A Survey on Querying Encrypted XML Documents for Databases as a Service

Ozan Ünay
Boğaziçi University
ozan.unay@boun.edu.tr

Taflan İ.Gündem
Boğaziçi University
gundem@boun.edu.tr

ABSTRACT

"Database as a service" paradigm has gained a lot of interest in recent years. This has raised questions about the security of data in the servers. Firms outsourcing their XML databases to untrusted parties started to look for new ways to securely store data and efficiently query them. In this paper, encrypted XML documents, their crypto index structures and query processing using these structures are investigated. A comparison of various algorithms in the literature is given.

Categories and Subject Descriptors
A.1 [Introductory and Survey]

General Terms
Querying Encrypted XML Document Algorithms

Keywords
Encryption, XML, Database as a Service

1. INTRODUCTION

Recently a popular trend in business is "to concentrate on your own business and outsource the rest". This trend is also valid in information technology. Firms outsource their software or databases. Outsourcing software is known as "*software as a service*" and outsourcing the database is referred to as "*database as a service*" [10]. Firms using databases as a service outsource database management tasks such as back up, restore, availability and space management [5, 12]. Outsourcing a database provides the advantage of having reliable storage of large volumes of data, efficient query processing, and most importantly savings on the database administration cost for the data owner. On the other hand, some questions arise about the security of data due to the fact that firms share private or confidential information with third parties. This is not risky if the service providers are trusted. But what if they are not?

In recent years another popular trend is using XML databases. XML has already become a standard for exchanging data and storing semi structured data [7]. A lot of firms started to store their data in XML. It is increasingly becoming common to find sensitive information in XML [21]. Sensitive information can either be confidential (e.g. for a bank it is important to hide credit card information of their customers) or private (e.g. for a hospital it is important not to disclose its patients' diseases).

As a result it is important to secure XML data for most firms that use third parties for database outsourcing. The data have to be kept securely and should be visible neither to attackers nor to database service providers. One of the solutions to secure data in XML is using "*access control mechanisms*" which are out of scope of this survey. Using access control mechanisms alone may not be sufficient. The attackers who break into the system may gain access to private information. Either the communication channel or the storage itself may be insecure, e.g. the hard drive may be stolen. Thus, something more than an access control mechanism is needed. *Encryption* plays a key role at this point. In order for encryption to be reliable, the encryption key should only be known by the data owner. The database should be a black box for the service provider. At this point a serious question comes to mind. How will the service provider answer the user queries without knowing the database content? Some research has been done on this subject. This survey tries to summarize the work done in the literature about encrypted XML query processing. It compares the strengths and weaknesses of the various approaches and classifies them according to their properties.

The rest of the paper is organized as follows. In Section 2, brief information is given on encrypted query processing and encrypted XML query processing. In Section 3, classification of existing methods according to their index structures is given. Section 4 has the conclusions and some possible future research suggestions on the subject.

2. PRELIMINARIES

Research on database encryption started with key management [8]. Later on techniques have been developed to efficiently search keywords based on encrypted textual strings by Song, Wagner and Perrig [4]. Independent of the database type (relational, XML or text file) the naïve way of encrypted query processing is sending encrypted database totally to the data owner [12]. In such a case, the service provider does not serve as a query engine and the query processing responsibility is at the data owner side. This may be acceptable for only small volumes of data. Other problems with this approach are expensive cost of data transportation due to limited bandwidth and decryption and query processing of the whole database at the client side that may have limited processing capability. In [11] a novel bucketization and partitioning structure is proposed which influenced many of the papers in literature. An algebraic framework for query rewriting over encrypted attributes is described. The main idea is to map the plaintext values to ciphertext values by splitting the plaintexts in the domain into some partitions and giving them bucket ids. The success of this methodology is due to the mapping function of the bucket ids that uses order preserving encryption functions [16]. As a result the range queries can successfully be supported. In [9] mathematically well defined order and distance preserving encryption functions are used rather than partitioning techniques to encrypt the database. The proposed computing architecture is efficient in the sense that for some query types query processing can be completed at the server without having to decrypt the database. One future work proposed in [9] was to handle SQL queries with arithmetic expressions and aggregate functions as well as complex SQL queries with nested subqueries. This is accomplished in [18]. In [18] the authors present query execution strategies for the mentioned types of queries. They also quantify additional costs incurred in executing these queries. In [6] a hash based method suitable for selection queries is given. The index is maintained at the server side. The algorithm given in [6] provides a balance between efficiency and security. In [1] an algorithm for determining optimal bucket size for encrypted query processing is proposed.

