Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences 176 (2006) 1928–1947
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Secure query processing against encrypted XML data using Query-Aware Decryption

Jae-Gil Lee *, Kyu-Young Whang

Department of Computer Science and Advanced Information Technology Research Center (AITrc), Korea Advanced Institute of Science and Technology (KAIST), 373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea

Received 25 October 2004; received in revised form 1 June 2005; accepted 2 August 2005

Abstract

Dissemination of XML data on the internet could breach the privacy of data providers unless access to the disseminated XML data is carefully controlled. Recently, the methods using encryption have been proposed for such access control. However, in these methods, the performance of processing queries has not been addressed. A query processor cannot identify the contents of encrypted XML data unless the data are decrypted. This limitation incurs overhead of decrypting the parts of the XML data that would not contribute to the query result. In this paper, we propose the notion of Query-Aware Decryption for efficient processing of queries against encrypted XML data. *Query-Aware Decryption* allows us to decrypt only those parts that would contribute to the query result. For this purpose, we disseminate an encrypted XML index along with the encrypted XML data. This index, when decrypted, informs us where the query results are located in the encrypted XML data, thus preventing unnecessary decryption for other parts of the data. Since the size of this index is much smaller than that of the encrypted XML data, the cost of decrypting this index is negligible compared with that for unnecessary decryption of the data itself. The experimental results show that our

* Corresponding author. Tel.: +82 42 869 5562; fax: +82 42 867 3562.
E-mail addresses: jglee@mozart.kaist.ac.kr (J.-G. Lee), kywhang@mozart.kaist.ac.kr (K.-Y. Whang).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2005.08.001

method improves the performance of query processing by up to six times compared with those of existing methods. Finally, we formally prove that dissemination of the encrypted XML index does not compromise security.

Keywords: XML databases; Query processing; Database privacy

1. Introduction

As XML becomes a standard for disseminating data on the internet, applications disseminating XML data on the internet are rapidly increasing. In particular, dissemination of XML data occurs frequently in Peer-to-Peer (P2P) [15] and Web Service [6] applications. In these applications, dissemination of XML data could breach the privacy of data providers unless access to the disseminated XML data is carefully controlled. Hence, access control for disseminated XML data becomes an important research issue [4,13].

Bertino and Ferrari [4] and Miklau and Suciu [13] have proposed access control methods for disseminated XML data. These methods encrypt each part that needs to be access-controlled using a different key, and then, disseminate encrypted XML data along with the keys. Access control on the disseminated XML data can be done by allowing the users to decrypt only those parts for which they have the keys.

These methods, however, have not discussed the performance of processing queries against the encrypted XML data using the keys. To protect the disseminated data from misuse, it is preferable to store the disseminated data in the encrypted form rather than in the decrypted form [1,17]. Thus, a query processor cannot identify the contents of encrypted XML data until decrypting them. This limitation incurs overhead of decrypting the parts of the XML data that would not contribute to the query result.

Example 1. Suppose we encrypt XML data as shown in Fig. 1(a) and disseminate the encrypted data. An encrypted element and its descendants are replaced with an `EncryptedData` element [11]. k_1, k_2, and k_3, which are shaded, represent the keys used for encryption. Then, suppose a user who has the keys k_2 and k_3 executes a query `/subjects/subject/analysis/tests/HIV` against the encrypted XML data in Fig. 1(a). Two consecutive decryptions shown in Fig. 1(b) do not contribute to the query result. This unnecessary decryption degrades the performance of processing queries.

In this paper, we propose the notion of Query-Aware Decryption for efficient processing of queries against the encrypted XML data. *Query-Aware Decryption* allows us to decrypt only those parts that would contribute to

