing

In this section, the key properties of an index to avoid these types of risks will be explained. The term *fossilized index* means that once a record is stored in WORM storage, it is accessible in an unaltered form and in a timely fashion. The key properties for such an index are explained in [1] in a detailed manner. Thus, we give an overview about this type of index structure in this section, and we consider these properties for the medical records.

The first property of a fossilized index is that once a record is committed, both the index entry for that record and the path to that entry must be immutable. Once the insertion of a medical record into the index has been committed to WORM storage, the record is guaranteed to be accessible through that index unless the WORM storage is compromised. Secondly, the index must support incremental growth. Especially indexes on medical data must scale to extremely large collections of medical records. Also, the space overhead of the index must be acceptable. Although storage has become relatively inexpensive with the

rapid improvement in disk areal density, storage efficiency is still a major consideration in medical databases.

2.4. Recent Work on Fossilized Indexing

There has been much previous work on indexing structures for WORM storages. But most of them do not meet the requirements of a fossilized index. For example, some models have stored the index on rewritable disks while the actual data has been stored in WORM structures [19]. Regardless of the type of index used, an adversary can easily modify the index to cause records to be effectively hidden or altered in this indexing method. Another approach was a balanced tree stored in WORM storage. It works by making a new copy of the tree each time an insertion occurs. But, the time and space overhead of this approach is prohibitive. Thus, persistent search trees which copy only the nodes involved in rebalancing the tree have been used for a straightforward optimization [20, 21]. Other approaches for indexes stored in WORM storages have been write-once B-tree [22], multi-version B-tree [14] and the append-only trie [23]. But, all of these approaches are again vulnerable to tampering, since it is infeasible to look up each version of the tree to defend against tampering even if every alteration of the nodes is preserved in WORM storage.

2.5. Generalized Hash Tree

The index structure that maintains the properties of a fossilized index is GHT structure [1]. Since our proposed indexing model for encrypted medical XML documents uses GHT structure, we explain the basic properties of GHT in this section. Also, we give its insertion and search algorithms for better understanding of our proposed algorithms in the next sections.

The basic properties of GHT data structure satisfy all the properties of a fossilized index. As mentioned in the previous sections, any approach that requires the rebalancing of a tree allows an adversary to create new paths to records. Also, trees that grow from the leaves up to the root have the same disadvantage, because an adversary could modify records at will by exploiting the provision for creating new versions of the tree nodes.

Lastly, any method that permits index entries to be relocated is not trustworthy because it opens the door for an adversary to create new versions of any entry. But GHT data structure is a tree that grows from the root down to the leaves without relocating committed entries and that is balanced without requiring dynamic adjustments to its structure. Also, it has an efficient dynamic hashing schema that does not require rehashing.

The basic properties of insert and search algorithms of GHT data structure can be presented as follows. Attempts to insert or lookup a record start at the root node of the tree. If it is unsuccessful, the process is repeated at one or more of its children subtrees. When a record cannot be inserted into any of the existing nodes, a new node is created and added to the tree as a leaf. At each level, the possible locations for inserting the record are determined by a hash of the record key. Consequently, the possible locations of a record in the tree are fixed and determined solely by that record. Moreover, inserted records are never rehashed or relocated. The data structure is called as *generalized* because it represents a family of hash trees. By using different parameters and hash functions, hash trees can have different characteristics. In the thesis, we focus on *Thin Tree* for our indexing method for encrypted medical XML databases.

For a better understanding of our proposed indexing method, GHT data structure terms must be known well. In basic GHT, a *record* is represented by a key and a pointer to the actual data. But, since our proposed indexing model is designed for XML documents, we will need some extra fields in a record. These will be explained in a detailed manner in next sections. A *bucket* is an entry in a tree node to store a record. A *tree node* consists of buckets and is the basic allocation unit of a tree. These are the basic units of GHT data structure. Also, there are some rules of the units. The size of a tree node may vary with its level in the tree.

Let $M = \{m_0, m_1, m_2, \ldots\}$ where m_i is the size of a tree node at level i. A *growth factor* k_i denotes that the tree may have k_i times as many buckets at level (i+1) as at level i. The growth factor may vary for each level. Let $K = \{k_0, k_1, k_2, \ldots\}$ where k_i is the growth factor for level i. Let $H = \{h_0, h_1, h_2, \ldots\}$ denote a set of hash functions where h_i is the hash function for level i.

There are three criteria to consider when choosing the hash functions. First, they should be independent to reduce the chances that records colliding at one level of the tree will also collide at the next. Second, they should be efficient to compute. Third, they should be insensitive to the size of the target range. In [1], the universal hash function is used for GHT. It is defined as h(x) = ((ax + b) mod p) mod r.

Here, p is a prime chosen such that all the possible keys are less than p, r is the size of the target range, a is element of {1, 2,…, p-1}, b is element of {0, 1, 2,…, p-1}. The hash function for each level is selected randomly at the time the level is created by picking random values for a and b. Using randomly selected hash functions prevents the GHT from degenerating largely into a list on specific inputs and allows it to achieve provably good performance on average [1].

After giving enough information about the terms and functions of GHT, we can give the TREE_INSERT (see Figure 2.3) and TREE_SEARCH (see Figure 2.4) algorithms of the data structure. Here, *t* is a pointer to the root of a tree, and *x* is a record to be inserted into GHT and also *key* is a search key used in search algorithm.

```
1: i = 0; p = t.root; index = h0(key)
2: loop
3:     if node p does not exist then
4:         p = allocate a tree node
5:         p[index] = x
6:         return SUCCESS
7:     end if
8:     if p[index] is empty then
9:         p[index] = x
10:        return SUCCESS
11:    end if
12:    if p[index].key = x.key then
13:        return FAILURE
14:    end if
15:    i = i+1 {Go to the next tree level}
```

```
16:     j = GetNode(i, h_i(key))
17:     p = p.child[j]
18:     index = GetIndex(i, h_i(key))
19: end loop
```

Figure 2.3. TREE-INSERT (t, x) algorithm of GHT

```
1: i = 0; p = t.root; index = h_0(key)
2: loop
3:     if node p does not exist then
4:         return NULL
5:     end if
6:     if p[index] is empty then
7:         return NULL
8:     end if
9:     if p[index].key = key then
10:        return p[index]
11:    end if
12:    i = i+1 {Go to the next tree level}
13:    j = GetNode(i, h_i(key))
14:    p = p.child[j]
15:    index = GetIndex(i, h_i(key))
16: end loop
```

Figure 2.4. TREE-SEARCH (t, key) algorithm of GHT

Here, there are some functions to be explained for a better understanding of the algorithms above, which are *GetHashTableSize(i)*, *GetNode(i, j)*, and *GetIndex(i, j)*. *GetHashTableSize(i)* is equal to *m (if i = 0); m * k (if i ≠ 0)*. *GetNode(i, j)* is equal to *j div m*, and *GetIndex(i, j)* is equal to *j mod m*.

After giving all the terms and algorithms needed for better understanding of the GHT data structure, we now give an insertion example for GHT in Figure 2.5.

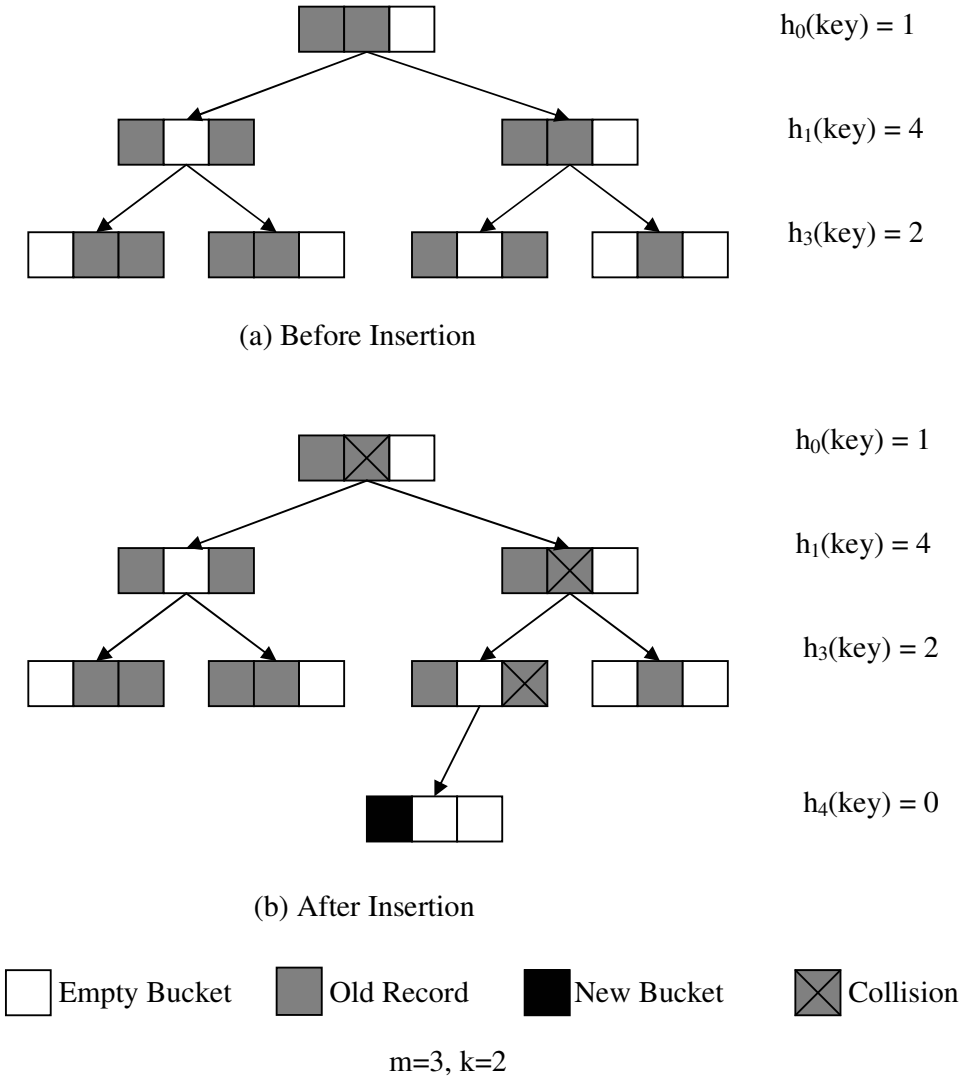


Figure 2.5. Inserting records into GHT

Figure 2.5 illustrates the insertion of a record into a GHT tree where m = 3 and k = 2. H stands for a predefined hash function. Here, we assume that (h_0, h_1, h_2, h_3) (key) = (1, 4, 2, 0) and use (level, node, index) to denote a bucket in a tree node. We first try to insert the record into the root node with fist level hash function value h_0(key) = 1. However, the target bucket (0,0,1) is not empty. Thus, we try again in the hash table at the next level, which is formed by the two children nodes of the root. h_1(key) = 4 indicates that the target bucket is (1, 1, 1) which is not empty either. Since the collision happens in node 1, its two children nodes form the next level hash table. But, the next attempt collides in the bucket (2,0,2). The fourth attempt succeeds since the tree node containing the target bucket does

not exist. We allocate a new tree node and insert the record into the bucket (3,0,0). Intuitively, if the hash functions are uniform, the tree grows from the root down in a balanced fashion.

The procedure TREE_SEARCH lists the steps to retrieve a record given its key. The procedure returns the record if it exists in the tree and NULL otherwise. When a target bucket is full, we test if its key matches the search key. If they match, then the record is found; otherwise we follow the same process as in insertion to probe the next level of the tree. When a target bucket is empty or the target tree node has not been allocated, the search fails.

The example given above summarizes the basic insertion and search algorithms of the thin tree type of GHT data structure. Since we use the thin tree type of the GHT data structure in our proposed fossilized indexing method for encrypted medical XML documents, this example is a starting point for the thesis. Now, in the next section, some conventional encryption techniques will be summarized, and then the order preserving encryption schema (OPES) in [2] will be explained for a better understanding of our proposed encryption schema to store medical XML documents in WORM storage structures.

2.6. Importance of Encryption in Medical Documents

We now give some conventional encryption information in this section. Also, we discuss why we need encryption in medical documents stored in XML format in WORM storages. As mentioned in the introduction part, the second contribution of our thesis study is a novel encryption schema to store the medical XML documents in WORM storage structures.

The privacy of medical records is an important issue. For example, a research in United States showed that during a typical stay in a hospital, a patient's record could be viewed by as many as 77 different people. Having a patient's record on computer would most likely increase that number dramatically. Some hospitals have developed procedures to protect the confidentiality of their electronic information [9]. Hospitals are not the only

ones archiving medical records on patients. Data banks of such organizations as health maintenance organizations (HMO's) and drug companies are also gathering information and storing them in a computer format. Thus, the secrecy and privacy of medical data gains an extra concern. The Health Insurance Portability and Accountability Act (HIPAA) establishes a set of national standards for the protection of individually identifiable heath information at certain categories of institutions. Non-compliance with these rules result in civil and criminal penalties [25, 26].

