An Efficient Approach to Support Querying Secure Outsourced XML Information*

Yin Yang, Wilfred Ng, Ho Lam Lau, and James Cheng

Department of Computer Science
Hong Kong University of Science and Technology
{yini, wilfred, lauhl, csjames}@cs.ust.hk

Abstract. Data security is well-recognized a vital issue in an information system that is supported in an outsource environment. However, most of conventional XML encryption proposals treat confidential parts of an XML document as whole blocks of text and apply encryption algorithms directly on them. As a result, queries involving the encrypted part cannot be efficiently processed. In order to address these problems, we propose XQEnc, a novel approach to support querying encrypted XML. XQEnc is based on two important techniques of vectorization and skeleton compression. Essentially, vectorization, which is a generalization of columns of a relational table, makes use the basic path of an XML tree to label the data values. Skeleton compression collapses the redundant paths into a multiplicity attribute. Our analysis and experimental study shows that XQEnc achieves both better query efficiency and more robust security compared with conventional methods. As an application, we show how XQEnc can be realized with relational techniques to enable secure XML data outsourcing.

1 Introduction

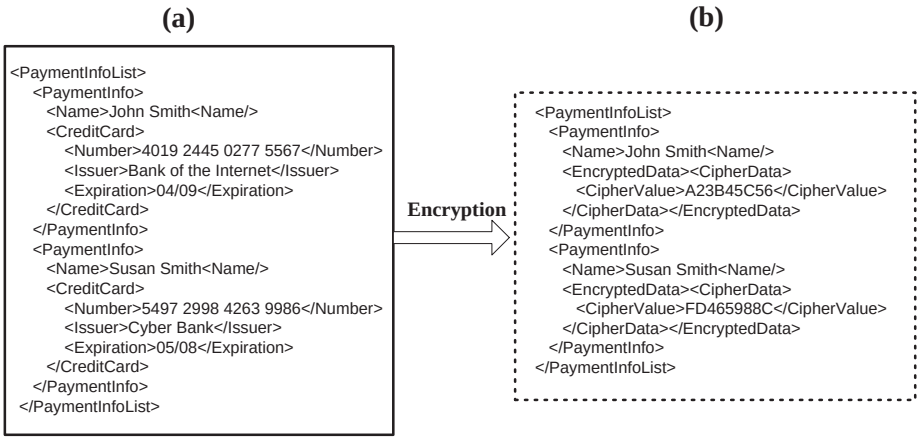
XML has emerged as a new standard for data representation and exchange on the Internet. As more data is expressed in XML, it is increasingly common to find sensitive information in XML, and thus security becomes an important issue. In order to avoid unauthorized access, the confidential parts of the XML document have to be protected. This can be done by *access control* mechanisms, e.g. security views [1] in the XML repository, or by applying encryption. In many cases, the access control components can be bypassed and encryption is a must. For instance, when transmitting data via an untrusted channel, and when the data is stored in vulnerable storage [2], e.g. the hard drive may be stolen.

The heterogeneous nature of XML data raises new requirements for encryption. Specifically, different parts of data need different treatments. Consider the running example of XML snippet given in Table 1(a) in which the details about the customer are confidential and thus must be encrypted, while the names of

* This work is partially supported by RGC CERG under grant number HKUST6185/03E.

the customers may be accessed by multiple parties and therefore should be kept in plain text. The proposal recommended by W3C [3] addresses this problem. Using the methods described in [3], only details about credit cards are encrypted, resulting in the XML code given in Table 1(b) (some details such as namespaces are omitted for the sake of simplicity in illustration).

Table 1. (a) A running example of XML snippet (b) the Encrypted XML snippet



In the above treated XML fragment shown in Table 1(b), the plain text segment from "<CreditCard>" to "</CreditCard>" is encrypted and replaced by an `EncryptedData` node. Note that the protected data is still treated as a whole block of text, and its internal structure is ignored. One problem is that the redundancy introduced by the XML format can be exploited to attack the encryption. For instance, the fact that the encrypted part always ends with the string "</CreditCard>" can be used for cryptanalysis. Another problem is that, since each confidential part is replaced by its corresponding encrypted block, the *context* around it may be exploited by the adversary. In our example, one can judge that the first encrypted block is the credit card information for John Smith. Besides, the fact that John Smith has some secrets (in this case a credit card) in this data file is exposed. When more items are encrypted together, the adversary is able to find out the rough number by judging the length of the cipher text. In other words, some *statistical information* may be exposed.