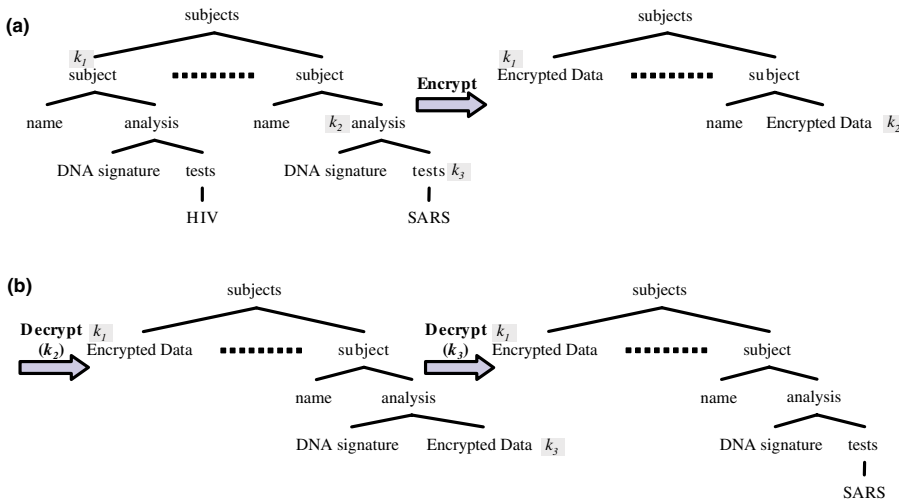


Fig. 1. An example showing unnecessary decryption. (a) An example of encrypting XML data. (b) An example of decrypting XML data.

the query result. For this purpose, we disseminate an encrypted XML index along with the encrypted XML data. This index, when decrypted, informs us where the query results are located in the encrypted XML data, thus preventing unnecessary decryption for other parts of the data. Due to the small size of this index (typically, less than 1/20 of the size of the encrypted XML data), the cost of decrypting this index is negligible compared with that for unnecessary decryption of the data itself.

The scope of this paper is to facilitate efficient query processing on encrypted XML data, which plays a crucial role in secure dissemination of XML data. Typically, a user executes queries to access encrypted XML data, where query processing tends to be time consuming due to decryption which is a very costly operation. Other issues related to secure dissemination of XML data—including specification of access control policies, encryption of XML data using the minimum number of keys, and dissemination of encrypted XML data and keys—are beyond of the scope of this paper.

In summary, the contributions of this paper are as follows:

- We propose the notion of Query-Aware Decryption for preventing unnecessary decryption and a method for performing Query-Aware Decryption using the encrypted XML index.
- We formally prove that dissemination of the encrypted XML index does not compromise security.

- We demonstrate, by extensive experiments, that our method significantly outperforms existing methods.

The rest of this paper is organized as follows. Section 2 describes previous work on encrypting XML data for access control on disseminated data. Section 3 proposes the notion and the mechanism of Query-Aware Decryption. Section 4 proves that our method does not compromise security. Section 5 presents the results of performance evaluation. Finally, Section 6 concludes the paper.

2. Related work

In this section, we summarize existing work on the encryption of the XML data. Section 2.1 explains the XML encryption standard [11] proposed by W3C. Section 2.2 explains access control methods [4,13] using encryption for disseminated XML data.

2.1. XML encryption standard

W3C has proposed the XML encryption standard [11]. According to this standard, encrypted data retain the XML syntax, and an element name and element body is replaced with an encrypted string called the `EncryptedData` element. This element has four subelements: `EncryptionMethod`, `KeyInfo`, `CipherData`, and `EncryptionProperties`. `EncryptionMethod` represents an algorithm and the parameters used for encryption/decryption. `KeyInfo` represents a key name used for encryption/decryption (it does not contain the key value). `CipherData` has `CipherValue` as the subelement, and `CipherValue` represents a string generated by encrypting an element name and element body. `EncryptionProperties` provides additional information related to the generation of `EncryptedData`. Reencryption of encrypted data is allowed, and is called *super-encryption*.

Example 2. Fig. 2(b) shows the XML data generated by encrypting the first `subject` element in Fig. 2(a). We note that the first `subject` element and its subelements are replaced with the `EncryptedData` element.

2.2. Access control methods using XML encryption

Bertino and Ferrari [4] and Miklau and Suciu [13] have proposed access control methods using encryption for disseminated XML data. These two methods are essentially identical except for the specification of access control policies. To specify access control policies, Bertino and Ferrari use an authorization model, while Miklau and Suciu use an extension of the XQuery [5] language.

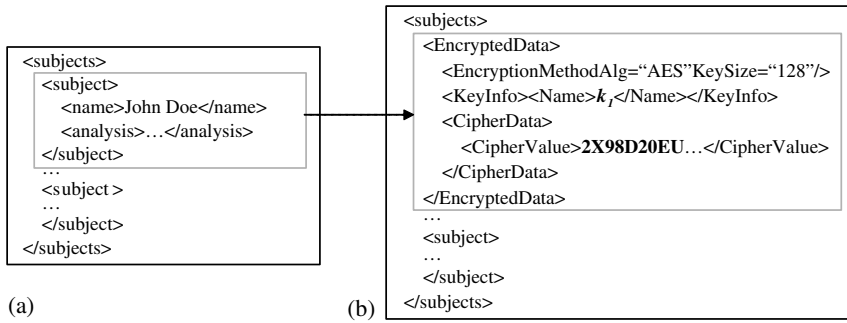


Fig. 2. An example of XML encryption [11]. (a) XML data before encryption. (b) XML data after encryption.