2.1 General Architecture of Encrypted Query Processing in XML

To speed up query processing most of the work load should be at the service provider which usually has more processing capabilities (e.g. better CPU) and more resources (e.g. memory) than the client. However since the service provider doesn't have the decryption key, some clues for answering queries should be given to the service provider. These clues should be just enough for service provider to return the encrypted tuples but not sufficient to retrieve the structure (schema) or the content (instance) of the XML document. These clues are usually given by maintaining crypto - indexes on either the service provider or the data owner side. The general architecture of encrypted query processing is as follows. The user creates a query which is then translated into its encrypted form by the query translator at the client side. The rules of encryption are determined by the client and given to the query translator. After the query becomes secure enough not to show the structure of the XML database, the service provider answers the query by some predefined rules that are at the server side. The result set returned by the service provider is not the exact result set that the user wants. It is a superset of the actual result set. The client decrypts the results and post filters the results in order to get the actual result set.

The client should have some processing capability in order to post process the results. The main purpose of encrypted XML query processing is to increase the work done by the service provider and decrease the work done by the client.

Some papers in literature mention architectures different from the one explained in the preceding paragraphs. For example in SemCrypt project (that will be summarized later) a number of messages should be exchanged between the server and the client in order to get the results.

2.2 W3C Encryption Standard

W3C has proposed standards for XML encryption [19]. The details of XML and its encryption can be found out in [19, 7, and 20]. According to the mentioned standards, the tags and the contents that are going to be encrypted are replaced with a string called the Encrypted Data element. There are 4 sub elements of Encrypted Data. (a) *Encryption method* indicates the encryption algorithm and the parameters of the specified algorithm. (b) *Key Info* indicates the key name but not the value. (c) *Cipher Data* contains cipher value as sub element which indicates the encrypted element together with its content. (d) *Encryption properties* contain additional information related to generation of Encrypted Data.

2.3 Attack Types

There are many specific attack types in cryptanalysis. The fundamental categories of attack types may be summarized as follows.

Brute force attacks: In this type of attack, the attacker tries every key until the correct key is reached to break the encryption.

Cipher text only attacks: In this attack type, it is assumed that the attacker has access to the encrypted message only and does not know what the original plaintext is.

Known plaintext attacks: In this attack type, the attacker has samples of both the plaintext and its encrypted version (cipher text) and makes use of them to obtain the key.

Chosen plaintext attacks: In this attack type, it is assumed that the attacker chooses an arbitrary piece of plaintext and is able to find the corresponding cipher text.

Adaptive chosen plaintext attacks: In this attack type, it is assumed that the attacker chooses a piece of plaintext and is able to determine the corresponding cipher text iteratively making use of previous results.

Chosen cipher text attacks: In this attack type, it is assumed that the attacker chooses an arbitrary piece of cipher text and is able to find the corresponding plaintext.

In the papers examined in our survey, also the following specific attack types are explicitly stated and used [3].

Frequency based attack: If the attacker can find a match between the cipher text and the plain text values, then it is possible for the attacker to determine the algorithm and the key used in the encryption. This may be possible by knowing the exact frequency of domain values (e.g. suppose that Johnny White has won 10 prizes and there is only one value in the encrypted database that occurs 10 times. The attacker can infer that Johnny corresponds to that encrypted value), or by knowing the query workload (e.g. suppose that, for an e-product catalog, it is known that the main query asked is [book/ author/ [year=2007]], then the attacker can guess which encrypted tag corresponds to which plaintext tag).