Database systems typically offer access control mechanisms to restrict access to sensitive data. This mechanism protects the privacy of sensitive information provided data is accessed using the intended database system interfaces. However, access control, while important and necessary, is often insufficient. Attacks upon computer systems have shown that information can be compromised if an unauthorized user simply gains access to the raw database files, bypassing the database access control mechanism altogether [2]. This means that encryption methods must be used to provide an extra level of security for the sensitive data. Thus, we have agreed that an encryption schema must be applied on the medical XML data that are concerned with our thesis.

The encryption schema we propose is basically combines the traditional encryption with the order preserving encryption schema. Here, the term *conventional cryptography* contains the known symmetric-key and public key algorithms. Since we use DES (Data Encryption Standard) encryption method in the proposed schema in the thesis, we give some essential information about symmetric-key algorithms. Symmetric-key algorithms use the same basic ideas as traditional cryptography such as transportation and substitution, but its emphasis is different. Symmetric-key algorithms use the same key as encryption and decryption. The key length varies according to the algorithm used. For example, in DES, a 56 bit key is used while AES (Advanced Encryption Standard) uses 128 or 256 bit length key. Also, the extended version of DES, named as *triple DES*, uses 112 bit key. DES algorithm, adopted by the U.S. government in 1977, is a block cipher that transforms 64-bit data blocks under a 56-bit secret key, by means of permutation and substitution. The DES algorithm is widely used and is still considered reasonably secure. In our proposed encryption method, medical XML documents can be partially encrypted by specifying an

encryption flag as an attribute for the data to be encrypted. The specified parts will be encrypted with DES encryption algorithm using 56-bit length key.

2.7. Recent Work on Order Preserving Encryption

There have been some proposed encryption schemas for achieving order preserving of numeric data, and querying on encrypted data in the literature so far. But, most of them have some leakages to achieve the necessities for a complete order preserving schema. For example, a simple scheme has been proposed in [17] that computes the encrypted value c of integer p as $c = \Sigma^{p}_{j=0} R_j$, where R_j is the j^{th} value generated by a secure pseudo-random number generator R. Unfortunately, the cost of making p calls to R for encrypting or decrypting c is prohibitive for large values of p. Also, in another example, in [7], a sequence of strictly increasing polynomial functions is used for encrypting integer values while preserving their order. But, this encryption method does not take the input distribution into account. Since the shape of the distribution of encrypted values depends on the shape of the input distribution, this scheme may reveal information about the input distribution, which can be exploited.

Another approach for order preserving encryption was bucketing. In this approach in [8], tuples are encrypted using conventional encryption, but an additional bucket id is created for each attribute value. The constants appearing in a query are replaced by their corresponding bucket ids. But the leakage of this approach was that, the result of a query will contain false hits that must be removed in a post-processing step after decrypting the tuples returned by the query. All the summation of random numbers, polynomial functions and bucketing approaches have some leakages for order preserving requirements with respect to performance or reliability.

2.8. Order Preserving Encryption Schema (OPES)

We now explain the order preserving encryption schema in this section. In our proposed encryption schema, we use OPES for partial encryption and decryption of medical XML documents. Partial encryption and decryption prevents the time losses which results from unnecessarily encrypted and decrypted parts of the documents.

As mentioned in the previous sections, encryption is a well established technology for protecting sensitive data [14, 24]. But, the integration of existing encryption techniques with database systems causes undesirable performance degradation. For example, if a column of a table containing sensitive information is encrypted, and is used in a query predicate with a comparison operator, an entire table scan would be needed to evaluate the query. The reason is that the current encryption techniques do not preserve order and therefore database indices can no longer be used. OPES solves this problem by preserving the order in the encryption phase [2]. Thus, this allows comparison operations to be directly applied on encrypted data. This key property of OPES helps us to decrypt the whole medical XML documents when a query comes.

Some other properties of OPES makes our proposed schema more robust. The results of query processing over data encrypted using OPES are exact. They neither contain any false positives nor miss any answer tuple. This feature of OPES sharply differentiates it from schemes such as [13] that produce a superset of answer, necessitating filtering of extraneous tuples in a rather expensive and complex post-processing step. Also, OPES handles new insertions gracefully. A new value can be inserted in a column without requiring changes in the encryption of other values. This property also helps us in our proposed encryption schema.

The key idea of OPES is to take as input a user-provided target distribution and transform the plaintext values in such a way that the transformation preserves the order while the transformed values follow the target distribution [2]. The important point here is that the same target distribution is obtained even if the input distributions are different from each other. This property has an advantage over the other order preserving schemas discussed in the previous section.

OPES algorithm can be understand better with the following encryption schema. Generate |P| unique values from a user-specified target distribution and sort them into a table T. The encrypted value c_i of p_i is then given by $c_i = T[i]$. That is, the i^{th} plaintext value in the sorted list of |P| plaintext values is encrypted into the i^{th} value in the sorted list of |P| values obtained from the target distribution. The decryption of c_i requires a lookup into a reverse map. Here T is the encryption key that must be kept secret. This simple scheme,

while instructive, has the following shortcomings for it to be used for encrypting large databases: The size of encryption key is twice as large as the number of unique values in the database. Also, insertions are problematic. When adding a new value p, where $p_i < p < p_i+1$, we will need to re-encrypt all p_j, $j > i$. But, OPES has been designed such that the result of encryption is statistically indistinguishable from the one obtained using the above scheme, thereby providing the same level of security, while removing its shortcomings. OPES works in three stages as explained in [2]:

1. *Model:* The input and target distributions are modeled as piece-wise linear splines.

2. *Flatten:* The plaintext database P is transformed into a "flat" database F such that the values in F are uniformly distributed.

3. *Transform:* The flat database F is transformed into the cipher database C such that the values in C are distributed according to the target distribution.

One important point here to be noted is that $p_i < p_j \rightarrow f_i < f_j \rightarrow c_i < c_j$.

3. PROPOSED SYSTEM

The proposed model will be discussed in this section. Some examples will be given for each case for a better understanding of the algorithms and pseudo codes presented for the model.

First, a general layout of the proposed model will be given. Then, inserting a new medical XML document into the WORM storage will be presented with its phases. Phases include global path indexing, local preorder traversing and partial DES and OPES encryption. The next part of the model is inserting the corresponding number values of XML paths into the 1st level GHT, and inserting the DES encrypted attribute and node values into the 2nd level GHT. First and second layer of GHT structure is an extension to the basic GHT to manage the XML queries.

We then present search operation phases for a better understanding how our proposed fossilized index structure expedites projection, selection and join operations on the encrypted medical XML documents. Then, we give how our model manages the projection, selection and join operations on encrypted XML documents, and lastly, we discuss the contribution of this study to conclude the section.

3.1. General Layout of the Proposed System

In our proposed model, there are several stages to insert the document into the WORM storage and to send a query to the stored document. Thus, a general layout of the model will be given in this section. Before accessing the fossilized index structure to response a query, the insertion of the document into the WORM storage must be specified for a compete model. Thus, a summary of the insertion of a medical XML document into the WORM storage will be presented. The Figure 3.1 gives a general layout of the proposed complete model.

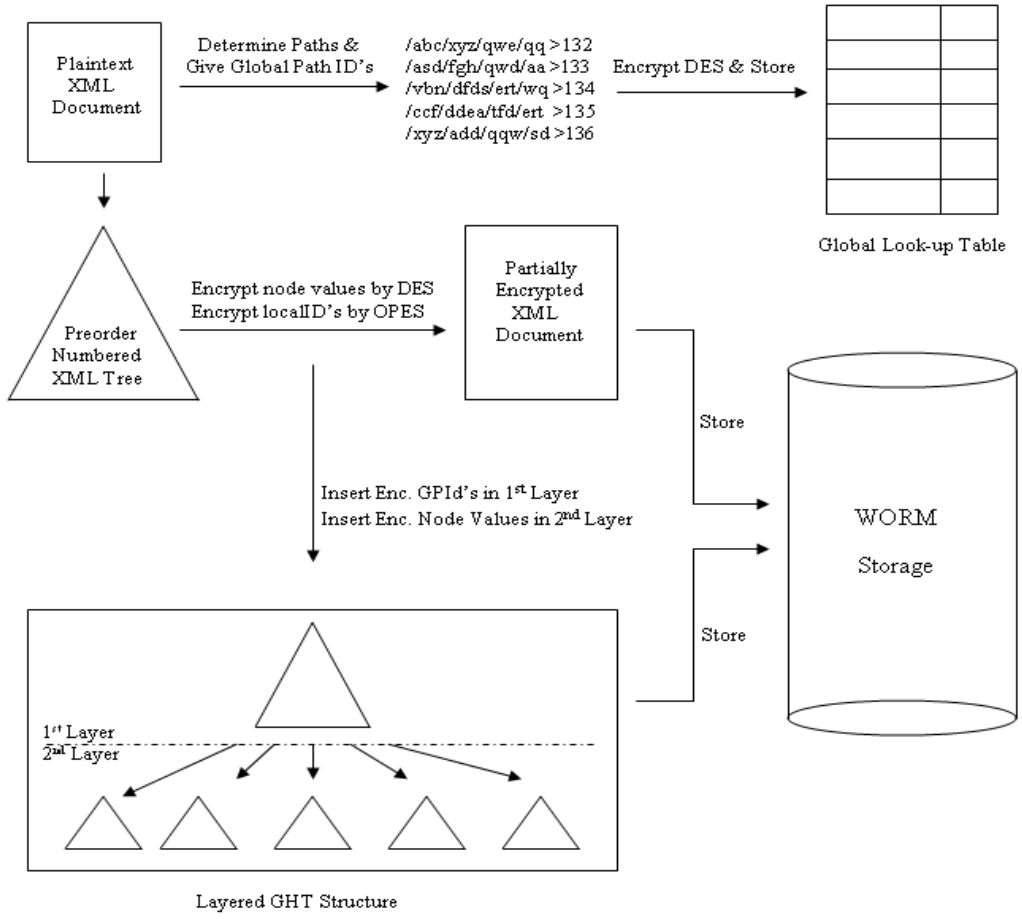


Figure 3.1. General layout of the proposed complete system

The first step in the proposed model is that the medical XML document is taken as an input into the system. Then, the tree representation of the XML document is constructed in the internal memory. All the paths in the XML document are determined by using the tree representation of the XML document. Then, the paths which end at a leaf node are given ID numbers. The system keeps a global look-up table to store the given ID numbers for every XML document stored in WORM storage. All different paths which end at a leaf node are given global path id numbers. The global look-up table is stored in the WORM structure, too. The path values in the look-up table are encrypted with DES algorithm for security reasons. The small number of XML document format types prevents the look-up table having overloading records. The key point here is that if a path is already existed in the look-up table and is given a path id before, then the given path id is used for the path of

the new document. The numbering operation is started from the last remaining number in the insertion of previous medical XML document. By doing this, all new paths of the inserted documents have different path id numbers. This is why we call the term *global path id*.

The next phase is to give local id numbers to each node in the XML tree. A node can contain an element, an attribute or a leaf node value. This numbering operation is done by using the preorder traversing algorithm. But, these given numbers are local numbers for the XML document. The reason why we give numbers to all the nodes regardless of the element, attribute or leaf node is that we index both structure and content. These local path ids will then be encrypted by using OPES encryption algorithm and these encrypted values will be stored in the fossilized indexes as an offset for the faster localization of data in a specified document. These encrypted values will be used for searching. The detailed information about these algorithms will be given in the following sections.

The next phase of the system is encrypting the XML document before storing it in the WORM storage. DES and OPES encryption algorithms are combined and a hybrid encryption schema is proposed to store the medical documents in the WORM storage. As mentioned in the previous sections, security is a necessity for critical medical data. Thus, we extend our work to answer the queries on encrypted data. Also, this encryption schema manages the partial encryption and decryption to prevent the performance decreases due to encryption and decryption operations. Local path ids which are produced in the previous section are used for our encryption schema. An extra encryption flag is set for the field to be encrypted as an attribute value in the XML document.

The specified parts of the given XML document are encrypted with DES encryption algorithm. Then, an "*encryption-data*" element is inserted in the encrypted part and start and end point values are written in *encryption-data* tag as illustrated below. The *start* value is the OPES encrypted value of the local id of the first encrypted node. The *end* value is the OPES encrypted value of the local id of the last encrypted node. The detailed information about these algorithms will be given in the following sections since a general layout is given in this part. The encrypted part of the XML document is figured in Figure 3.2.

<encrypted-data start = 123 end= 231>
encrypted data with DES algorithm
</ encrypted-data >

Figure 3.2. Encrypted form of XML document

Here, 123 and 231 are the OPES encrypted values of, let's say, 14 and 47 respectively. The reason for using OPES encryption schema to encrypt the local path ids is that we must preserve the order of the start and end point values. This will help us to localize a searched data in a given document. As mentioned in the preliminary concepts section, the key property of OPES is to preserve the order of the numeric data after encryption and to enable evaluating comparison queries on the encrypted numeric data without decrypting the data. Since the local path ids are stored in OPES encrypted form in the GHT index structure, system can make a search in the encrypted document to localize the data without decrypting the document.