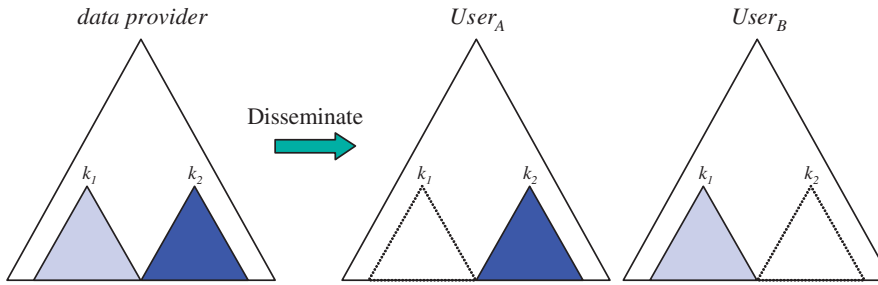


Fig. 3. The methods by Bertino and Ferrari [4] and Miklau and Suciu [13].

Fig. 3 shows the concept of these methods. After encrypting two different parts using the keys k_1 and k_2, respectively, the encrypted XML data are disseminated with the key k_1 to $User_A$ and with the key k_2 to $User_B$. The data accessible by $User_A$ (or $User_B$), which are unshaded, are what can be decrypted using the key k_1 (or k_2) possessed by $User_A$ (or $User_B$) as well as the data that have not been encrypted at all.

These methods, however, have not addressed the issue of query processing performance. Miklau and Suciu [13] have proposed a query processing method that first decrypts the `EncryptedData`'s that have been encrypted using the keys possessed, and then, checks whether an element decrypted matches that specified in the query. However, these methods can incur unnecessary decryption as discussed before. In this paper, we solve this problem by using the notion of Query-Aware Decryption.

3. Query-Aware Decryption

In this section, we propose the notion of Query-Aware Decryption. Section 3.1 first defines Query-Aware Decryption. Section 3.2 describes the encrypted

XML index devised for Query-Aware Decryption. Section 3.3 describes a query processing algorithm that uses Query-Aware Decryption.

3.1. Problem definition

We first define the *RelevantEncryption* function in Definition 1, and then, Query-Aware Decryption in Definition 2 using this function.

Definition 1. *RelevantEncryption* is a function for checking whether an encrypted element is relevant to the query result and is defined as follows. Here, *ED* is an `EncryptedData` element, *KEYS* is the set of keys that the user possesses, and *Q* is an XPath [9] expression.

$$RelevantEncryption(ED, KEYS, Q) = \begin{cases} true & \text{if } ED, \text{when decrypted using } KEYS, \text{ contains part of the results of } Q \\ false & \text{otherwise} \end{cases}$$

Definition 2. *Query-Aware Decryption* is decrypting only the `EncryptedData` elements that satisfy $RelevantEncryption(ED, KEYS, Q) = true$.

3.2. Encrypted XML index

Query-Aware Decryption requires an efficient method for evaluating the $RelevantEncryption(ED, KEYS, Q)$ function. To help evaluate the $RelevantEncryption(ED, KEYS, Q)$ function, we disseminate the information that summarizes the structure of the XML data. This summary information enables us to know the locations of the parts of the XML data that contribute to the query result, thus informing us whether an `EncryptedData` element contains the query result. We also encrypt this summary information itself in the same manner as we do XML data.

In this paper, we adopt an inverted index [12] as the structure for the summary information since the inverted index has been widely used to index XML data [18].

Fig. 4 shows the structure of the XML index used in this paper. This index is analogous to the conventional inverted index in that element types correspond to keywords in a text document, and the occurrences of each element type correspond to the posting list.[1] Each entry in the XML index represents (1) a set of key names, (2) an element type, and (3) the occurrences of the element type in

[1] The posting list [12] is a component of the inverted index. It is a list of entries that indicate which documents contain a given term, and how often the term appears in the document.