Size-based attack: If the length of the plain text determines the length of the cipher text, the attacker may eliminate the candidate databases whose lengths do not match. This type of attack is referred to as size based attack.

3. INDEX TYPES

There are basically two types of index structures used in encrypted XML documents. One of them is the structural index and the other one is the value index. Purpose of the structural index is to determine whether the path in the query matches any of the paths in the XML documents. Purpose of the *value Index* is to check the constraints in range queries. These indexes can be maintained either at the server side or client side.

3.1 Maintaining Indexes at the Server

There is a well known index structure in unencrypted XML documents. In this index structure every tag is given a sequence number starting from 1 and incremented by 1. The sequence number of the opening tag of a node represents the left bound of the node and the sequence number of the closing tag represents the right bound of the node. This enumeration brings up a general rule that states "for a parent node p and child node c, p.leftbound < c.leftbound and p.rightbound > c.rightbound". Table 1 (b) gives an example of this index.

Table 1. (a) Sample XML document (b) and its unencrypted Index

(a)

```
<Bib>
  <Book>
    <Title>Spring</Title>
    <Author>
      <Name>F.WELL</Name>
      <Education>
        <BS>X School</BS>
      <Education>
    <Author>
  </Book>
  <Book>
    <Title>Football</Title>
    <Author>
      <Name>J.HAND</Name>
      <Education>
        <MS>X School</MS>
      <Education>
    <Author>
  </Book>
</Bib>
```

(b)

Node name	LB	RB
Bib	1	26
Book	2	13
Title	3	4
Author	5	12
Name	6	7
Education	8	11
BS	9	10
Book	14	25
Title	15	16
Author	17	24
Name	18	19
Education	20	23
MS	21	22

LB : Left Bound
RB : Right Bound

In order not to disclose the hierarchical structure of the XML document, the schema just explained is modified and is called *discontinuous structural index* (*DSI*) in [12]. In DSI, the interval [0, 1] is assigned to the root. The children are assigned sub intervals of the parent's interval. The intervals of the children are determined by an algorithm at run time. The general rule still holds; for a parent p and a child c, p.leftbound < c.leftbound and p.rightbound > c.rightbound. Table 2 illustrates DSI for the XML document in Table 1(a). DSI hides the structure of the XML document from the server.

Two tables are used for the structural index at the server side in [12]. One of them is the encryption block table and the other one is the DSI table. The structures of these tables are given in Table 3. DSI table holds the tags in one column and the corresponding intervals in the other column. Only confidential tags are encrypted. This provides efficient query processing on nodes which are unencrypted.

Table 2. Representation of the modified schema in [12] for the XML Document in Table 1 (a).

Node name	left Bound	right Bound
Bib	0	1
Book	0.12	0.56
Title	0.23	0.28
Author	0.34	0.54
…	…	…

Table 3. Representation of Structural Index tables for the sample XML document in Table 1 (a).

Encryption Block Table

ID	Interval
1	[0.23,0.28]
2	[0.34,0.54]

DSI Table

Tag	DSI
Bib	[0,1]
Book	[0.12,0.56]
UXM45	[0.23,0.28]
WRETS	[0.34,0.54]

In [12] the value index has order preserving encryption with splitting and scaling (OPES). The value index is maintained at the server side to support range queries. Splitting and scaling is used to prevent frequency based attacks. By using splitting, each plaintext value is encrypted into one or more ciphertext values. As a result an unencrypted word is represented by different encrypted words. Scaling is done after splitting. By using scaling, target domain size is multiplied. Number of occurrences of encrypted words is multiplied by a scale factor. Main purpose of splitting and scaling is to change frequency distribution of encrypted data values in the value index so that they are different from the frequencies of the original values.

Query processing in [12] is as follows. When a query is submitted to the server, the query translator at the client transforms the query into encrypted form. The query translator replaces every tag with the corresponding encrypted tags in the structural index. The DSI of the tags in the query are found from the DSI table. These intervals are used to find out the bucket ids in the encryption block table. The bucket ids returned are the results of the structural index processing. In the second phase the client translates the value-based constraints in the query. Server finds out the bucket ids satisfying the value index. Finally the server intersects the bucket ids returned from the structural index and the value index. The result of the intersection is sent to the client for further decrypting.