Searching operation finishes when the interval for the searched data is found. Then, only this part of the whole medical XML document is decrypted. For example, we try to find the data, let's say, with local id 35 in the above document. The OPES encrypted value of 35 is of course between 123 and 231, so the searched data can be localized in the encrypted document. Then, only this part of the document is decrypted.

After encrypting the medical XML document using DES and OPES encryption algorithms, the encrypted document is stored in the WORM storage structure and the global path indexes and the DES encrypted node values are inserted into the proposed fossilized index structure.

The insert and search algorithms of GHT for ordinary documents proposed in [1] have been explained in the preliminary concepts section. In this section, the general properties of our extended GHT data structure will be explained. The detailed information and the pseudo codes of the operations on extended GHT will be given in the following sections. The proposed fossilized indexing method contains a two layered GHT data structure. First layer of GHT contains the global path ids of the inserted documents. 2nd layer GHT contains several small-depth GHT trees for every numbered path. Each GHT in

the 2nd layer is top-down structured and balanced in its own scope, and also maintains the basic properties of GHT data structure.

The insertion of global path ids and the encrypted leaf node values are as follows: Firstly, the global path id values of each numbered path are inserted into the 1st layer GHT data structure. If there is already a bucket for a global path id in 1st GHT, then the DES encrypted leaf node values of this path are inserted into the pointed 2nd layer GHT. But, if the path is inserted into the 1st layer GHT for the first time and has leaf node values, then a new 2nd layer GHT root is created and the encrypted leaf node values are inserted into the 2nd layer GHT. This operation is done for every numbered path in the given XML document. Since the 1st layer GHT tree and the 2nd layer GHT trees are not large-depth trees, the overall performance is reliable and comparable with the standard indexing data structures such as B-Tree. The detailed performance studies of the thesis are presented in the performance study section.

The search operation on this layered index structure is similar to the insertion operation. Search operations are frequently used to perform projection, selection and join operations. While searching a leaf node value in the index structure, the searched value is encrypted using DES algorithm since the values are kept in an encrypted and hashed form in the GHT buckets. Then the global path id of the record is retrieved from the global look-up table. This id is searched in the 1st layer GHT. If the id is found, then the 2nd layer GHT, which is pointed by the 1st layer GHT, is searched to find the hashed value of the record. In this part, the standard GHT search algorithm is used to find the record. If the record is found, the corresponding *document ID* and the OPES encrypted *local ID* is used to localize the record in the document.

The OPES encrypted *local ID* of the record is used to decrypt the appropriate part of the encrypted medical XML document specified by the *document ID*. The comparison operations for finding the appropriate interval in the encrypted XML document are done without decrypting the document. Then the essential data is retrieved from the partially decrypted part of the XML document.

In this section, the general layout of the proposed model has been given. In the following sections, the detailed explanation of each step with examples and essential pseudo codes will be presented.

3.2. Inserting a New XML Document into WORM Storage

The first step of the proposed model is to insert a new XML document into the WORM storage. Before accessing the fossilized index structure to process a query, the insertion of the document into the WORM storage must be specified for a compete model. Thus, as the first step, the insertion of a new medical XML document will be presented.

There are three sub operations to realize this step. These are:

- Global Path Indexing of XML Document,
- Preorder Traversing of XML Document (Local Id Numbering),
- Encryption of XML Document by DES and OPES (Hybrid Encryption Schema)

Before going into the sub operations, it is essential to give an example medical XML document to be used in the following sections for example cases.

As we mentioned before, medical records contain sensitive information about medical procedures such as diagnosis of illnesses, medical treatments of patients, the medicines used, the treatment results of patients and so on [5, 9]. Also, we mentioned about the XML document formats used in medical departments. In this study, we assume that each department in the hospital has different XML document format types. For example, surgery information about a patient is very different from the information about an eye disorder. Also, essential information for a physiologist is very different from the needed information for an orthopedics specialist. Also, it causes performance decreases to specify a general XML document format which is common for every department and physician. Here, in the medical XML document in Figure 3.3, some common tags for all different medical XML document formats are given.

```
<medical-treatments>
    <medical-treatment>
        <patient-info>
            <patient-name>Pelin Korkmaz</patient-name>
            <patient-age>45</patient-age>
        </patient-info>
        <diagnosis-info>
            <disease-name>breast cancer</disease-name >
            <diagnosis-date>12.10.2003</diagnosis-date>
        </diagnosis-info>
        <medicine-info>
            <medicine-name>salsalate</medicine-name>
            <medicine-name>palifermin</medicine-name>
            <medicine-name>busulfan</medicine-name>
        </medicine-info>
    </medical-treatment>
    <medical-treatment>
        <patient-info>
            <patient-name>Ayhan Ersoy</patient-name>
            <patient-age>54</patient-age>
        </patient-info>
        <diagnosis-info>
            <disease-name>tuberculosis</disease-name >
            <diagnosis-date>03.01.2004</diagnosis-date>
        </diagnosis-info>
        <medicine-info>
            <medicine-name>amoxicillin</medicine-name>
            <medicine-name>bisacodil</medicine-name>
        </medicine-info>
    </medical-treatment>
</medical-treatments>
```

Figure 3.3. An example medical XML document

When the medical XML document is taken as an input, the tree representation of the XML document is constructed in the internal memory. The tree representation of the example medical XML document is illustrated in Figure 3.4.

In the figure below, some abbreviations are used because of space constraints. MTs stands for Medical Treatments, MT is Medical Treatment, PI is Patient Info, DI is Diagnosis Info, MI is Medicine Info, PN is Patient Name, PA is Patient Age, DN is Diagnosis Name, DD is Diagnosis Date, and MN stands for Medicine Name. The leaf node values stand for the values in the XML document above.

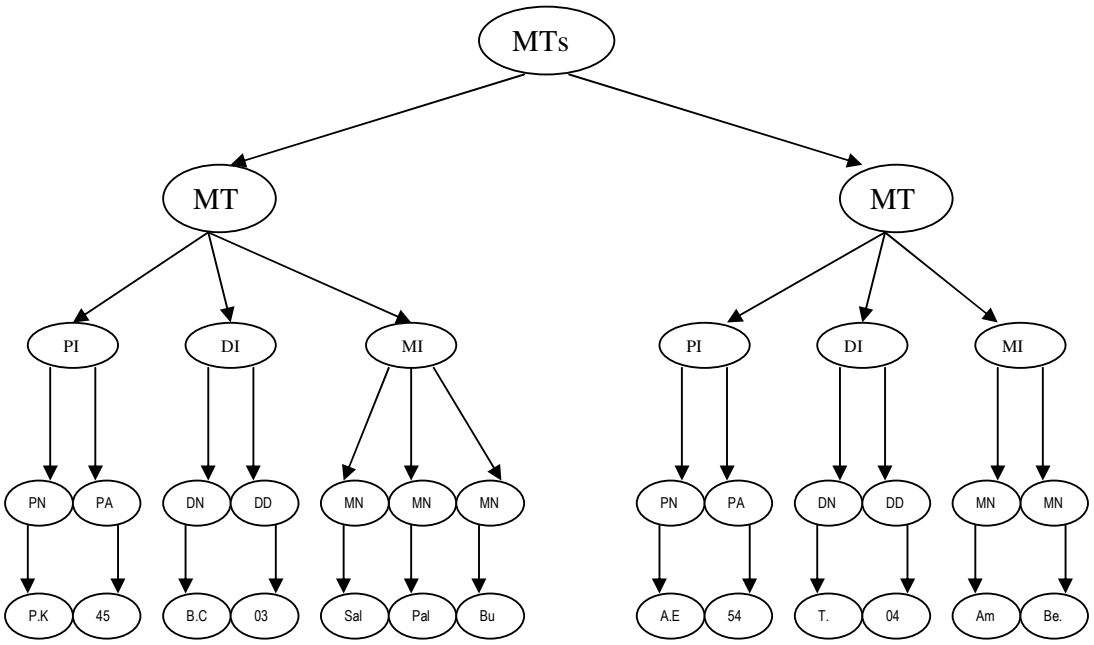


Figure 3.4. Tree representation of the example medical XML document

After presenting the example medical XML document and the tree representation of the document, the sub operations for inserting a given XML document into the WORM storage is explained in the next sections.

3.2.1. Global Path Indexing of XML Document

The first operation after constructing the tree structure from the given XML document is to number all possible paths which end at a leaf node in the XML tree. These numbers are the global path IDs of the possible paths. The path IDs are global because the numbering operation is started from the last remaining number in the insertion of previous medical XML document. For each of the XML documents stored in the WORM structure, all the possible paths and their corresponding global path ID values are stored in a simple look-up table. The key point here is that if a path has already existed in the look-up table and has been given a global path id before, this global path id is used for the path of the new document.

The look-up table is also stored in the WORM structure. Because no modification operation is done on the look-up table, only insertions will be performed on this table as new XML documents are entered into the system. Therefore, the table satisfies the requirements of WORM storage. A tuple is as [Possible Path Name, Global Path Id] in the global look-up table.

Due to the security of sensitive medical data, the *Possible Path Name* column of the look-up table is encrypted by DES encryption algorithm. The important point here is that the look-up table must not be overloaded because of the performance problems. As we mentioned in the previous sections, all the departments in the hospital have different type of medical XML document formats. Since the number of departments is not too big in a hospital, the number of different XML document formats which contains different possible paths is not so many. The look-up table, which has already some data in, is as follows after the insertion of the paths in the given XML document.

[medical-treatments / medical-treatment / patient-info / patient-name, 195]
[medical-treatments / medical-treatment / patient-info / patient-age, 196]
[medical-treatments / medical-treatment / diagnosis-info / disease-name, 197]
[medical-treatments / medical-treatment / diagnosis-info / diagnosis-date, 198]
[medical-treatments / medical-treatment / medicine-info / medicine-name, 199]

Figure 3.5. An example look-up table view

Since the repeated paths are not inserted each time they occur in the XML document, the number of inserted rows into the look-up table is not so much, so this property prevents the table from overloading records. Also, there is one more issue here to increase the overall performance. As we have summarized in the previous section, the *global path ID* values of the document are then inserted into the 1st layer GHT structure. According to the insertion algorithm of GHT data structure, the inserted key value is hashed by a hash function $h_i(x)$ for the level i. Hashing a number is much faster than hashing a long string which contains full path information. Thus, the look-up table increases the overall performance.

3.2.2. Preorder Traversing of XML Document

The next operation is to give local ID numbers to the each element, attribute and leaf nodes in the constructed XML tree. This numbering operation is done by using the preorder traversing algorithm. But, unlike the global path IDs, the scope of these IDs is the XML document not the whole WORM storage. The reason for numbering all the nodes regardless of the element, attribute or leaf node is that we index both structure and content. After giving the *local ID* numbers for all nodes in XML tree, an additional number is calculated for each node. Each node has a number pair (n_1, n_2). The second number n_2 is the total number of siblings of the node n_1. Figure 3.6 illustrates the XML tree after the local parsing sub operation.

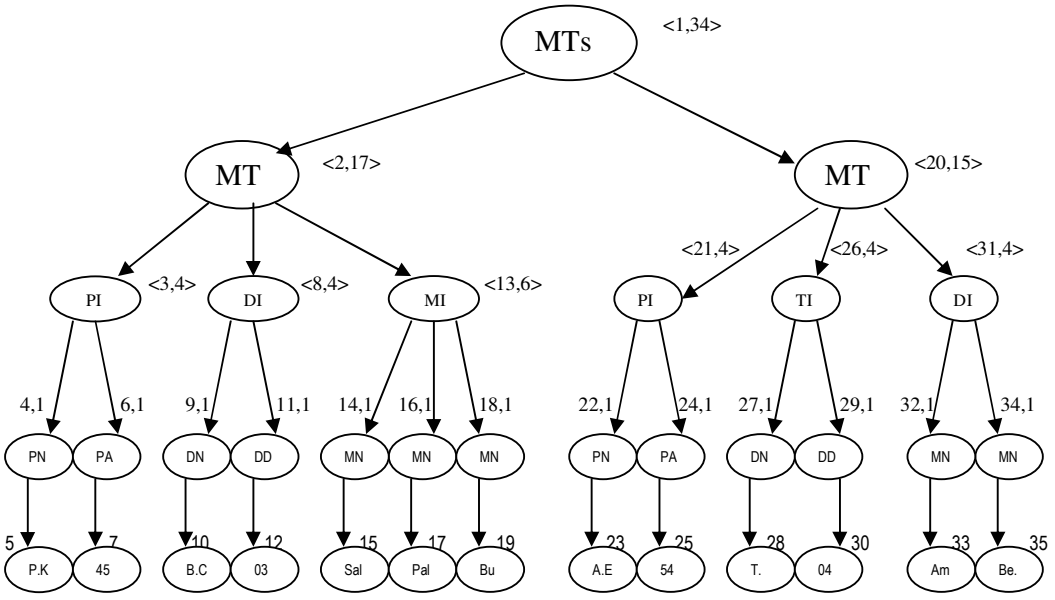


Figure 3.6. Local ID numbering of XML tree

In the Figure 3.6, some abbreviations have been used because of space constraints. MTs stands for Medical Treatments, MT is Medical Treatment, PI is Patient Info, DI is Diagnosis Info, MI is Medicine Info, PN is Patient Name, PA is Patient Age, DN is Diagnosis Name, DD is Diagnosis Date, and MN stands for Medicine Name. The leaf node values stand for the values in our example XML document. The second numbers in the number pairs of leaf nodes are always zero, thus they are not stated in the figure.