Apart from these security defects, treating the protected data as a whole inevitably incurs efficiency problems. Consider the following XPath [4] query:

```
//PaymentInfo[//Issuer = "Bank of the Internet"]/Name
```

Among various details of credit cards, only information about issuers is necessary to answer this query. However, since the entire block of sensitive information is encrypted, there is no way to extract the issuer alone from the encrypted

blocks. Consequently, a large amount of unnecessary decryption is performed, which may seriously slow down query processing.

Motivated by these security and efficiency drawbacks of existing solutions, we propose XQEnc, a novel XML encryption approach based on recent developments of XML repositories, namely, vectorization and skeleton compression [5], [6], [7]. We show experimentally that compared with existing methods, XQEnc has efficient query processing capability.

Importantly, XQEnc facilitates secure XML data outsourcing, which has not been discussed in the literature. In a nutshell, secure XML data outsourcing makes it possible for organizations to store confidential XML data on untrusted database servers and shift the query workload to the server as much as possible, without revealing the content of the data to the server. From business point of view, organizations following this paradigm are able to enjoy the benefit that their resources can be better invested in their core business but on the other hand, data management is supported by a dedicated service provider.

The rest of the paper is organized as follows: Section 2 surveys related work, focusing on existing XML security schemes and the theoretical foundation of our work: XML vectorization and skeleton compression. Section 3 presents XQEnc with analysis. Section 4 then describes how our XQEnc is able to employ relational technology to enable XML data outsourcing. Section 5 supports our security and efficiency claims by presenting a comprehensive experimental study. Finally, Section 6 concludes the paper with directions for future work.

2 Related Work and Preliminaries

In this section, we present the background of secure data outsourcing and the fundamentals of XML vectorization and skeleton compression.

2.1 XML Encryption

The most influential XML encryption method is the one officially recommended by W3C [3]. Essentially, its emphasis is on providing a mechanism such that different parts of the same document can get different treatments. One of the main virtues of this technique is in its flexibility. Since the encrypted document is still a valid XML document, an XML document can be encrypted for several times by different parties on different parts. An intuitive example of XML encryption using this method has been given in Section 1.

The problems of this approach, and also similar approaches, are apparent. Since the focus is on flexibility, rather than data security or query efficiency, naturally it does not satisfy the requirements of many applications where data security and query efficiency is highly important. In XQEnc, we try to match its flexibility, while at the same time we provide enhanced security and optimized query efficiency.

2.2 XML Vectorization and Skeleton Compression

One technique we adopt in XQEnc is called vectorization. XML vectorization generalizes the well-known technique of *vertical partitioning* in relational databases for optimizing query performance. An extreme form of vertical partitioning, which is called vectorization, is to store each column of a relational table separately. Vectorization means partitioning the document into path vectors in the context of XML. The result of partitioning outputs a sequence of data values appearing under all paths and bearing the same *path labeling*. Applying vectorization on the XML fragment presented in Table 1 of the running example, we obtain the set of vectors (PaymentInfoList is the root node) in Table 2:

Table 2. Path vectors in the "payment information list" document of Table 1

/PaymentInfoList/PaymentInfo/Name	[John Smith, Susan Smith]
/PaymentInfoList/PaymentInfo/CreditCard/Number	[4019 2445 0277 5567, 5497 2998 4263 9986]
/PaymentInfoList/PaymentInfo/CreditCard/Issuer	[Bank of the Internet, Cyber Bank]
/PaymentInfoList/PaymentInfo/CreditCard/Expiration	[04/09, 05/08]

Each of these vectors corresponds to a path of labels that leads to a nonempty text node. This technique has been employed in the recently proposed "semantic compressor" XMILL [8] to achieve optimal compression ratio of XML documents. As we will show in Section 3, XML vectorization can also be utilised to enhance security in addition to compressing XML data in XQEnc.

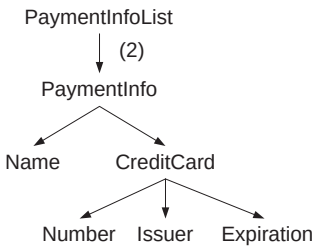


Fig. 1. The compressed skeleton of the running example

Another important technique called "skeleton compression" was originally proposed in [6] for supporting query processing of compressed XML documents. The main idea is to remove the redundancy contained in the XML document tree by sharing common sub branches and replacing identical and consecutive branches with one branch and a multiplicity annotation. The compressed skeleton of our running example in Table 1 is shown in Fig. 1. Note that the two PaymentInfo records are compressed into one branch and one multiplicity annotation, (2). According to the experimental results reported in [6], the compressed XML skeleton is small enough to fit well into main memory. This empirical fact motivates our proposed approach for query processing on outsourced XML data,

described in Section 4. Finally, it is worth mentioning that Buneman et al. [5] extend the skeleton compression technique to facilitate the processing of XQuery queries. However, to our knowledge there has been no attempt to apply the technique in querying encrypted XML data in literature.