The main contribution of the approach in [12] is allowing the execution of range queries at the server side by employing order preserving encryption with splitting and scaling. The proposed value and structural indexes are provably secure. Sensitive structural information and value associations are hidden from attackers who possess exact knowledge of domain values and their occurrence frequencies. Splitting and scaling used in this paper make the encrypted values in the database nearly uniformly distributed. Thus it prevents an attacker from making a statistical analysis. Since value and structural indexes are maintained at the server side, burden of query processing is mainly at the server side. In the proposed approach, the client should have a query translator and also a simple query engine in order to post filter the results after decrypting. One of the limitations of OPES is that security achieved by scaling encrypted data causes an increase in data size. Increase in data size implies extra time in query processing. Another limitation of the approach in [12] is that it can not provide security against prior knowledge of tag distribution, query workload distribution and correlation among data values. Also this approach is not very efficient in insertions and updates.

Query processing takes place in three phases in [14] as shown in Figure 1. The first phase is the query preparation phase which is offline. This phase contains encoding the structure and the instance of the XML document. In this phase, to encode the structure of the XML document all the paths are extracted from the encrypted XML document. Each node is converted to a value using a predefined rule (e.g. take the first n characters of a node) and a hash function. Then each path is converted to a value using the values of the nodes. Values of paths which have different lengths are stored in different hash tables. To encode the instance of the XML document all the attribute and value pairs are encoded and stored in a hash table. Details of hashing and encoding can be found in [14], but mainly a function called Base26ValueOf ("string") is used that calculates the Base26 of a number. To support range queries the authors use the bucketization technique that we explained in Section 2.

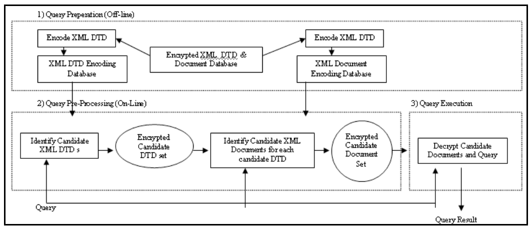
1) Query Preperation (Off-line)
Encode XML DTD
XML DTD Encoding Database
Encrypted XML DTD & Document Database
Encode XML DTD
XML Document Encoding Database
2) Query Pre-Processing (On-Line)
Identify Candidate XML DTD s
Encrypted Candidate DTD set
Identify Candidate XML Documents for each candidate DTD
Encrypted Candidate Document Set
3) Query Execution
Decrypt Candidate Documents and Query
Query
Query Result

Figure 1. Framework for querying encrypted data in [14]

The second phase is the query preprocessing phase. It is the first online phase. In this phase inappropriate XML document candidates are filtered by examining query conditions. In the third phase the selected candidate databases are returned to the client for further decrypting.

The main contribution of the approach in [14] is using hashing techniques to compute encodings. The encodings use order preserving encryption functions so that range queries are successfully supported. In [14] indexes are maintained at the server side so that most of the query processing can be done at the server side. Security of this approach is directly related to the security of the hashing function used.

Another approach that uses indexing at the server is given in [17]. Main contribution of the approach given in [17] is that it introduces powerful encryption primitives. These encryption primitives help clients specify a rich class of security policies for XML data. It is possible to selectively hide sensitive data by using these primitives. There are mainly three encryption primitives proposed. E_V (encrypt value) primitive encrypts a subtree and replaces it by an encrypted node. The subtree rooted at node "Author" (shown in Figure 2) is encrypted and replaced with an encrypted node which is shown on the right in Figure 3. E_T (encrypt tag) primitive encrypts just the tags of the subtree rooted at node n (including the tag of node n). E_S (encrypt structure) primitive hides the relationship between two specified nodes. When E_S primitive is applied to the relationship between "Book" and "Author" in Figure 2, the relationship becomes hidden as shown on the left in Figure 3. The encrypted XML storage model proposed takes as input the XML schema of the unencrypted node and three encryption primitives and outputs a server side XML representation. $E_{V,}$ E_S and E_T are applied sequentially in the given order.