The second numbers in the number pairs of XML tree nodes stays for security reasons. They will be used in our hybrid encryption schema. The detailed information will be presented in the next sections. Now, an example can make the subject clear. As mentioned earlier, our encryption schema supports partial encryption and decryption. For example, if a node with <3,4> value pair has an encryption flag which is set to *TRUE*, the nodes starting from 3 to 7 will be encrypted.

The local ID values of the XML tree nodes will be encrypted by using OPES encryption algorithm for a completely secure model. By encrypting the local IDs, the structural information of the medical document is hidden from an adversary. The reason why they are encrypted by OPES instead of DES is that the order of the numbers are preserved after encryption. This will help us to make comparison operations on the encrypted data when searching the correct position of the data in a specified XML document. After encrypting the local IDs by OPES schema, the encrypted values are stored in the fossilized indexes as an offset for the faster localization of data in a specified document.

3.2.3 Proposed Encryption Schema

In this section of the thesis, the proposed encryption schema to store the XML documents in the WORM storage is presented. DES and OPES encryption algorithms are combined and a hybrid encryption schema is proposed to store the medical records. As mentioned in the previous sections, security is a necessity for critical medical data. Therefore, we extend our work to answer the queries on encrypted medical XML data. Also, this encryption schema manages the partial encryption and decryption to prevent the performance decreases due to encryption and decryption operations of data which is not critical.

An extra encryption flag is set as an attribute for the field to be encrypted. Local ID number pairs are used for the encryption schema. For a better understanding of the schema, we add some extra fields into our example medical XML document. An *encryptionFLAG* attribute into *diagnosis-info* and *medicine-info* tags has been added in order to specify that there are some parts to be encrypted. Then the number pairs of these elements are used to

determine which nodes will be encrypted. Some part of the example XML document with encryptionFLAG attribute is shown in Figure 3.7.

```
<medical-treatment>
        <patient-info>
                <patient-name>Pelin Korkmaz</patient-name>
                <patient-age>45</patient-age>
        </patient-info>
        <diagnosis-info encrptionFlag = 'TRUE'>
                <disease-name>breast cancer</ disease-name>
                <diagnosis-date>12.10.2003</diagnosis-date>
        </diagnosis-info>
        <medicine-info encryptionFLAG = 'TRUE'>
                < medicine-name>salsalate</ medicine-name>
                < medicine-name>palifermin</ medicine-name>
                < medicine-name>busulfan</ medicine-name>
        </medicine-info>
</medical-treatment>
```

Figure 3.7. XML document with encryptionFLAG attribute

The number pair of *diagnosis-info* element is <8,4>. This means that the start point of the encryption part is 8, and the end point is 8+4=12. In other words; nodes with local ID 8, 9, 10, 11, and 12 will be encrypted. In the same way, the number pair of *medicine-info* is <13,6>, and this means that the nodes with local ID 13, 14, 15, 16, 17, 18, and 19 will be encrypted in the same manner.

The DES algorithm is used to encrypt the determined nodes in the medical XML document. The key length for DES algorithm is 56 bits in the general case, but for a better security of the sensitive medical data, the key length may be 112 bits with a new encryption algorithm. This algorithm is triple DES which is an extended version of DES with a higher security. After encrypting the marked nodes of the XML tree, an "*encryption-data*" element is inserted into the encrypted part in the XML document. Then, start and end point values are written between the *encryption-data* tags. Since we propose a complete security in our study, the start and end point values are encrypted like the actual data.

The start and end number values can reveal the structure of the XML document. If an adversary knows the preorder traversing is used to number the nodes in XML tree, he can predict the structural properties of the encrypted part in the medical document by the help of these numbers. Thus, this sensitive data is also encrypted. The encryption algorithm to encrypt these number values is OPES encryption method.

The reason for using OPES encryption schema to encrypt the local path ids is that we must preserve the order of the start and end point values in order to localize a searched data in a given document. As mentioned in previous sections, the key property of OPES is to preserve the order of the numeric data after encryption and to be able to compare the encrypted numeric data without decrypting it. Since the local path ids are stored as OPES encrypted in the GHT index structure, system can make comparisons for searching in the encrypted document to localize the data without decrypting the document.

Searching operation finishes when the interval for the searched data is found. Then, only this part of the whole medical XML document is decrypted. The medical XML document is as follows after the fields with encryptionFLAG attribute have been encrypted by the DES algorithm. But the OPES encryption has not been applied yet into the document figured in Figure 3.8.

```
<medical-treatment>
        <patient-info>
                <patient-name>Pelin Korkmaz</patient-name>
                <patient-age>45</patient-age>
        </patient-info>
        <encrypted-data start = 8 end = 12>
                encrypted data with DES algorithm
        </ encrypted-data >
        <encrypted-data start = 13 end = 19>
                encrypted data with DES algorithm
        </ encrypted-data >
</medical-treatment>
```

Figure 3.8. XML document after partial encryption with DES

An "*encryption-data*" element is inserted into the place of the encrypted part and start and end point values are written between *encryption-data* tags as illustrated above.

Then, the start and end points are encrypted by OPES algorithm. Since OPES preserves the order of numbers, the corresponding OPES encrypted values of 8, 12, 13, and 19 may be 33, 58, 67, 91 as an example. Then, the document is shown in Figure 3.9.

```
<medical-treatment>
        <patient-info>
                <patient-name>Pelin Korkmaz</patient-name>
                <patient-age>45</patient-age>
        </patient-info>
        <encrypted-data start = 33 end = 58>
                encrypted data with DES algorithm
        </ encrypted-data >
        <encrypted-data start = 67 end = 91>
                encrypted data with DES algorithm
        </ encrypted-data >
</medical-treatment>
```

Figure 3.9. XML document after partial encryption with DES and OPES

After encrypting the medical XML document using the proposed encryption schema, the encrypted document is stored in the WORM storage structure.

3.3. Proposed Fossilized Indexing System

The indexing of the inserted document with the proposed index structure will be explained in this section. Firstly, the structural properties of the novel indexing method will be presented. Then, the insertion of the global path IDs and the leaf node values into the layered GHT data structure is explained. Also, the pseudo code of the insertion algorithm of the extended GHT will be given for the two different cases. The cases are the insertion of global path IDs into the 1st layer GHT, and the insertion of leaf node values of XML tree into the 2nd level GHT. An example case for the indexing method will be given for understanding the indexing schema better.

3.3.1. Structural Properties of the Indexing System

The proposed fossilized indexing method contains a two layered GHT data structure. The new layered GHT data structure uses the GHT index structure proposed in

[1]. The general properties of this GHT and the pseudo codes of insertion and search operations have been presented in the preliminary concepts section. As mentioned there, the proposed indexing model in [1] only manages basic insert and search operations for ordinary text documents. But, in our proposed model, the indexing model expedites the selection, projection and join queries on the specially encrypted medical XML documents.

The first layer of GHT contains the global path ids of the inserted documents as the key values. The corresponding global path ID of each numbered path is inserted into the 1st layer GHT. 2nd layer GHT contains several small-depth Generalized Hash Trees which are pointed by the nodes of 1st level GHT. This means that each 2nd layer GHT contains the encrypted leaf node values of a specific path in the 1st level GHT. Each global path ID in 1st level GHT has one small-depth GHT among the 2nd layer General Hash Trees for storing its contents.

If we discuss the performance issues of the extended GHT, we can say that the structural property of the layered GHT for XML documents decreases the search time for an XML leaf node value, and thus, increases the overall performance. Also, each GHT in the 2nd layer is top-down structured and balanced in its own scope. This property of each GHT makes the overall performance in scalable intervals compared as standard indexing structures such as B-Tree.

Each GHT alone in the layered structure have the properties of the basic GHT described in [1]. The structural units of the extended version are the same with the basic GHT such as buckets, nodes, hash function and extension factor properties. The extended version has some additional properties to handle the queries send on medical XML documents. The layered structure is constructed for the structural property of the XML documents. XML documents contain the structure and contents together in the same document. In order to handle this property, the first layer manages the structural part while the second layer manages the contents for each path.

3.3.2. Insertions in the Indexing System

After explaining the structural properties of the proposed model, the insertion operations for the layered GHT will be explained in this section. There are two types of insertion operation in this proposed model. The first one is to insert the global path ID values of the paths into the 1st layer GHT data structure. The second one is inserting the encrypted leaf node values into the one of the 2nd layer GHTs. We say *encrypted*, because the leaf node values are encrypted by DES algorithm before inserting them into the GHT. The encrypted leaf node value is inserted into the 2nd layer GHT which is pointed by the owner path in the 1st layer GHT structure.

The algorithmic explanation of the insertion operation of global path ids and the encrypted leaf node values are as follows: Firstly, the global path id values of each path in the given medical XML document are inserted into the 1st layer GHT data structure according to the basic insertion algorithm of GHT. The insertion of encrypted leaf node values is different. To insert an encrypted leaf node value into the 2nd layer GHT, the global path ID of the owner path is searched in the 1st layer GHT. If the search fails, the global path id is inserted into 1st layer GHT.

The pointer value of this bucket is checked if the search is successful. If the pointer of the bucket points to any 2nd layer GHT, it means that an encrypted leaf node value has been inserted for this path before. Then the DES encrypted value of the leaf node of this path is inserted into the pointed 2nd layer GHT data structure. But, if the pointer to the 2nd layer GHT is NULL, it means that no leaf node value for this path has been inserted into the any 2nd layer GHT yet. In this case, a new 2nd layer GHT root is created in the WORM structure, and the encrypted leaf node value is inserted into the 2nd layer GHT. This operation is done for every leaf node value of this path, and also for every path which has been given a global path id in the XML document. The pseudo code for the insertion algorithm of the proposed novel indexing structure is figured in Figure 3.10.

EXTRACT the *path value* of the leaf node
ENCRYPT the *path value* with DES
RETRIEVE the *global path ID* of the encrypted path from look-up table
SEARCH the *global path ID* in the 1st layer GHT
IF the *global path ID* is NOT found THEN {SEARCH returns FAILURE}
    INSERT the *global path ID* into the 1st layer GHT
ELSE IF the *global path ID* is found THEN (SEARCH returns the BUCKET)
    CHECK the *2nd layer GHT pointer value* of the bucket
    IF the *pointer* is NULL THEN
        CREATE the root for a new 2nd layer GHT
        POINT to the root of 2nd layer GHT from the 1st layer GHT
    END IF
    ENCRYPT the *leaf node value* with DES –the key value-
    ENCRYPT the *local ID value* with OPES –the offset value-
    INSERT the *encrypted value* into the 2nd layer GHT
    Return SUCCESS
END IF

Figure 3.10. LAYERED-GHT-INSERT algorithm of layered GHT

The explanations of the functions used in the novel insertion algorithm as follows: The *EXTRACT* function divides the insertion query into two parts, separates the path value from the leaf value. For example, the *EXTRACT* function divides the statement below into two parts.

INSERT medical-treatments/medical-treatment/patient-info/patient-age/'45'

The divided parts are the path value 'medical-treatments/medical-treatment/patient-info/patient-age' and the leaf node value '45', and returns the path value.

The *ENCRYPT* function encrypts the input string with DES or OPES according to the encryption flag value, and returns the encrypted value. The *RETRIEVE* function returns the global path ID value of a path taken as an input to the function. This is done by

accessing the look-up table. Since the path values are stored in DES encrypted form in the look-up table for security reasons, *RETRIEVE* function takes the encrypted path value as input, and makes an encrypted search in the look-up table.