2.3 Secure Data Outsourcing

Recently, the problem of secure data outsourcing (also referred to as "privacy preserving data outsourcing" in literature) has drawn considerable attention. In the data outsourcing paradigm proposed in [9], data owners store their data on rented servers, and query the server to get desired information. The database server is not trusted, and thus the data must be stored in encrypted form. Meanwhile, the server needs some information about the data (for example the use of "crypto-index" in [10]) in order to process queries. The result of a query is usually an encrypted superset of the actual result and is transferred to the client. There needs second processing on the (trusted) client side by decrypting the data and filtering out those that do not satisfy the query conditions. The goal is to shift query processing as much as possible to the server side while maintaining data security during processing and data transfer. Fig.2 illustrates a simplified

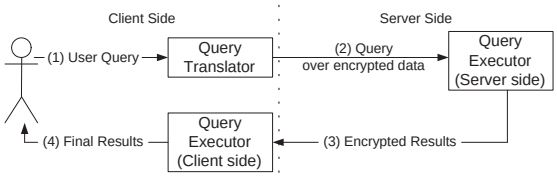


Fig. 2. A simplified architecture of a data outsourcing system

architecture of a typical data outsourcing system. A user query is translated by the query translator into two sub-queries: a query over encrypted data, which is executed at the server side with the help of the crypto-index, and a "filtering" query executed at the client side to select the real answer to the query from the temporary results returned by the server. The temporary results returned by the server are encrypted tuples, and the client needs to do decryption first in order to perform the filtering step.

There are different proposals for defining the crypto-index that provides the helper information for the server to process the queries. The original proposal in [9] is to first partition the entire data space into disjoint buckets, and then stores the bucket IDs on the server. During query processing, the values in the queries are translated into their corresponding bucket IDs. This method reveals information (i.e. bucket IDs) of the original data and the server returns a super set of the actual results. A more efficient way is to use the order preserving encryption algorithm proposed in [2], which guarantees no information leakage and optimal communication overhead. All these proposals, however, consider only the data in relational setting. These proposals are not applicable in XML setting, since the internal structure of the sensitive nodes can also be confidential in XML,

thus simply substituting values by crypto-indices may reveal the structural information to the (untrusted) server. In our running example, storing the sensitive credit card details with only values encrypted (e.g. substituted by bucket IDs) exposes the internal structure of the CreditCard node. Therefore, existing methods designed for querying encrypted relational data are not appropriate for XML data outsourcing. This motivates our development of XQEnc.

3 XQEnc: Queriable XML Encryption

Let d denote the plain text XML document to be treated. The goal of XQEnc is to transform d into another XML document d^T such that confidential information is protected and queries and are efficiently processed. For ease of presentation, we first simply assume that all the data contained in d is confidential, and describe the basic ideas of XQEnc in Section 3.1. In Section 3.2 we tackle the more general case in which only some parts of d need to be encrypted. The security and efficiency of XQEnc is analyzed in Section 3.3.

3.1 Basic Ideas of XQEnc

We first illustrate the basic idea of XQEnc by assuming that all textual and structural information in the plain text document d is confidential and needs to be encrypted. In our running example, this means not only the details of credit cards, but also other document information such as names and document structures needs to be encrypted. Hence, the resulting document d^T has only one text node, containing the cipher text of the original document.

```
<EncryptedData><CipherData>
        <CipherValue>2313FB3D980A0</CipherValue>
</CipherData></EncryptedData>
```

Note that this transformation fully complies with the W3C XML encryption standard. The crucial part in the transformation is how to generate the cipher value based on the original document. Traditional methods simply treat the original document as a whole piece of text and apply an encryption algorithm like triple DES on the text. The drawbacks of this methodology have been discussed in Section 1. The basic idea of XQEnc is that we first compute the compressed skeleton and the corresponding set of data vectors, and then encrypt these two entities separately.