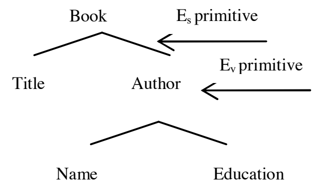

Figure 2. A tree representation of an XML document with encryption primitives E_s and E_v to be applied

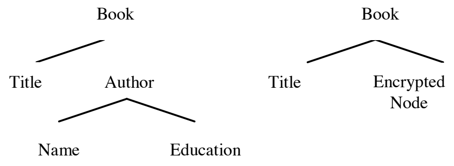

Figure 3. Affect of applying E_v and E_s on the document given in Figure 2 shown on the left and right hand side, respectively.

Another contribution of [17] is proposing a multidimensional partitioning strategy. The information stored at the server is viewed as an N-dimensional space. This N-dimensional space is partitioned into a set of partitions. Each partition is given a random identifier. The partitions cover the whole domain and do not overlap. Equi-width partitioning is used when partitioning the domain which helps prevent frequency based attacks. Multidimensional partitioning strategy overcomes the security limitations of single dimensional techniques. In [17] majority of the query processing is done at the server side. Another advantage of the proposed schema is that it allows range queries to be processed at the server side.

In [13] authors use query aware decryption. According to the proposed schema in [13] a relational index file is maintained at the server side which consists of three columns. The first column is "*key name*" column which holds the keys. The second column is "*element type*" column which holds the XML tags. The third column is the "*occurrences*" column which holds the Dewey numbers of elements in "*element type*" column. All three fields are encrypted using the keys in "*key name*" column. For the sample XML document in Table 1 (a) Dewey numbering schema is given in Figure 4 and the resulting encrypted XML document's tree representation is given in Figure 5.

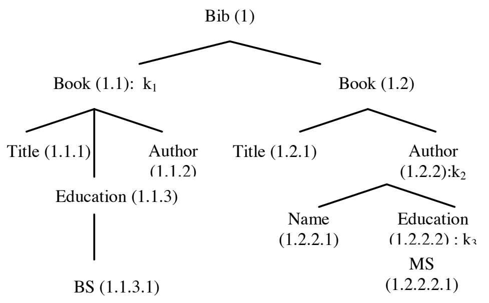

Figure 4. Dewey numbering schema for the sample document in Table 1 (a).

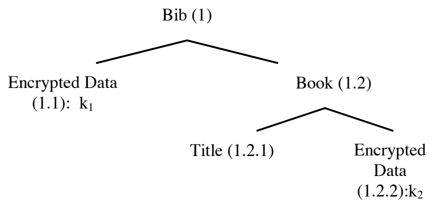

Figure 5. Encrypted XML data for the sample document in Table 1 (a).

The index file proposed in [17] (maintained at the server side) for the sample document in Table 1 (a) is given in Table 4.

Table 1. Encrypted XML Index in [17] for the sample document in Table 1 (a).

Key Name	Element Type	Occurrences
Null	Bib	1
Null	Book	1.2
Null	Title	1.2.1
k_1	Book	1.1
k_1	Author	1.1.2
k_1	Education	1.1.3
k_1	BS	1.1.3.1
k_2	Author	1.2.2
k_2	Name	1.2.2.1
k_2, k_3	Education	1.2.2.2
k_2, k_3	MS	1.2.2.2.1

Query processing in [13] is as follows. Suppose that a user who has keys k_1, k_2 and k_3 sends the query "//book//BS" on the sample XML data in Table 1 (a) to the server. The query processor first decrypts the "*key name*" field using the keys k_1, k_2 and k_3. It then decrypts the "*element type*" field using $\{k_1\}$, $\{k_2\}$ and $\{k_2, k_3\}$. Then the processor decrypts the "*occurrences*" field of the row associated with element type "*BS*" which is asked in the query. Element type "*BS*" is located at the node with Dewey number 1.1.3.1 in Figure 4 and "*Encrypted data*" element is located at the node with Dewey number 1.1 in Figure 5. As we understand from its Dewey number, "*BS*" is in "*Encrypted data*" node with Dewey number 1.1 in Figure 5. Thus "*Encrypted data*" node with Dewey number 1.1 is decrypted. However "*Encrypted data*" node with Dewey number 1.2.2 is not decrypted. Thereby unnecessary decryption is avoided.