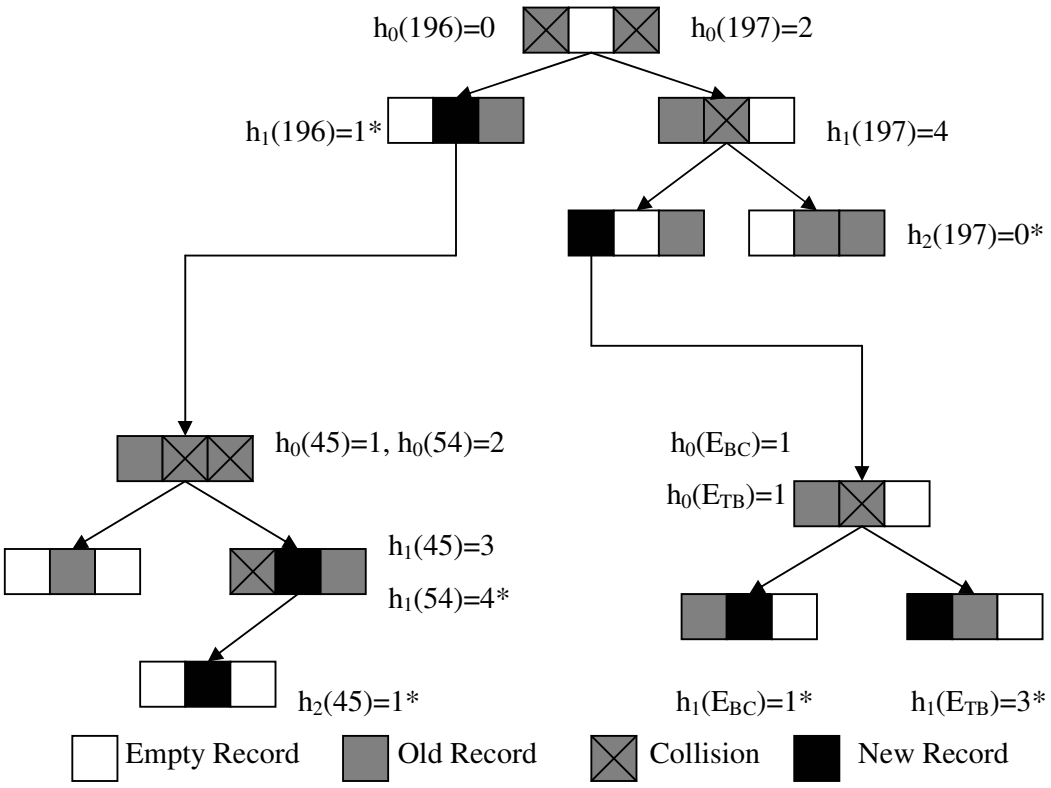
The *INSERT* and *SEARCH* functions are the basic insertion and searching functions of the basic GHT. The search function of our proposed model which will be then used for projection, selection, and join operations will be explained in the next section.

The *CHECK* function controls if the pointer value of a bucket in the 1st layer GHT is NULL. The *CREATE* function allocates a new space in the WORM structure for the root of the new 2nd layer GHT. Finally, the *POINT* function points to the root of the newly created 2nd layer GHT.

Since all the hash trees, which are the one in the 1st layer and the 2nd layer GHT structures, are not large-depth trees, the overall performance is comparable with the standard indexing data structures such as B-Tree. The detailed performance studies of the thesis are presented in the performance study section.

An example case with our sample XML document makes the subject clearer. There are 5 newly inserted global path IDs for the example XML document which are 195, 196, 197, 198, and 199. Also, there are 13 new leaf node values to insert the 2nd layer GHTs. These values are ‘Pelin Korkmaz’ and ‘Ayhan Ersoy’ for the global path ID 195; ‘45’, and ’54’ for the global path ID 196; ‘breast cancer’, and ‘tuberculosis’ for the global path ID 197; ‘12.10.2003’, and ‘03.01.2004’ for the global path ID 198; ‘salsalate’, ‘palifermin’, ‘busulfan’, ‘amoxicillin’, and ‘bisacodil’ for the global path ID 199.

The case explained above can be made clear with an illustration. As an example, the insertion of the encrypted leaf node values of the paths with global path IDs 196 and 197 into the 2nd level GHT is illustrated in Figure 3.11.



* Stars indicate successful insertions.

m=3 and k=2, where m is the size of three node and k is the growth factor.

Figure 3.11. Insertion of leaf node values into 2nd layer GHT

As illustrated in the Figure 3.11, the global path ID values 196 and 197 have been inserted into the 1st layer GHT. The leaf node values '45' and '54' have been inserted into the 2nd layer GHT which is pointed by the global path ID 197. These values are not encrypted since the *encryptionFLAG* of these records are not set to TRUE in our example medical XML document.

The encrypted leaf node values 'breast cancer' and 'tuberculosis' indicated as E_{BC} and E_{TB} respectively have been inserted into the 2nd layer GHT which is pointed by the global path ID 198. When inserting the global path IDs and leaf nodes, some collisions occurred in some levels. The corresponding hashed values of the inserted values in each level are given near the buckets. Also, we marked the successful insertions with a star sign.

The insertion of the encrypted value of ‘tuberculosis’ can be explained as follows: Firstly, the path value of ‘tuberculosis’ value is extracted from the query. Then, the DES encrypted value of this path is searched in the look-up table. The corresponding global path ID of this path is retrieved from the table. Then, the global path ID 197 is inserted into the 1st layer GHT with two collisions in level 0 and level 1. In level 2, the value is successfully inserted into the first bucket since $h_2(197) = 0$. Then, the leaf node value is encrypted with DES algorithm. Also, the localID value of this leaf node, which is the offset of the record, is encrypted with OPES algorithm. Then, the encrypted value of ‘tuberculosis’, which is indicated as E_{TB}, is inserted into the 2nd layer GHT that the bucket with global path ID 197 points to. There have been a collision in the first layer in the 2nd layer GHT since there have been some other insertions for this path before this insertion. But, in the second level, the insertion is successful and the E_{TB} value is inserted into the forth bucket since $h_1(E_{TB})=3$. The other values are inserted in the same way.

The inserted record of the encrypted leaf node value in the 2nd layer GHT contains the encrypted value of the leaf node and an array. The array consists of records which contain a document ID and a local ID. The localID is an offset value which will be used for fast searching in the document with Document ID. An example record is as shown in Figure 3.12.

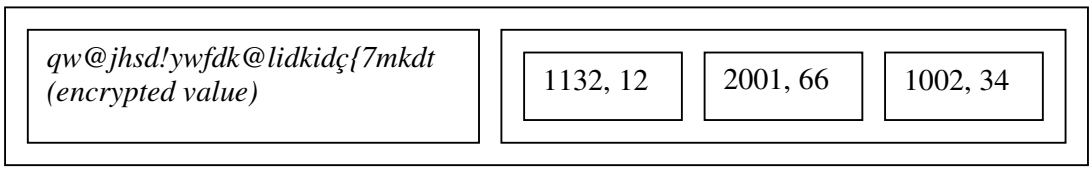


Figure 3.12. Structure of an inserted record in 2nd layer GHT

This record states that the encrypted value ‘*qwjhsdywt’* is present in the documents with document IDs 1002, 1132, and 2001. The OPES encrypted local ID offset values of these documents are 34, 12, and 66 respectively. Here, one important point is that the local ID offset values of the leaf node values are encrypted by using OPES encryption schema. The reason is to search the XML document partially without decrypting the whole document since OPES preserve the order and allows comparison operations on encrypted data.

3.3.3. Searching in the Indexing System

In this section, the searching operation is presented with the algorithmic explanations and pseudo codes. Before explaining the projection, selection, and join operations on the specially encrypted medical XML document, it is essential to describe the searching operations in the indexing model. The searching operations are performed in the stages of projection, selection and join operations.

The search operation on this layered index structure which uses extended GHT is similar to the insertion operation. While searching a leaf node value in the index structure, the searched value and also its path information is encrypted using DES algorithm since the values are kept in an encrypted form in the 2nd layer GHT buckets and in the look-up table respectively. Then the global path id of the record path is retrieved from the global look-up table by using the encrypted path information. This ID is searched in the 1st layer GHT. The searching algorithm of the basic GHT is used to search the global Path ID of the leaf node value. If the ID is found, then the 2nd layer GHT pointed by the global path ID in the 1st layer GHT is searched. Then, the standard GHT search algorithm is used to find the record in the 2nd layer GHT. The pseudo code for the search algorithm of our novel indexing model is presented below in Figure 3.13.

```
EXTRACT the path value of the leaf node
ENCRYPT the path value with DES
RETRIEVE the global path ID of the encrypted path from look-up table
SEARCH the global path ID in the 1st level GHT
IF the global path ID is NOT found THEN {SEARCH returns FAILURE}
        Return FAILURE
ELSE IF the global path ID is found THEN (SEARCH returns the BUCKET)
        CHECK the 2nd level GHT pointer value of the bucket
        IF the pointer is NULL THEN
                Return FAILURE
        ELSE IF the pointer is NOT NULL THEN
                ENCRYPT the leaf node value with DES
                SEARCH the encrypted leaf node value in the 2nd level GHT
```

```
                IF it is NOT found THEN {SEARCH returns FAILURE}
                    Return FAILURE
                ELSE IF encrypted value is found THEN
                    Return the BUCKET
                END IF
            END IF
        END IF
```

Figure 3.13. LAYERED-GHT-SEARCH algorithm of layered GHT

If the record is found in the 2nd level GHT and the function returns the bucket, the corresponding *document ID* and the OPES encrypted *local ID* is used to localize the record in the document with *document ID*. The OPES encrypted *local ID* of the record is used to decrypt the appropriate part of the encrypted medical XML document. The comparison operations for finding the appropriate interval in the encrypted XML document are done without decrypting the document. This is done by the help of key property of OPES which preserves the order. When the location of the searched data in the medical XML document has been found with the comparison operations on the encrypted data, only this part of the whole medical XML document is decrypted Then the essential data is retrieved from the partially decrypted part of the XML document specified with *document ID*.

3.3.4. Projections, Selections, and Joins in the Indexing System

In this section, accessing the proposed index structure to expedite the projection, selection and join operations will be discussed by sending example queries on our DES and OPES encrypted medical XML document.

3.3.4.1. Projection Queries

Firstly, we explain how the projection operations performed on the encrypted XML documents are expedited by accessing the novel indexing structure. Performing a projection operation is equal to retrieving all the data for a specified path in this structure.

For a better understanding, we give an example case for our example encrypted medical XML document. The example projection query asks the question below:

Question: List the disease names diagnosed so far in the hospital.

The corresponding representation of that query is like that:

/medical-treatments/medical-treatment/diagnosis-info/disease-name

When the projection query above is given into the system, the proposed indexing schema is accessed to find the related documents in order to decrease the search time of the operation. For that purpose, firstly the path value in the query is encrypted with DES algorithm and then the corresponding global path ID information is retrieved for that path from the look-up table.

The global path ID of this path is 197 for our example document. After, the global path ID is searched in the 1st layer GHT with the basic search operation of the GHT data structure. If the key is found in the 1st GHT, the pointer information of this bucket is checked, and the 2nd layer GHT that this bucket points to is accessed in the WORM storage. The key 197 is found in the first bucket of the first node of level 2 in the 1st layer GHT in the index of our example document. Then, the 2nd layer GHT is accessed. The *document ID* and *local ID* values of all buckets are retrieved from the 2nd layer GHT.

When this operation is applied on our example document, the value pairs [5, 33] and [5,121] are obtained. 5 is the document ID value of our example document in the WORM structure. 33 is the OPES encrypted value of 10 which is the local ID of 'Breast Cancer' and 121 is the OPES encrypted value of 28 which is the local ID of 'Tuberculosis' in the encrypted medical XML document. Then, these values are used to access the encrypted document with Document ID 5, and the OPES encrypted local ID offset values are used for fast localization of data on the encrypted data without decryption in the found document.

3.3.4.2. Selection Queries

In this section, we explain how the selection operations are fastened by accessing the novel indexing structure. In our indexing structure, performing a selection operation is similar with the search operation explained in the previous sections. For a better understanding, we give an example case for our example encrypted medical XML document. The example selection query asks the question below:

Question: List the medicine names used with the palifermin medicine.

The corresponding representation of that query is like that:

/medical-treatments/medical-treatment/medicine-info
[medicine-name= 'palifermin'] /medicine-name

When the selection query above is given into the system, the path value of the query is extracted firstly. The path value in the query is encrypted with DES algorithm and then the corresponding global path ID information is retrieved for that path from the look-up table. The global path ID of this path is 197 for our example document. After, the global path ID is searched in the 1st layer GHT with the basic search operation of the GHT data structure. If the key is found in the 1st GHT, the pointer information of this bucket is checked, and the 2nd layer GHT that this bucket points to is accessed in the WORM storage. The key 199 is found in the 1st layer GHT, and the 2nd layer GHT that the bucket with 199 global path ID points to is accessed.

Another search operation is performed in the 2nd layer GHT after that. For that purpose, the 'palifermin' value is encrypted using the DES algorithm, and the encrypted value is searched according to the basic search algorithm of standard GHT structure. When it is found in the 2nd layer GHT, the *document ID* and *local ID* values of the found buckets are retrieved. When this operation is applied on our example document, the value pair [5, 81] is obtained. 5 is the document ID value of our example document in the WORM structure. 81 is the OPES encrypted value of 17 which is the local ID of 'palifermin' in the encrypted medical XML document. Then, these values are used to access the encrypted

document with Document ID 5, and the OPES encrypted local ID offset value is used for fast localization of data on the encrypted data without decryption in the found document. When the data is found, the needed part of the document, which is specified by the OPES encrypted start and end points in the document, is decrypted and the medicine names used together with the 'palifermin' is listed as the query response values, which are 'salsalate' and 'busulfan' in our example document.