In our implementation, XQEnc adopts the approach based on vectorization and skeleton compression for building a structural index called Structure Index Tree (or SIT). The SIT helps to remove the redundant, duplicate structures in an XML document. An example of a SIT is shown in Fig. 4(b), which is the index of the tree in Fig. 4(a), the structure of the sample XML extract in Fig. 3 modelled as a tree. Note that the duplicate structures in Fig. 4(a) are eliminated in the SIT shown in Fig. 4(b). In fact, large portions of the structure of most XML documents are redundant and can be eliminated. For example, if an XML document contains 1000 repetitions of our sample XML extract (with different

data contents), the corresponding tree modelling its structure will be 1000 times bigger than the tree in Fig. 4(a). However, the structure of its index tree will essentially have the same structure as the one in Fig. 4(b), implying that the search space for query evaluation is reduced 1000 times by the index.

```
1. <open_auctions>                  7.  </bid>                      13. <open_auction id = "open2"> 19.    <date> 11/29/2002 </date>
2.   <open_auction id = "open1">    8.  <bid>                       14.   <initial> $500.00 </initial>  20.    <increase> $0.50 </increase>
3.   <initial> $12.00 </initial>    9.    <date> 12/03/2000 </date> 15. </open_auction>                 21.  </bid>
4.   <bid>                          10.   <increase> $1.50 </increase> 16. <open_auction id = "open3"> 22. </open_auction>
5.     <date> 12/02/2000 </date>    11.  </bid>                     17.   <initial> $1.50 </initial>   23.</open_auctions>
6.     <increase> $2.00 </increase> 12. </open_auction>             18.   <bid>
```

Fig. 3. A simple Auction XML Extract

Our implementation avoids full decryption by grouping the data (i.e. v.ext in Fig. 4(b)) into many small blocks. We utilize the index to evaluate queries on the encrypted XML data. The novelty is that we apply an encryption algorithm (like triple DES) to encrypt each data block in XQEnc. After that, the encrypted blocks are combined to form the cipher value for the original document. Query processing in XQEnc requires that we first decrypt the relevant encrypted data blocks necessary to answer the query. Our design is not only compatible with compression on the data blocks but also supports a fine-grained encryption as will be discussed later. An immediate benefit of using SIT as

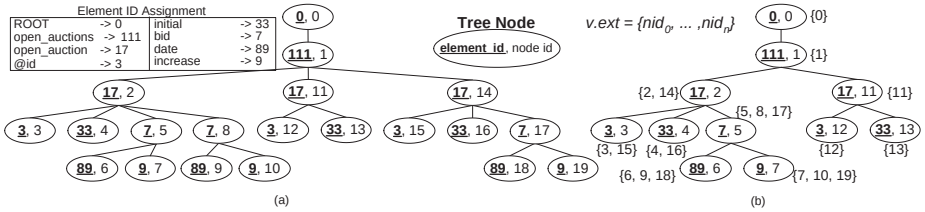


Fig. 4. (a) A simple Auction XML Extract Structure Tree (contents of the exts not shown) of the Auction XML Extract and (b) its corresponding SIT

indexing in XQEnc is that we are able to compress and decrypt the blocks at the same time. During query processing, a retrieved data block is first decrypted and then decompressed. It is clear that there is an overhead of doing compression/decompression, but the overall performance may be better, since compression removes redundancy in the data, the cost of doing encryption/decryption is also reduced. However, a more detailed study of the problem of compression and encryption interaction is not the scope in this paper.

3.2 The General Case in XQEnc

Because of the heterogeneous nature of XML data, in most cases only parts of the original text document d need to be encrypted. These confidential parts may scatter all over d, with or without clear patterns. An intuitive way is to apply the method described in the previous section on each confidential part of the

document separately, which complies with the W3C standard. However, there are several drawbacks with this approach as we have illustrated using our running example in Table 1. For example, if we apply XQEnc on the two blocks of credit card information separately, and replace them with their corresponding cipher text, the resulting XML document d^T is similar to the transformed document presented in Section 1, except that the cipher values are generated using XQEnc. The security concern remains that the context can be exploited to attack the encryption and derive statistical information. Moreover, in this example and many other cases, the confidential parts have exactly the same internal structure and the same compressed skeleton is kept multiple times. Meanwhile, the data vectors for a single confidential part are often not large enough to fill a data block, which seriously affects storage utilization and query efficiency.