The main contribution of [13] is to process only the encrypted blocks that contribute to the result. Although the proposed schema is efficient and provides a way to query encrypted XML documents, it has some flows in security. During query processing, keys are disclosed to the server. Also the proposed schema is open to frequency analysis. Another limitation of the paper is that it does not allow range queries to be executed without decrypting the encrypted block.

3.2 Maintaining Indexes at the Client

In [21] XQEnc is used for encrypted XML query processing. XQEnc uses vectorization and skeleton compression [2, 3]. In *vectorization*, an XML document is partitioned into path vectors which are composed of nonempty leaf nodes. In *skeleton compression,* redundancy of XML documents is removed by using common sub branch sharing. The identical and consecutive branches are replaced with one branch along with a multiplicity annotation. By this the XML document becomes much smaller. The experiments in [21] show that XML documents become small enough to fit into the main memory. In XQEnc approach, for each XML document, a compressed skeleton *S* is computed and stored at the client side and a set of corresponding data vectors *D* is computed and stored at the server side. In order to access D efficiently, a Structural Index Tree (SIT) is constructed at the server side. S is never shared with the server. Consequently, the structure of the XML document is hidden from the third parties.

For each item i in D a triple $<V_i, P_i, T_i>$ is created. V_i represents the vector ID, Pi represents the document position and Ti represents the textual value of i. Then each triple is transformed into the following representation; $<$etuple, Vi^c, Pi^c, $Ti^c>$, where etuple is the encrypted tuple and the other entries are the corresponding crypto indexes of the original triple. According to XQEnc, crypto indexes can either be bucket ids [11] or the encrypted values using order preserving encryption [16]. XQEnc algorithm runs at the client side. This algorithm generates the following query and then sends it to the server.

SELECT etuple FROM R (V) WHERE Vs = cryptoindex (v) AND Ps = cryptoindex (p) AND Ts = crptoindex (“Any string”)

The server is treated only as an external storage. The server starts its job after the client sends the query. The server retrieves the encrypted result and sends it back to the client for further decrypting.

The main contribution of the approach in [21] is storing the schema of the XML document as a compressed skeleton at the client making it inaccessible to the server. In this manner the structural information is hidden from the server. XQEnc may support range queries if order preserving encryption is used instead of bucketization technique as the crypto-indices. For queries containing highly selective predicates, XQEnc is very efficient since it only retrieves the necessary data for the client to decrypt. In [21] the burden of the query processing is at the client side which decreases the performance. The client needs to maintain indexes at its side and in the distributed environment. This means that every insertion into the XML database should trigger the client side for an index update. There is also the possibility of a problem with space management in [21]. Although it is claimed that the skeleton compression makes a document much smaller than the original one, there may still be a problem if the client has limited memory and/or the document is big and irregularly structured.

3.3 A Different Approach: Usage of Nonces

In [15] encrypted query processing is managed by both maintaining indexes at the server side and the client side. We investigate this approach under a different heading because it uses a novel approach. Suppose person A is communicating with person B. A uses key k and the encryption function E in order to encrypt plaintext p and get ciphertext c.

c = E (p, k)   p = D (c, k).

Person A sends c to person B. Person B decrypts the ciphertext c using key k and the decryption function D. The problem in this schema is that p is always encrypted as c. Consequently intruders can make frequency based attacks. To prevent intrusion, p is encrypted using k and a number called nonce which is used only once. Now the schema becomes as follows.

c = E (p, k, n)   p = D (c, k, n)

The nonce used is send to person B together with message p. By doing so every plaintext p is encrypted as ciphertext c1, c2 and so on.