Key Name	Element Type	Occurrences
$\{k_1\}$	*analysis*	dewey numbers of *analysis* element instances
▪▪▪▪	▪▪▪▪	▪▪▪▪

Fig. 4. The structure of the XML index.

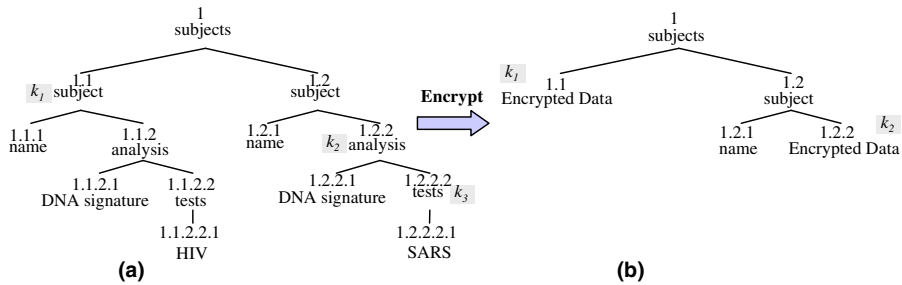
(2) whose instances are encrypted using the set of key names in (1). The Key Name field of the index entries pointing to the instances that are not encrypted becomes *null*. The Key Name, Element Type, and Occurrences fields themselves in the XML index are also encrypted using the keys in the Key Name field except when the Key Name field is *null*. Hence, a user who possesses the set of keys *KEYS* is capable of decrypting only the index entries whose Key Name field is a subset of *KEYS*.

We use the Dewey number [8] for representing the occurrences. The Dewey number has a good property that the Dewey number representing the occurrences remain unchanged when a subtree is replaced with an `EncryptedData` element. We call this property the *subtree-replacement invariant*. The Dewey number represents the position of an element instance in hierarchical data and is constructed as follows [8]: (1) The number of numeric values separated by a dot is the level of an instance; (2) The numeric values are assigned consecutively among siblings. For example, the Dewey number 1.3.2 represents the instance, at level 3, that can be reached by traversing from the root to its third child and to the second child of this third child. By this definition, if a subtree rooted at the Dewey number $i,\ldots,j$ is replaced with an instance, the instances having the Dewey number $i,\ldots,j,\ldots$ disappear, but the Dewey numbers of other instances remain unchanged. Hence, the Dewey number satisfies the subtree-replacement invariant.

Example 3. Suppose we encrypt the `subject` element instance numbered 1.1 using the key k_1, the `analysis` element instance numbered 1.2.2 using the key k_2, and the `tests` element instance numbered 1.2.2.2 using the key k_3. Fig. 5(b) and (c) show the encrypted XML data and the encrypted XML index, respectively. The Key Name field of the index entries pointing to the element instances numbered 1.2.2.2 or 1.2.2.2.1 contains $\{k_2, k_3\}$ because these two instances are encrypted first using the key k_3 and then using the key k_2. The shading in Fig. 5(c) means that the values have been encrypted.

3.3. Query processing algorithm using Query-Aware Decryption

We present Lemma 1 for evaluating the *RelevantEncryption* $(ED, KEYS, Q)$ function using the encrypted XML index.



Key Name	Element Type	Occurrences	
null	*subjects*	{1}	
null	*subject*	{1.2}	
null	*name*	{1.2.1}	
{k_1}	*subject*	{1.1}	Encrypted using k_1
{k_1}	*name*	{1.1.1}	
{k_1}	*analysis*	{1.1.2}	
{k_1}	*DNA signature*	{1.1.2.1}	
{k_1}	*tests*	{1.1.2.2}	
{k_1}	*HIV*	{1.1.2.2.1}	
{k_2}	*analysis*	{1.2.2}	Encrypted using k_2
{k_2}	*DNA signature*	{1.2.2.1}	
{k_2, k_3}	*tests*	{1.2.2.2}	Encrypted using k_2 and k_3
{k_2, k_3}	*SARS*	{1.2.2.2.1}	

(c)

Fig. 5. An example of the encrypted XML index. (a) Original XML data. (b) Encrypted XML data. (c) An encrypted XML index.