Rather than encrypting each confidential part individually, XQEnc puts them together and produces one single piece of cipher text, inserted as the last child of the root node. Using our running example, XQEnc generates the result as shown in the following transformed document:

```
<PaymentInfoList>
   <PaymentInfo>
      <Name>John Smith<Name/>
   </PaymentInfo>
   <PaymentInfo>
      <Name>Susan Smith<Name/>
   </PaymentInfo>
   <EncryptedData><CipherData>
      <CipherValue>E7FDA243B745CC586</CipherValue>
   </CipherData></EncryptedData>
</PaymentInfoList>
```

The cipher value consists of the following two components: the compressed skeleton of the *original* document d, and the confidential data partitioned in vectors, both components are in the encrypted form. Keeping the compressed skeleton of d ensures no loss of structural information. The compressed skeleton of a document is usually very small [6], which is much less than 1 megabytes for a document as large as hundreds of megabytes, or much less than 1% of the document size. This memory requirement is not demanding even in light-weight computing devices, given the trend of RAM size has been increasing as technology advances. Furthermore, an alternative is to keep only the "partial" compressed skeleton that is relevant to the confidential data. This makes the cipher value even shorter, at the cost of more complicated query processing. In our current design and implementation version of XQEnc, we adopt the former method (keeping the whole compressed skeleton) though we need to point out that a comparison with the latter is an interesting future work.

For the data vectors, we only include the confidential data, together with their document positions, in the cipher value. To answer an XPath query, first the compressed skeleton of the original document is decrypted from the cipher value. Then, the query processing algorithm of XQEnc described in the previous subsection is executed, treating the unencrypted part of the document and the

cipher value as two data sources. When the query processor needs the textual information of a non-confidential text node, it gets that from the plain text part of the document. XQEnc partitions the data vectors into blocks and encrypts each block individually. During query processing the minimum unit of data retrieval is a data block. This technical detail is omitted for the ease of presentation.

In our implementation the unencrypted part is first parsed during preprocessing and the value of a text node can be easily retrieved. When the information contained in a confidential text node is needed, the query processor extracts from the encrypted data vectors according to the document position of the text node . In our running example, the cipher text contains the compressed skeleton shown in Fig. 1, and the last three data vectors in Table 2. During query processing, the names of the card holders are retrieved from the plain text part while confidential details like the issuer of the credit cards are retrieved from the encrypted data vectors.

3.3 Discussion

Using the algorithms described in Section 3.2, there is only one piece of cipher text no matter how many confidential text nodes are scattered through the document, and the cipher text is always appended at the end of the document and affects only one block. This drastically reduces the concern that the context can be exploited to attack the encryption and derive sensitive information. Furthermore, the redundancy of the XML format is eliminated by vectorization, making it even harder to attack the encryption.

The query efficiency of XQEnc can be substantially improved, since rather than retrieving and decrypting the entire confidential part, XQEnc only accesses the data necessary to answer the XPath query, thus the overhead of data retrieval and decryption are reduced to a minimum. As shown in Section 5, XQEnc improves query efficiency by more than an order of magnitude.

4 Secure XML Data Outsourcing Using XQEnc

As discussed in Section 2.3, existing techniques for data outsourcing, which address mainly relational data, can not be applied directly to XML data, in which structural information can be confidential. In this section we propose our solution based on XQEnc, with analysis.

4.1 Assumptions and Setting

We first make several assumptions about the data to be outsourced. First, we assume that the outsourced data is expressed in XML format, and is to be stored in a rented server running a relational database system. This is practical because XML provides flexibility for data expression, while relational database systems are ubiquitous. The main reason for this assumption is that we want to utilize existing data outsourcing techniques by transforming XML data to relational

data. Note that this transformation must not expose the internal structure of the XML data to the server. Therefore, existing XML-to-relational transformation methods, e.g. [11, 12], cannot be applied. Second, for the sake of presentation simplicity we make the assumption that the entire XML data to be outsourced is confidential. The general case that only some parts of the data is confidential can be handled similarly by applying the techniques presented in Section 3.2. For queries, we limit our scope to answering XPath queries. Essentially, the XPath queries on the original document are translated to SQL queries on the transformed relational data by the query translator. Therefore, to handle the more general XQuery queries requires a modified query translator, which is left as future work.

Existing data outsourcing techniques mostly translate a user query on the original data to exactly one query on the encrypted data stored on the server side. In our method the query translator may translate one XPath query into a corresponding set of multiple SQL queries. This is natural because an XPath query can be very complicated and even not expressible in one single SQL query. Moreover, we may need to process the answer returned by the server in order to issue the next SQL query. Therefore, there are interactions between the query translator and the query processor at the client side.

Another issue is that data outsourcing requires stronger security than the core problem of encrypting data. This is because the server owns the knowledge of not only the encrypted data, but also the translated queries. Therefore the security requirement here is that the server cannot derive confidential information, including both textual and structural information, from the encrypted data and all the SQL queries it receives.