3.3.4.3. Join Queries

Lastly, in this section, we explain how the join operations performed on the encrypted XML documents. Performing a join operation is similar with two staged search operation explained in the previous sections. The example join query asks the question below:

Question: List the diagnosis dates of diseases which had a surgical operation.

In order to explain the join operations better, we need one more medical XML document format used in the hospital. For that purpose, we have constructed an example XML document format for the surgery department since our question above needs a join operation with the medical XML documents about surgical operations.

The corresponding representation of that query is like that:

/medical-treatments/medical-treatment/diagnosis-info [disease-name = /surgery-operations/surgery-operation/disease-info/disease-name]/diagnosis-date

When the join query above is given into the system, the proposed indexing schema is accessed to find the related documents. Firstly, the query is divided into two parts. In our example query, the divided parts are */medical-treatments/medical-treatment/diagnosis-info/disease-name* and */surgery-operations/surgery-operation/disease-info/disease-name.* After, the corresponding global path ID values are retrieved from the look-up table, which are 197 and 215 for the above paths respectively in our example document. Then the 2nd level GHTs that the buckets 197 and 215 in the 1st layer GHT point to are accessed. The

encrypted leaf value nodes of these GHTs are retrieved and then equality operation is performed on the encrypted leaf node values of the records. The records which have the same *disease-name* values in the two GHT are obtained after this operation. Then, the [document ID, local ID] number pairs of the final records are used to access the encrypted documents with specified Document IDs.

The OPES encrypted local ID offset values are used for fast localization of data on the encrypted data without decryption in the found document. When the data is found, the needed part of the document is decrypted and the *diagnosis-date* value is retrieved from the partially decrypted documents as the query response values.

4. PERFORMANCE STUDY

In this section, the performance study of the proposed system will be explained. Firstly, the detailed information about the simulation program will be given and then the obtained simulation results will be illustrated with resulting graphs.

We have implemented the pseudo codes given in the previous sections in order to demonstrate that the proposed system gives satisfactory results. Also, we have compared space and time complexities of the proposed data structure with basic GHT and rewritable B-Tree. We have obtained satisfactory results. The programming language that we use for the implementation is Java 1.5, and the development environment is Eclipse 3.2. The operating system used is Microsoft Windows XP Professional Version 2002 with Service Pack 2. The machine used for the simulations has an Intel(R) Pentium(R) 4 CPU with 2.80 GHz, and 1 GB internal memory.

4.1. Main Properties of the Simulation Program

The simulation program has been implemented by using Java programming language on the Eclipse 3.2 integrated development environment. The program firstly takes a valid file path from the text filed. Also, the user may choose the file to be parsed from the file chooser. Then “Insert the File into Layered GHT” button is pressed to insert the specified document into the layered GHT and WORM storage. All the medical XML documents are inserted into the GHT and WORM storage by this way. There are several sub operations in this insertion process.

The program firstly parses the inserted medical XML document, and extracts the path and leaf node values. The encrypted values are inserted into the layered GHT structure by using the new insertion algorithm. Then, the document is partially encrypted and inserted into the WORM storage.

The general view of the simulation program is shown in Figure 4.1.

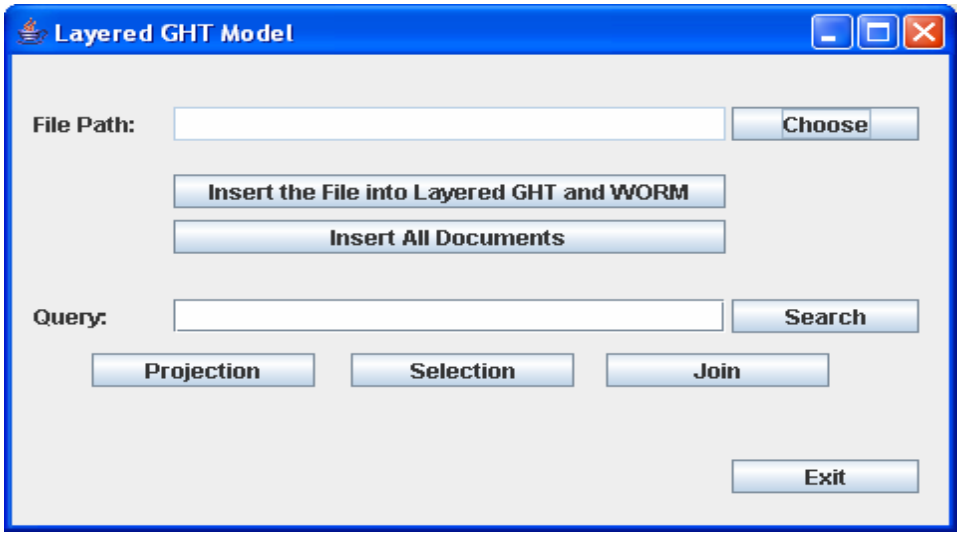


Figure 4.1. General view of the simulation program

The simulation program can perform search, projection, selection and join operations of the proposed system. The query is written in the query text field, and then the button of the desired operation is pressed. Then, the program displays the output of the processed operation if the operation is successful.

There are several sub operations of search, projection, selection and join operations of the simulation program as in the insertion operation. In the search operation, when the search query is written into the text field, firstly the path and leaf node values of the query are extracted. Then the encrypted values of these keys are searched in the layered GHT data structure. If the searched leaf node value is found in the 2nd layer GHT, the program displays the search time, the document ids and the offsets of the documents which contain the searched leaf value.

The flow for the projection operation is as follows: Firstly, the path information is entered as input into the search text field area. Then, the program searches for the path info in the 1st layer GHT structure after the corresponding global path id is found in the global look-up table. If the path key is found in the 1st layer GHT, the projection time, the

document ids and offset values of the documents which contain the searched path are displayed. Then, the found documents are partially decrypted by the help of offset values, and the corresponding results values are displayed.

Selection operation performs nearly the same operation with the search operation. It additionally decrypts the documents with the found document ids, after finding the searched leaf node value in the 2nd layer GHT. Then, it displays the final result values.

The join operation can be thought as two-part selection operation for this proposed structure. After the join query is entered into the text filed area, the entered query is divided into two distinct queries. Then each query is processed alone, and the different final results are obtained for each query. After that, the equalities of the final values are checked, and the values which satisfy the equalities are displayed.

As an example, our medical document, which document id is 123, presented in the above sections is inserted into the layered GHT and the WORM storage by using the simulation program. The leaf node values, the document and the look-up table is encrypted and saved into the WORM storage. Then, an example selection query is written into the search text filed area. The simplified version of the corresponding displays for the example is as follows:

Insertion Log: Insertion time: 15 milliseconds!!

Query: /medical-treatments/medical-treatment/diagnosis-info [disease name = 'tuberculosis']/diagnosis-date

Search Key: tuberculosis

DocID: 123

Offset: 18

Selection Result: 03.01.2004

Selection Time: 16 milliseconds!!

Figure 4.2. Display of a selection result from the simulation program

4.2. Methodology

We have used simulation with some prepared medical data in order to demonstrate that the proposed complete system gives satisfactory results. Also, we have compared our writeonce layered GHT indexing structure, with rewritable B-tree. The rewritable B-tree is a balanced tree, and all the leaves in a B-tree are on the same level and all the nodes except for the root and the leaves have at least k=2 children. The rewritable B-tree splits a node that overflows in place and is, hence, more space-efficient than its writeonce counterpart.

The two metrics that we focus on are the time and space cost. Since storage has become relatively inexpensive, the space cost has become less of an issue. Nevertheless, with the large volume of records typical today, it is still important to ensure that the space overhead is not excessively high. Also, the time complexity is a critical determinant of insertion and access performance.

The most important issue in performance comparison here is that we have implemented a complete system, not only an indexing structure which only manages simple insert and search operations. Also, the proposed indexing structure manages the XML data which contains both path and content information. Moreover, the layered GHT indexing structure is designed for the documents stored in WORM storage structures. The indexing structure itself is stored in WORM storage, too. Thus, these properties bring some performance losses in time and space complexities, compared with its rewritable counterparts such as B-Tree. Also, an encryption phase sacrifices some execution time in order to secure the complete system. We have presented the performance graphs for each case in the following section. Thus, the main purpose of the complete system is to obtain satisfactory results.

We have created a dataset which contains XML-based medical documents from different departments in the hospital. The dataset contains 120 medical document files from six departments. Each department has 20 documents. Each document contains 50 records, and each record contains approximately 12 leaf node values and path values. This means that, inserting a record corresponds to inserting 600 leaf node and path values, in other words 1200 items. Thus, the number of all items in the dataset is 1200 * 120 =

144000. The total size of the complete dataset is 2.45 MB. The machine for the simulations has Intel(R) Pentium(R) 4 CPU with 2.80 GHz, and 1 GB internal memory.

4.3. Time and Space Complexity Results

In this section, time and space complexity results will be given with different cases. Firstly, general time and space complexities of insert, search, projection, selection and join operations will be presented. Then, how time and space complexities change when encryption is added to the complete system will be explained by resulting graphs.

4.3.1. Time and Space Complexities of Operations

The main functions of the complete model have been simulated with some prepared medical data in the simulation program and the time and space complexities of these cases without encryption are presented by the graphs. The Figure 4.3 shows the time complexity of insert operation while the Figure 4.4 displays the space complexity of insert operation.

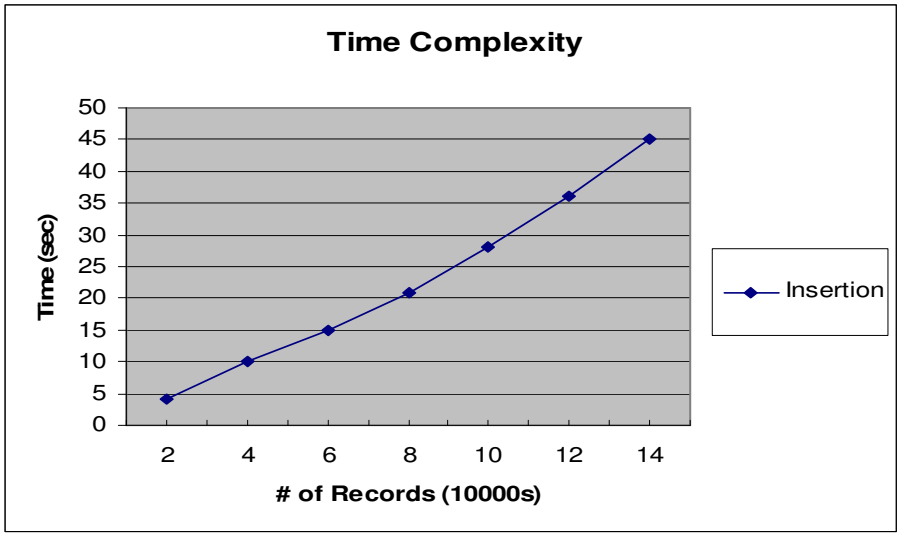


Figure 4.3. Time complexity of insertion operation of layered GHT

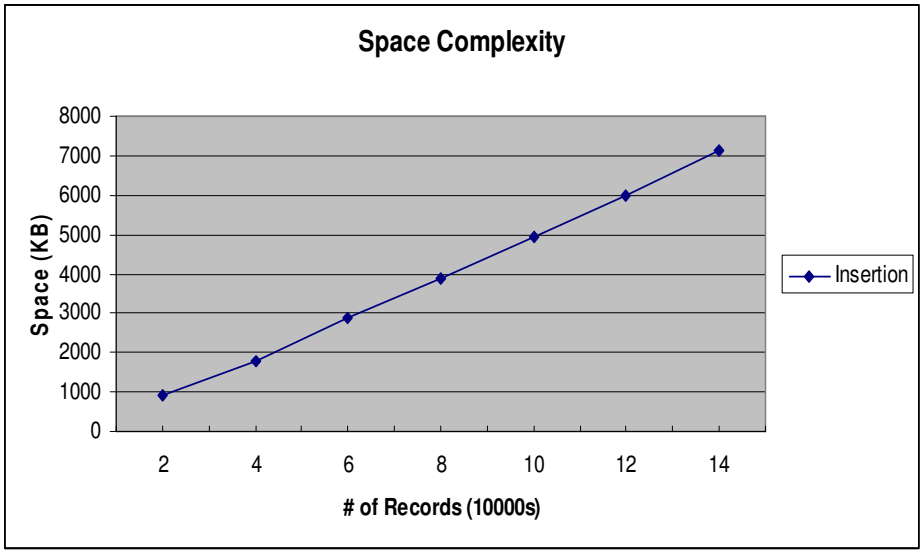


Figure 4.4. Space complexity of insertion operation of layered GHT

As seen in the Figure 4.3 and Figure 4.4, the operation time inserting all the records in the data set, which contains 120 documents with totally 144000 records is 45 seconds. The space to save the layered GHT with this data in the WORM storage is 7120 KB. Also, each graph has nearly linearly increasing lines. This means that as the number of records for the insertion operation increases, the insertion time and the space needed to save the layered GHT increases linearly. This is a desired result for time and space complexities.