Lemma 1. *Consider the index entries whose Key Name field is a subset of the set of keys KEYS that the user possesses and whose Element Type field is in the set of the element types of the results of a query Q. Suppose $\mathscr{R}$ is a set of the Dewey numbers stored in the Occurrences fields of those index entries; i.e., these Dewey numbers represent the instances that can potentially be the query result. e is the Dewey number of an* `EncryptedData` *element ED. Then,* $\{r | r \in \mathscr{R}$, *and e is a prefix of* $r\} \neq \emptyset \Rightarrow$ *RelevantEncryption* $(ED, KEYS, Q) =$ *true.*

Proof. By the definition of the Dewey number, the element a is an ancestor of the element d if the Dewey number $dewey_a$ of a is a prefix of the Dewey number $dewey_d$ of d. Besides, an `EncryptedData` element contains encrypted descendant elements [11]. Thus, if the Dewey number e of an `EncryptedData` element ED is a prefix of the Dewey number $r \in \mathscr{R}$, ED is an ancestor of the instance pointed by r and contains that instance encrypted. □

Fig. 6 shows a query processing algorithm that uses Query-Aware Decryption. We call it *Query-Aware Decryption*. Query-Aware Decryption consists of four steps. In the first step, the algorithm decrypts the Key Name field of the encrypted XML index using the set of keys *KEYS*. We note that the Key Name field containing a subset of *KEYS* can be decrypted. In the second step, the algorithm

Algorithm *Query-Aware Decryption*
Input: (1) the encrypted XML data, (2) the encrypted XML index,
(3) a query ***Q***, (4) a set of keys ***KEYS*** that the user possesses
Output: query results
1 Decrypt the "Key Name" field of the encrypted XML index using the set of keys ***KEYS***.
2 Decrypt the "Element Type" field of the index entries, using the keys in the "Key Name" field, whose "Key Name" field is decrypted.
3 Decrypt the "Occurrences" field of the index entries, using the keys in the "Key Name" field, whose "Element Type" field $\in$ { the element types of the query result }.
4 Execute the query ***Q*** against the encrypted XML data.
4.1 Search for the elements satisfying the query ***Q*** against the decrypted(or unencrypted) XML data.
4.2 If encountering an `EncryptedData` element ***ED***, evaluate *RelevantEncryption*(***ED***,***KEYS***,***Q***) by checking whether the Dewey number of ***ED*** is a prefix of the Dewey number stored in the "Occurrences" field decrypted in step 3.
If *RelevantEncryption*(***ED***,***KEYS***,***Q***) = *true*, then decrypt ***ED***. Otherwise skip ***ED***.
4.3 Repeat steps 4.1 and 4.2 until retrieving all the query results.

Fig. 6. The query processing algorithm *Query-Aware Decryption*.

decrypts the Element Type field of the index entries, using the keys appearing in the Key Name field, whose Key Name field is decrypted. In the third step, the algorithm identifies the index entries whose Element Type field is in the set of the element types of the query result, and then, decrypts the Occurrences field of only those index entries using the keys appearing in the Key Name field. The third step is for finding out the Dewey numbers of the instances that can potentially be the query result. We note that these instances should be decrypted and checked for the query conditions. In the fourth step, the algorithm decrypts only those `EncryptedData` elements satisfying *Relevant Encryption* $(ED, KEYS, Q) = true$, i.e., the `EncryptedData` elements containing part of the query result, and executes the query against the decrypted XML data.

Example 4. Suppose a user who has the keys k_1, k_2, and k_3 issues a query `//subject//HIV` against the encrypted XML data in Fig. 5(b) using the encrypted XML index in Fig. 5(c). A query processor first decrypts the Key Name field using the keys k_1, k_2 and k_3 in step 1. It then decrypts the Element Type fields of the element types `subject`, `name`, `analysis`, `DNAsignature`, `tests`, `HIV`, and `SARS` using the set of keys $\{k_1\}$, $\{k_2\}$, or $\{k_2, k_3\}$ in step 2. Next, it decrypts the Occurrences field of the `HIV` element, which is the element type of the query result, using the set of keys $\{k_1\}$ in step 3. Because the `HIV` element instance is located at 1.1.2.2.1, the `EncryptedData` element numbered 1.1 satisfies $RelevantEncryption(ED, KEYS, Q) = true$, but the `EncryptedData` element numbered 1.2.2 does not. Thus, a query processor does not decrypt and filter out the `EncryptedData` element numbered 1.2.2 in step 4.2. The element instances decrypted from the `EncryptedData` element numbered 1.1 are checked for the query conditions in step 4.1.