4.2 The Solution Based on XQEnc

The solution consists of two parts. The first part is the method to transform a given XML document d to relational data to be stored on the server. The second part is the query answering process. We describe them in sequel.

In order to transform the given document d to relational data, we first compute the compressed skeleton and data vectors of d, as in XQEnc. We denote the compressed skeleton by s, which is small even for very large XML documents as discussed. One key point of our design is that s is not stored on the server; rather, it is stored inside the query translator on the client as metadata. This means that it is impossible for the server to obtain any structural information of d, which is contained in s. Next, we need to transform the data vectors into relational data, and the security requirements in this step is reduced to ensuring confidentiality of textual information.

For each item i in the data vectors, we create a tuple $< V_i, P_i, T_i >$, where V_i is the vector ID that identifies the vector containing i; P_i is the document position of i, and T_i is the textual value of i. This step essentially transforms the data vectors into one single table, which has three columns V (the vector IDs), P (the document positions) and T (the textual data). The primary key of this relation is denoted as the pair $< V, P >$. This transformation is information

preserving, in the sense that the original data vectors can be restored using the resulting tuples.

After transforming the data vectors to relational data, the last step is to transform the relational data using existing data outsourcing techniques. Specifically, each tuple $< V_i, P_i, T_i >$ is transformed to another tuple $<$ *etuple*, V_i^S, P_i^S, $T_i^S >$, where *etuple* is the encrypted tuple, and X^S is the crypto-index of attribute X. Depending on the data outsourcing techniques used, the crypto-indices can be either bucket IDs using the bucketization technique [9], or encrypted values using the order preserving encryption technique [2].

A potential optimization during this step is to build multiple relations rather than one relation. The problem with cramming everything into one relation is that data in different vectors may have different ranges and distribution, which makes it difficult to compute a good bucketization or order preserving encryption scheme. In order to solve this problem, we devise an optimized cluster to group the vectors according to their sizes and characteristics of their textual values, and establish one relation for each cluster of vectors.

Finally, we support query processing as follows. We run the XQEnc query processing algorithm at the client side, treating the server as an external storage. A query is issued to the server whenever we need to access an item in one of the data vectors. Specifically, when we need the textual value of a data item in vector v and document position p, the following SQL query is sent to the Oracle database server.

$$\text{SELECT } \textit{etuple} \text{ FROM } R(v) \text{ WHERE } V^S = \textit{crypto} - \textit{index}(v)$$

$$\text{AND } P^S = \textit{crypto} - \textit{index}(p)$$

The result returned from the server is then decrypted and the textual data is used for further processing. In addition, when the path predicate contains a condition specifying the range or the textual values, e.g. *[issuer = "Bank of the Internet"]* in our running example, a corresponding selection condition is appended in the WHERE clause of the SQL statement sent to the server. In this example, the condition $T^S =$ *crypto-index("Bank of the Internet")* is appended to the SQL query to further reduce the amount of data transmitted from the server to the client.

4.3 Analysis

We now justify that our solution preserves data confidentiality, i.e. both textual and structural information is protected from unauthorized access by the server. First of all, the data stored on the server side does not contain any structural information, and thus the structural information is protected. This is because all structural information is contained in the compressed skeleton s, which is stored only on the client side and not accessible by the server. A single SQL query issued to the server does not contain any structural information either. The only concern is that by analyzing a *sequence* of queries, the server may derive some pieces of structural information. This can be further avoided by processing a set

of multiple XPath queries at the same time, and the client mixes together their translated SQL queries sent to the server.

The confidentiality of textual information is guaranteed by the traditional data outsourcing techniques, e.g. bucketization [9] or order preserving encryption [2]. These techniques are the state of the art techniques for relational data. In other words, that is the best possible robust scheme we can use to protect the textual information. For this reason, we claim that *the structural and textual information is best possible protected in XQEnc.*

Regarding query efficiency, one might think that a possible weakness of our approach is that we need the client to perform the second query processing, in addition to the first processing on the server. With a critical observation of the various factors affecting query efficiency, we argue that our solution is still efficient, despite of having this weakness. The bottleneck of the entire data outsourcing architecture is data transmission between the client and the server. This overhead in XQEnc is reduced to a minimum because the query processing algorithm only retrieves data necessary to answer the query. This also means the decryption work done at the client side is reduced to a minimum, which compensates the computation cost of running the XQEnc query processing algorithm. In general, *all data intensive work is reduced to a minimum on the client, while the amount of undesirable overhead imposed by running the XQEnc query processing algorithm is purely determined by the size of the XPath query.* Therefore, our solution based on XQEnc provides very competitive query efficiency as also supported by the experimental study next section.