Let’s turn back to our discussion of encrypted XML query processing. In [15] the schema of the XML document is stored at the client side. The paths are stored with their unique identifiers which are called path schema IDs (Table 5). The * indicates that there can be one or more nodes with the same tag name. Using * makes the schema document small so that the client can store it.

Table 2. XML Schema(Stored at the client side)

Path Schema ID	Path Schema
PS1	Bib/Book*/Author
PS2	Bib/Book*/Title
PS3	Bib/Book*/Author/Name
PS4	Bib/Book*/Author/Education
…	…

At the server side there are two hash tables. First hash table (Table 6 (a)) uses path instances as key and the second one (Table 6 (b)) uses path values as key.

Table 3. Hash Tables at the server side.
(a) Table used by GetValueForPathInstance function

Cryptographic Hash(PI)	E(value, k, nonce)	Nonce
H(PS2-1)	E(Spring,k,10)	10
H(PS3-1)	E(F.WELL,k,11)	11
H(PS3-2)	E(J.HAND,k,12)	12
H(PS2-2)	E(Football,k,13)	13
…	…	…

(b) Table used by GetPathInstanceForValue function

Cryptographic Hash (PS-V)	E(PI*, k , nonce)	Nonce
H(PS2-Spring)	E({1},k,21)	21
H(PS3-F.WELL)	E({1},k,22)	22
H(PS3-J.HAND)	E({2},k,23)	23
H(PS2-Football)	E({2},k,24)	24

Query processing in SemCrypt project is as follows. Suppose that the client wants to submit a query “/book [title=’spring’]/author/name”. The client first looks up the schema stored at its side. The client finds out that the path schema id of bib/book*/title is PS2. The client computes the cryptographic hash function H(PS2-spring). Then the client revokes the function getPathInstancesForValue with parameter H(PS2-spring). The value returned from the server is E({1},k,21). The client decrypts this answer using the nonce together with the key and finds out that the answer is at first instance ({1}) of the book in the XML schema. Then the client filters the title path and adds the author/name path to the query. The client knows that author/name path is PS3. Now the resulting query becomes bib/book[1]/author/name which is PS3-1. The client revokes the function GetValueforPathInstance with parameter H (PS3-

1)). Finally the server returns the encrypted value together with its nonce. E(F.WELL, k, 11). The client decrypts this answer by using nonce 11 and the key and finds out the answer F.WELL.

The main contribution of the approach in [15] is that it introduces an encryption technique based on using nonces. Usage of nonces prevents frequency based attacks since the same plaintexts are encrypted as different ciphertexts. One of the drawbacks of the approach in [15] is that it requires multiple rounds of communication between the server and the client which consumes bandwidth and increases the query processing time. Another limitation of this approach is that it does not allow range queries to be executed. It is good only for selection queries. It is also important to mention that the clients should have considerable query processing capability because they continuously process the encrypted results and compute hash functions. Thus the burden of query processing is divided between the server and the client.

4. SUGGESTIONS FOR FUTURE WORK

Existing methods mostly concentrate on retrieval in indexing structures in encrypted query processing. Management of indexes is usually not taken into account. There should be efficient mechanisms to handle updates efficiently in index structures. This is important especially in XML documents which are frequently updated. Most of the papers (with few exceptions) in the literature propose index structures that are applicable to all attributes of the XML documents. The mechanisms that allow users to build indexes only on specific attributes of the encrypted XML document should be improved. Another improvement can be supporting regular expression queries. In order to answer a [a-z] b we need 26 queries (one query for each character in the alphabet) for encrypted XML documents. A good indexing mechanism and a query processor in the future may handle this kind of regular expression queries. Since encrypted XML query processing is a time consuming job, distributed and parallel servers may need to be devised. Multiple computation nodes may significantly improve the performance of query evaluation. Another important future work would be making an inference control analysis of each proposed approach to measure how secure they are as far as inference is concerned. An example of this would be [9] which contains a detailed inference control analysis of the paper's own approach. In general a well defined measure of security is needed for most of the techniques in the literature to show how secure they are.