The Figure 4.5 compares the time complexities of search, projection, selection and join operations on the inserted 144000 items. Several different search, projection, selection and join queries have been sent on the medical dataset stored in the WORM storage. The time complexities of different queries for an operation type vary in small amounts with the query sent on the dataset. Thus, the results on the graph are the average results of many different query processing times.

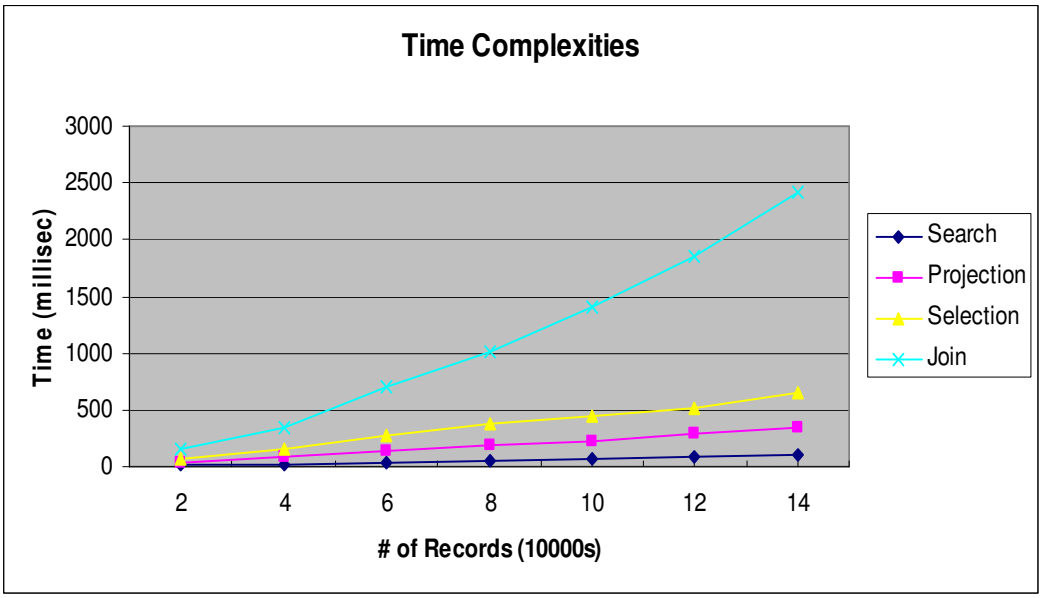


Figure 4.5. Time complexities of search, projection, selection and join operations

The Figure 4.5 shows that a search operation on a dataset which contains 144000 items takes 105 milliseconds which is a very satisfactory result for a search operation. The search operation is the fastest one since the search operation only tries to find the searched value in the layered GHT. The other three operations take more time to process compared with the search operation. This is mainly because of the time losses resulting from retrieving the query results from the specified XML documents among all the documents in the WORM storage.

Also, the projection operation is seemed faster than the selection operation. This is mainly because of saving all the document ids and offsets for a specified path in the 1st layer GHT buckets. By the help of this property, there is no need to parse all the nodes in the 2nd layer GHT which is pointed by the path in the 1st layer GHT. Since the join operation checks the equalities of the values for two different paths, it has not a perfect linearly increasing line. As the number of records increase, the time increase in join operation is much more than the other operations.

4.3.2. Sensitivity to Encryption

In this section, the encryption process has been added to make the complete system more secure. Since there has been added an extra encryption phase, some time has been sacrificed for encryption process in order to make the system more secure. We have made simulations with the same amount of data by encrypting them. In the first simulations, only the content information of XML documents has been encrypted. Encrypting only the content means encrypting the leaf node values inserted into the layered GHT and the content values in the XML document stored in the WORM storage. The OPES encryption is not used in this step since the path information is not encrypted. Satisfactory results have been obtained compared with the unencrypted version.

Then, both the path and content information of medical XML documents have been encrypted. Both the DES and OPES encryption algorithms have been used in this step. So, there have been more time losses compared with the only content encryption version. The graphs below show the time and space complexity differences between the three versions. These are unencrypted, content encrypted, paths and content encrypted versions. Approximately half of the data set has been encrypted for simulations.

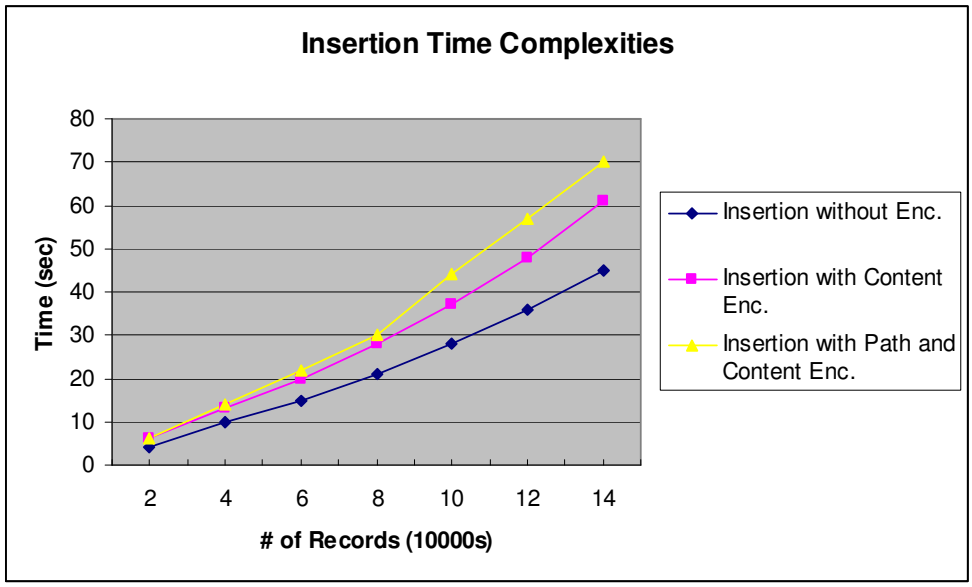


Figure 4.6. Time complexities of insertion for three different encryption simulations

The Figure 4.6 shows that the insertion of 144000 items into the layered GHT and the WORM storage with content encryption takes 35% more time. This is because of the encryption process of the leaf node values in the medical XML documents. Also, the path and content encryption time complexity results are up to 55% less efficient than the unencrypted version. This is because of both path and content encryption of the XML documents and items inserted into the layered GHT. The look-up table stored in the WORM storage is also encrypted in this case. Thus, this phase takes some more time in the complete system.

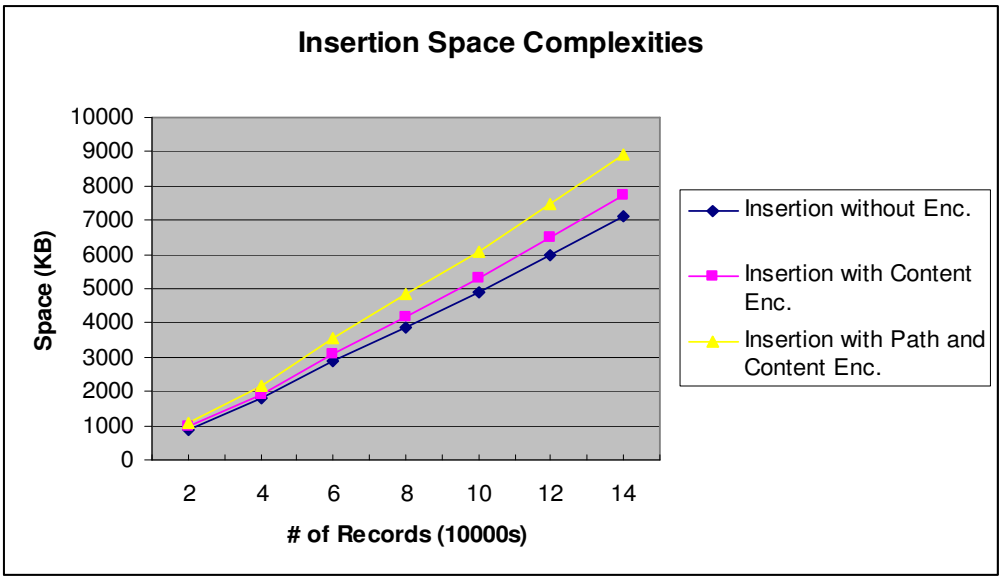


Figure 4.7. Space complexities of insertion for three different encryption simulations

The Figure 4.7 shows that the insertion of 144000 items into the layered GHT and the WORM storage with content encryption takes 8% more space. This is because of the long encrypted strings resulting from DES encryption. Also, the path and content encryption results for space complexities are equal or up to 25% less efficient than the unencrypted version. This is because of long length of the encrypted path and leaf node values of the XML documents and items inserted into the layered GHT. The look-up table stored in the WORM storage is also encrypted in this case. Thus, this phase takes some more space in the complete system.

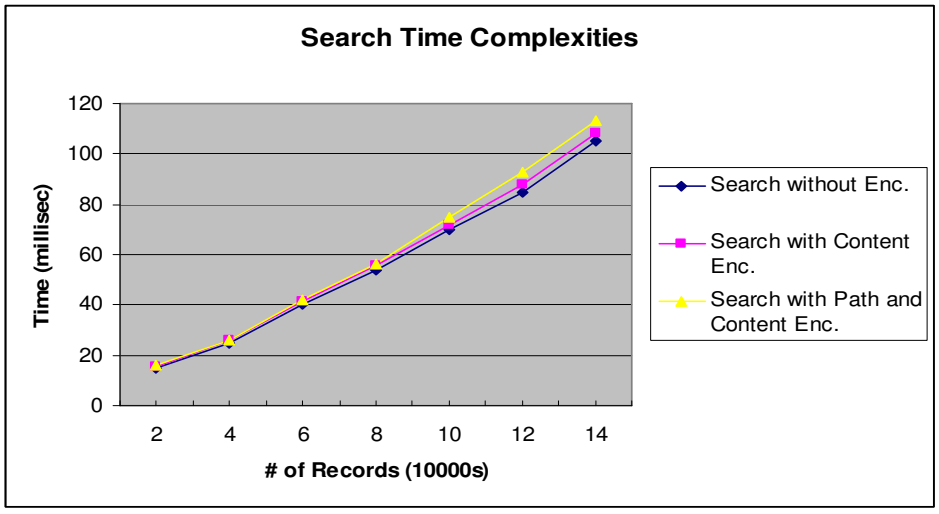


Figure 4.8. Time complexities of search for three different encryption simulations

The Figure 4.8 demonstrates us that search operations on the encrypted medical XML documents is less efficient in time complexity results compared with the unencrypted version. This is because of the encryption of leaf node values before searching. Since this process does not take so much time in the complete system, the content encrypted version is 3% less efficient. Also, the search operations on the path and content encrypted dataset are up to 5% less efficient in time complexities.

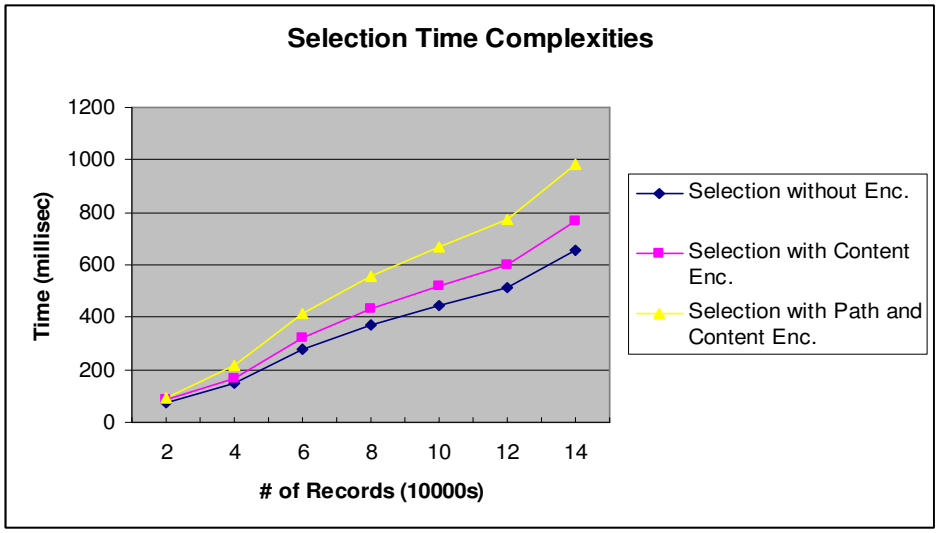


Figure 4.9. Time complexities of selection for three different encryption simulations

The Figure 4.9 shows that the selection operations on the encrypted dataset are less efficient in time complexity compared with the unencrypted version. But the increase in time losses in the selection operation is more than the increase in the search operation. This is mainly because of the much more time losses resulting from retrieving the query results from the encrypted XML documents. When the encryption is applied on both path and content data, the time loss is equal or up to 29% while the time loss in the only content encryption is 17%.