5 Experiments

In this section we evaluate the query efficiency of XQEnc through experiments. We have implemented a prototype for XQEnc. All the experiments were run on the Windows XP Professional platform. The CPU was a 1.5 GHz Pentium 4, while the system had 512MB of physical memory. For system parameters, the block size of data vectors is the maximum of 2 megabytes and 1000 data items, which is the empirical optimal block size obtained by the experimental study of [7]. The encryption algorithm chosen is DES.

We carried out the experiments on five different real XML datasets, all of which are well established benchmarks for studying XML query processing algorithms. The sizes of these data sets are listed in Table 3. For more details of the datasets, the readers may refer to [13] describing these datasets.

Table 3. Five data sets used in the experiments

Dataset	DBLP	SwissProt	LineItem	TreeBank	Shakespeare
Size (MB)	127	109	30.7	82	7.4

In order to evaluate the query performance of XQEnc, we need to make practical assumptions about which parts of the XML documents are consid-

ered confidential, as well as to choose several representative queries involving confidential parts of the document. Due to limited space we describe the experimental settings and results in detail for the DBLP dataset. The settings for other datasets are listed in Appendix A.

The document structure for the DBLP dataset is relatively simple. It is basically a fact sheet of various publications. Since the focus here is to test the efficiency of XQEnc, there should be a large part of the document considered confidential. In our experiments we assume all the "inproceedings" nodes, both the internal structures and textual values are confidential. In addition, we make further assumptions that all document URLs, and theses not in public domain are confidential for copyright reasons. The queries used in the experiments are listed below:

```
(Q1) /dblp/inproceedings/title
(Q2) //masterthesis/author
(Q3) //article[year = "2002"]/url
(Q4) //inproceedings[booktitle = "DASFAA"]/url
(Q5) //inproceedings[author = "Wilfred Ng"]/title
```

The query Q1 is to show that the major factor of query performance is the size of the result, and the most time consuming operation is decryption. Because there is a large number of records for conference papers, the result for Q1 is very large, while the query itself is relatively simple to parse and process. Query Q3 involves both confidential and non-confidential information, and Q4 and Q5 contain highly selective predicates. The conventional method needs to retrieve and decrypt lots of unnecessary data and thus should be much slower than XQEnc, which only retrieves the data needed to answer the query.

We report several aspects of the efficiency of XQEnc. First, we compare the time needed to encrypt the confidential part, using both conventional methods (i.e. treating each part as a whole piece of text) and XQEnc. Second, we show the time needed for decrypting the entire document. In the extreme case, everything in the document is confidential and this time is the lower bound for a conventional method to answer most XPath queries. Third, we report the response time for processing the queries, both the conventional method and XQEnc.

Table 4. Experimental results. Here X/C means the ratio of response time by XQEnc to that by Conventional method. The response times are rounded to the nearest second

Dataset	Encryption time (X/C)	Decryption time (C)	Q1 (X/C)	Q2 (X/C)	Q3 (X/C)	Q4 (X/C)	Q5 (X/C)
DBLP	50/40	38	10/30	1/23	5/23	1/30	1/30
SwissProt	49/35	33	8/30	2/28	2/28	1/28	2/28
LineItem	13/11	10	2/8	1/7	1/8	1/7	1/7
TreeBank	49/27	25	8/18	8/17	8/17	9/17	8/17
Shakespeare	4/3	2	1/2	1/2	1/2	1/2	1/2

The experimental results are shown in Table 4. All the numbers in the table are response times measured in seconds. For encryption and query response time, we give both the time needed for XQEnc and the conventional method, in the

shown query order. In general, the encryption cost of datasets in XQEnc is larger than the conventional method but still within an acceptable range as it is not frequent. The response time shows that XQEnc is very competitive in answering queries. Another interesting fact is that decrypting the entire dataset is very expensive. Therefore, when the entire dataset is confidential, XQEnc is more than an order of magnitude faster than conventional methods.

The results in the DBLP dataset clearly show that the response time for Q1 is relatively greater than other queries, thought Q1 itself is simple. This can be explained by the fact that the results for Q1 is much larger, and thus decryption becomes the major cost of query processing and XQEnc enjoys less cost-saving benefit as expected. However, XQEnc is much faster for queries Q4 and Q5, which justifies our efficiency analysis.