5. REFERENCES

[1] B.Hore, S.Mehrotra, G.Tsudik. Privacy Preserving Index for Range Queries. *Proceedings of the 30th VLDB Conference, 2004. Toronto, Canada*

[2] Buneman, P., Choi, B., Fan, W., Hutchison, R., Mann, R., Viglas, S.Vectorizing and querying large XML repositories.*21st International Conference on Data Engineering. April 5, 8 261–272*

[3]Cheng, J., Ng, W.: XQzip: Querying compressed XML using structural indexing. *9th International Conference on Extending Database Technology March 14, 18. 2004. 219–236*

[4] D.X. Song, D.Wagner, and A.Perrig. Practical techniques for searches on encrypted data. *In Proc. of the 2000 IEEE Symposium on Security and Privacy, p: 44-55, Oakland, CA, USA, May 2000.*

[5] E. Mykletun and G. Tsudik, On using Secure Hardware in Outsourced Databases. *International Workshop on Innovative Architecture for Future Generation High Performance Processors and Systems January 2005*

[6]E.Damiani, S.Jajodia Balancing confidentiality and efficiency in Untrusted Relational DBMSs. *CCS'03 October 27–30, 2003, Washington, USA.*

[7] Extensible Markup Language, XML 1.0 http://www.w3.org/TR/REC-xml, October 2000

[8] G.I. Davida, D.L. Wells, and J.B. Kam. A database encryption system with subkeys. *ACM Transactions on Database Systems, 6(2)p:312-328, June 1981.*

[9] G.Ozsoyoglu, D.Singer, S.Chung. Anti-tamper databases: Querying Encrypted Databases *In Proc. of the 17th Annual IFIP WG 11.3 Working Conferece on Database Applications and Security, August 2003.*

[10] H.Hacigumus, S.Mehrotra, and B.Iyer. Providing Database as a Service. *Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, p: 29-40, 2002.*

[11]H.Hacigümüs, B.Iyer, C.Li, and S.Mehrotr. Executing SQL over encrypted data in the database-service-provider model. *In Proc. of the ACM SIGMOD'2002,Madison,Wisconsin,USA June 2002.*

[12] H.Wang, L.Lakshmanan.Efficient Secure Query Evaluation over Encrypted XML Databases. *32nd International Conference on Very Large Data Bases, 2006 September 12-15.*

[13] J. Lee , K. Whang. Secure query processing against encrypted XML data using Query-Aware

Decryption. *Elsevier, Information Sciences. 2006 p:1928–1947*

[14] L. Feng and W. Jonker. Efficient Processing of Secured XML Metadata. *OTM Workshops 2003 p: 704-717*

[15] M.Schrefl, K.Grun, J. Dorn. SemCrypt – Ensuring Privacy of Electronic Documents through Semantic-Based Encrypted Query Processing. *21st International Conference on Data Engineering Workshops. April 5, 8 p: 1191*

[16] R.Agrawal, J.Kiernan ,R. Srikant, Y. Xu Order preserving encryption. *SIGMOD 2004 June 13-18, Paris, France*

[17] R.C.Jammalamadaka, S.Mehrotra. Querying Encrypted XML documents. *IDEAS'06*

[18] Sun.S.Chung, G.Ozsoygolu. Anti-tamper databases: Processing Aggregate Queries over Encrypted Databases *In Proc. of the 22nd International Conference on Data Engineering Workshops, ICDEW '06.*

[19] T. Imamura, B. Dillaway, E.Simon, XML Encryption Syntax and Processing, *W3C Recommendation, December 2002.* http://www.w3.org/TR/xmlenc-core/ March 2002.

[20]XML Encryption Requirements, http://www.w3.org/TR/xml-encryption-req ,March 2002.

[21] Y.Yang, W.Ng, H.L.Lau, and J.Cheng. An Efficient Approach to Support Querying Secure Outsourced XML Information *CAiSE 2006, LNCS 4001, p:157–171, 2006.*