4.4. Comparison Results with B-Tree

In this section, we compare the witeonce layered GHT with the rewritable B-tree. The space complexity results of insert operations for both indexing structure are summarized in Figure 4.10.

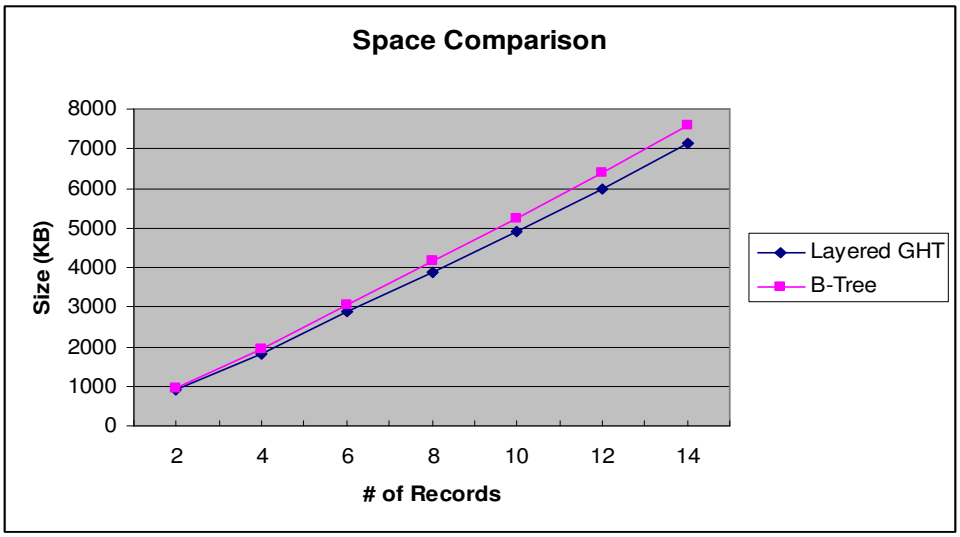


Figure 4.10. Space complexity comparison results for insertion operation

The layered GHT structure has an advantage in space efficiency over the B-tree that exceeds 7%. When k and m is configured well and linear probing is used, the layered GHTs incur reasonable space cost. The time complexity comparisons of insert and search operations for both indexing structure are summarized in the Figure 4.11 and Figure 4.12.

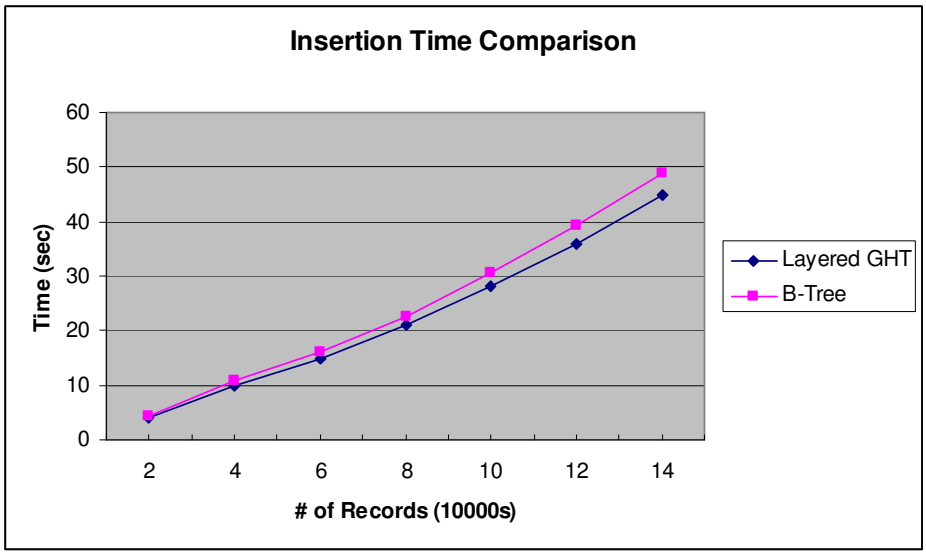


Figure 4.11. Time complexity comparison results for insertion operation

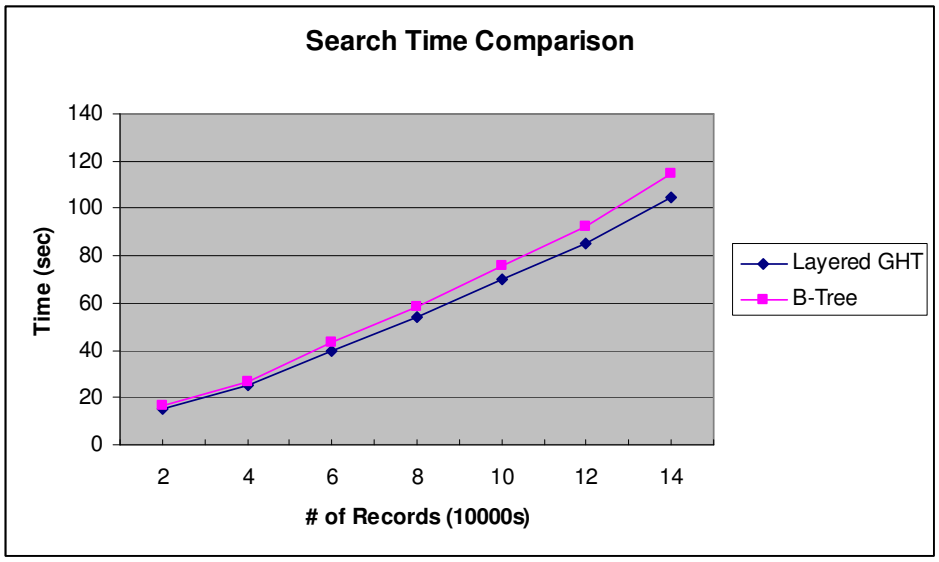


Figure 4.12. Time complexity comparison results for search operation

The B-tree performs well but is vulnerable to logical tampering of records. The layered GHTs are able to satisfy the requirements of an index for WORM storages and yet perform equal or up to 9% better than the B-tree in time complexities. This indicates that hashing works very well in practice. As a result, the layered GHTs are well-balanced, implying good search and access performance. These properties also lead to good

projection, selection and join processing times. But, we can not compare projection, selection and join operations since B-Tree does not support these operations.

We obtained these comparison results for only insert and search operations by inserting same amount of item into the layered GHT and B-Tree structures. In the previous sections, we had obtained that projection, selection, and join operations are less time efficient than search operation of layered GHT. Thus, of course, projection, selection and join operations of the layered GHT structure is less time efficient than B-tree search operation. This is mainly because of the time losses resulting from retrieving the query results from the specified XML documents. Also, the encryption phase brings some time and space loss if a secure complete system is a necessity. But, these losses are mandatory to form a complete model which makes projection, selection and join operations over encrypted XML-based medical documents.

The layered GHTs incur reasonable space cost and offers good performance, thereby showing that an index structure for XML-based medical documents stored in the WORM storage can be achieved.

5. CONCLUSION

The trustworthiness of the sensitive medical data is critical for hospitals and for the public care in general. In the thesis, we explained that the storage of the medical data in WORM structures is not adequate for the trustworthiness of the critical data. The records stored in WORM storage can be logically modified if the index can be suitably manipulated. Also, we have given detailed information about the GHT data structure that we use for our proposed indexing system. Also, we have explained DES and OPES algorithms that we use in our encryption schema. Then, the growing usage of XML standards in medical record keeping has been discussed. Additionally, we have explained that the XML data stored in the WORM storage may also be encrypted in order to protect the sensitive data and satisfy security requirements.

After explaining the preliminary concepts and the related work, we have presented our novel indexing structure and the encryption schema for the encrypted medical XML documents stored on WORM structures. For that purpose, we have firstly explained the insertion of a given medical XML document into the WORM storage with its sub operations. Then, we have presented our encryption schema which combines DES encryption algorithm with OPES. The novel indexing technique which expedites projection, selection and join operations on encrypted medical XML records kept in WORM storages have been presented. We have given the algorithms and pseudo codes of the proposed fossilized indexing schema. Also, we have given some examples on a medical XML document for a better understanding of our proposed fossilized indexing system and the encryption schema. We have demonstrated that the proposed system give satisfactory performance.

REFERENCES

1. Zhu, Qingbo and Windsor W. Hsu. Fossilized Index: The Linchpin of Trustworthy Non-Alterable Electronic Records. Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 395-406, 2005.

2. Agrawal, Rakesh, Jerry Kiernan, Ramakrishnan Srikant and Yirong Xu. Order Preserving Encryption for Numeric Data**.** Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 563-574, 2004.

3. Wang, Haixun, Sanghyun Park, Wei Fan Philip and S. Yu. ViST: A Dynamic Index Method for Querying XML Data by Tree Structures. Proc. SIGMOD, pp. 110-121, 2003.

4. United States Department of Health and Human Services. National Standards to Protect the Privacy of Personal Health Information. Available at http://www.hhs.gov/ocr/hipaa/finalreg.html, 2006.

5. How to Use an Internet-Based Medical Records Repository and Retain Patient Confidentiality. Available at http://www.hipaadvisory.com/action/patientconf.htm, 2006.

6. World Wide Web Consortium. Extensible Markup Language (XML) 1.1. Available at http://www.w3.org/TR/2004/REC-xml11-20040204, 2004.

7. Singer, David and S. C. Gultekin Ozsoyoglu. Anti-tamper Databases: Querying Encrypted Databases. In Proc. of the 17th Annual IFIP WG 11.3 Working Conference on Database and Applications Security, pp. 25-59, Estes Park, Colorado, August 2003.

8. Iyer, Bala, H. Hacigumus, C. Li and S. Mehrotra. Executing SQL Over Encrypted Data in the Database-Service-Provider Model. In Proc. of the ACM SIGMOD Conf. on Management of Data, pp. 44-69, Madison, Wisconsin, June 2002.

9. Personal Privacy and Access to Medical Databases. Avaliable at http://www.lbl.gov/Education/ELSI/privacy-main.html, 2006.

10. Cohasset Associates, Inc. The Role of Optical Storage Technology. White Paper, pp. 11-30, April 2003.

11. EMC Corp. EMC Centera Content Addressed Storage System. Available at http://www.emc.com/products/systems/centera_ce.jsp, 2006.

12. IBM Corp. IBM TotalStorage DR550. Available at http://www-1.ibm.com/servers/storage/disk/dr, 2006.

13. Broder, A. Z. and A. R. Karlin. Multilevel Adaptive Hashing. In 1st ACM-SIAM Symposium on Discrete Algorithms, pp. 1-35, 1990.

14. Stinson, D. R. Cryptography: Theory and Practice. CRC Press, 2nd edition, pp. 95-140, 2002.

15. Electronic Privacy Information Service. Medical Privacy. Available at http://www.epic.org/privacy/medical, 2006.

16. Health Level Seven Inc. Available at www.hl7.org, 2006.

17. Bebek, G. Anti-tamper database research: Inference control techniques. Technical Report EECS 433 Final Report, pp. 23-29, Case Western Reserve University, November 2002.

18. W3C XML Encryption Working Group. Avaliable at http://www.w3.org/Encryption, 2006.

19. Stonebraker, M. The Design of the POSTGRES Storage System. In 13th VLDB Conference, pp. 1-50, 1987.

20. Myers, E. W. Efficient Applicative Data Types. In 11th ACM Symposium on Principles of Programming Languages, pp. 12-22, 1984.

21. Sarnak, N. and R. E. Tarjan. Planar Point Location Using Persistent Search Tree. Communications of the ACM, 29(7), pp. 34-43, July 1986.

22. Easton, M. C. Key-Sequence Data Sets on Indelible Storage. IBM Journal of Research and Development, pp. 101-120, May 1986.

23. Rathmann, P. Dynamic Data Structures on Optical Disks. In 1st International Conference on Data Engineering, pp. 1-30, 1984.

24. Denning, D. Cryptography and Data Security. Addison-Wesley, pp. 55-90, 1982.

25. Cody, Patrick M. Dynamic Security for Medical Record Sharing. Available at http://groups.csail.mit.edu/medg/projects/share/cody2003.pdf , 2006.

26. Office for Civil Rights, Department of Health and Human Services. Summary of the HIPAA Privacy Rule: HIPAA Compliance Assistance. GPO, 2003.

27. Berens, Steve. Storage down cold: DLTIce is a Compliant Electronic Storage Medium. Computer Technology Review, pp 120-141, July 2004.

28. Delshad, Daniel. Storage Strategies Meet Regulatory Burden on Data Retention. Enterprise Networks and Servers, pp. 12-23, November 2005.

29. Zambroski, Ray. Protecting the Digital Medical Record. Essential Security Software Articles, pp. 23-29, 2006.