6 Conclusions

We propose XQEnc, which is a novel XML encryption technique based on XML vectorization and skeleton compression techniques. The technique is useful to support query processing of XML information in an outsourcing environment. Compared with existing solutions, XQEnc provides strengthened security and efficient query processing capability. The resulting document after applying XQEnc complies with the W3C encryption standard, which allows different treatments to be applied on different parts of the XML Document. The techniques can be used together with the existing compression technologies to reduce the data exchange overhead in network.

Throughout, we explain and show how secure XML data outsourcing can be achieved using XQEnc and existing relational data outsourcing techniques. Our solution guarantees robust protection for structural and textual information. Importantly, we demonstrate with a spectrum of XML benchmark datasets that the query performance of our solution is very competitive. Specifically, all data intensive computation and transmissions are reduced to a minimum.

XQEnc gives rise to many interesting topics for future work. At the current stage our design and implementation of XQEnc focus on XPath query support. We plan to extend our solution to more general query languages. For example, Schema-based (e.g. XSchema) validation on encrypted XML data is another promising subject to further study. We also plan to compare our approach with other approaches beside [3]. The analytical study of the relationship between the block parameters, query workload and the encryption efficiency underpinning the cost model is also important to further optimise query processing in XQEnc.

References

1. Fan, W., Chan, C., Garofalakis, M.: Secure XML querying with security views. In: SIGMOD Conference. (2004) 587–598
2. Agrawal, R., Kiernan, J., Srikant, R., Xu., Y.: Order preserving encryption for numeric data. In: SIGMOD Conference. (2004) 563–574
3. Imamura, T., Dillaway, B., Simon, E.: XML encryption syntax and processing. W3C Recommendation (2002)

4. Clark, J., DeRose, S.: XML path language (XPath). W3C Working Draft (1999)
5. Buneman, P., Choi, B., Fan, W., Hutchison, R., Mann, R., Viglas, S.: Vectorizing and querying large XML repositories. In: ICDE. (2005) 261–272
6. Buneman, P., Grohe, M., Koch, C.: Path queries on compressed XML. In: VLDB. (2003) 141–152
7. Cheng, J., Ng, W.: XQzip: Querying compressed XML using structural indexing. In: EDBT. (2004) 219–236
8. Liefke, H., Suciu, D.: XMILL: An efficient compressor for XML data. In: SIGMOD Conference. (2000) 153–164
9. Hacigumus, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: SIGMOD Conference. (2002) 216–227
10. Hore, B., Mehrotra, S., Tsudik, G.: A privacy preserving index for range queries. In: VLDB. (2004) 720–731
11. Bohannon, P., Freire, J., Roy, P., Simeon, J.: From XML schema to relations: A cost-based approach to XML storage. In: ICDE. (2002) 64–76
12. Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D.J., Naughton, J.F.: Relational databases for querying XML documents: Limitations and opportunities. In: VLDB. (1999) 302–314
13. Miklau, G.: The XML data repository. http://www.cs.washington.edu /research /xmldatasets/ (2006)

A Appendix: The Datasets for the Experiments

```
Dataset: SwissProt
Confidential Parts: all entries whose class is "standard" and mtype is "PRT"
Q1: //Species
Q2: //Ref[DB = "MEDLINE"]
Q3: //Features[//DOMAIN/Descr = "HYDROPHOBIC"]
Q4: //Entry[AC = "Q43495"]
Q5: //Entry[//Keyword = "Germination"]
Dataset: LineItem
Confidential Parts: all lines whose order key value is between 10000 and 40000
Q1: //table/T/L_TAX
Q2: /table/T[L_TAX = "0.02"]
Q3: /table/T[L_TAX[[. >= "0.02"]]]
Q4: //T[L_ORDERKEY = "100"]
Q5: //L_ DISCOUNT
Dataset: TreeBank
Confidential Parts: everything enclosed by <_QUOTE_> tags
Q1: //_QUOTE_//_NONE_
Q2: //_QUOTE_//_BACKQUOTES_
Q3: //_QUOTE_//NP[_NONE_ = "FTTVhQZv7pnPMt+EeoeOSx"]
Q4: //_QUOTE_//SBAR//VP/VBG
Q5: //_QUOTE_//NP/PRP_DOLLAR_
Dataset: Shakespeare
Confidential Parts: all speeches
Q1: //SPEAKER
Q2: //PLAY//SCENE//STAGEDIR
Q3: //SPEECH[SPEAKER = "PHILO"]/LINE
Q4: //SCENE/SPEECH/LINE
Q5: //SCENE[TITLE="SCENE II. Rome. The house of EPIDUS"]/LINE
```