Query-Aware Decryption can be used also for twig queries [7]: for example, $//e_A[.//e_B//e_C]//e_D$. Twig queries are processed in two consecutive phases [7]. In the first phase, individual linear paths are extracted from the twig query ($//e_A//e_B//e_C$ and $//e_A//e_D$ in the example query), and the result for each is retrieved. In the second phase, results for individual linear paths are merge-joined. We can use Query-Aware Decryption during the first phase that handles only linear path expressions.

Example 5. Reconsider Example 4 with a twig query `//subject[.//HIV]//name` that retrieves the names of patients having HIV. Before executing Query-Aware Decryption, two linear path expressions—`//subject//HIV` and `//subject//name`—are extracted. Then, Query-Aware Decryption is applied to these linear path expressions, respectively. For `//subject//HIV`, as explained in Example 4, the `EncryptedData` element numbered 1.1 satisfies $RelevantEncryption(ED, KEYS, Q) = true$ because the `HIV` element instance is located at 1.1.2.1.1. This `HIV` element instance is returned as the result. For `//subject//name`, the `EncryptedData` element numbered 1.1 satisfies $RelevantEncryption(ED, KEYS, Q) = true$ because the `name` element instance is located at 1.1.1. This `name` element instance is returned as the result. These results for individual linear paths are merge-joined to check whether they come from the same twig. Therefore, the `name` element instance numbered 1.1.1 is returned as the query result.

Only the XPath axes that point to descendant nodes—`attribute`, `child`, and `descendant-or-self` (or `descendant`)—are primarily supported in Query-Aware Decryption. These axes are the most frequently used ones, and thus, even have abbreviated syntax: @, /, and //, respectively. XPath queries containing other axes such as `ancestor`, `following`, and `preceding` can be processed by assuming $RelevantEncryption(ED, KEYS, Q) = true$ in the same manner as in conventional methods [4,13]. Such queries, however, have minor effect on the overall performance since they are not frequently used.

4. Analysis of security compromise

Security compromise is the ability of inferring the information that a user is not permitted to know using the information that the user knows [10]. Since our method disseminates the encrypted XML index, we need to investigate whether, due to the encrypted XML index disseminated, a user can infer information that could not be inferred without the index.

In this section, we formally prove that dissemination of the encrypted XML index does not incur security compromise. For a background, Section 4.1

explains the query-view security [14] model proposed by Miklau and Suciu. Section 4.2 proves that our method is query-view secure.

4.1. Query-view security

Dobkin et al. defined *security compromise* [10] as in Definition 3, and Miklau and Suciu defined *query-view security* [14] as in Definition 4. Then, we discuss relationship between the two notions.

Definition 3 (*Dobkin et al.*). Suppose there are a set of data elements called UNKNOWN that a user U is not permitted to know and a set of data elements called KNOWN that a user U knows. A user U asks a sequence of queries $q_1,\ldots,q_n$ and enlarges his KNOWN. The security of a database is *compromised* if KNOWN and UNKNOWN intersect [10].

Definition 4 (*Miklau and Suciu*). Suppose confidential data in a database are represented by a query S, and a set of views $\{V_i\}$ are published. If the difference between the probability of guessing the result of S with $\{V_i\}$ and that without $\{V_i\}$ is zero, the query S is *query-view secure* to the set of views $\{V_i\}$ (Definition 3.1 in Ref. [14]).

Example 6 (*Miklau and Suciu*). Suppose there is a relation *Employee*(*name*, *department*, *phone*). (1) $S_2(n,p)$:-*Employee*(n,d,p) is not query-view secure to $V_2(n,d)$:-*Employee*(n,d,p) and $V'_2(d,p)$:-*Employee*(n,d,p) because the pairs of a name and a phone can be retrieved by joining V_2 and V'_2. (2) $S_4(n)$:-*Employee*$(n,$"*Shipping*", $p)$ is query-view secure to $V_4(n)$:-*Employee*$(n,$"*Admin*", $p)$ because the names of employees in the Admin department reveal nothing about those in the Shipping department.

S and $\{V_i\}$ in Definition 4 correspond to UNKNOWN and KNOWN in Definition 3, respectively. If the probability of guessing the result of S is not increased by the existence of $\{V_i\}$, UNKNOWN and KNOWN cannot be intersected. It means that what is query-view secure implies there is no security compromise. Miklau and Suciu [14] provide two theorems for deciding query-view security. That is, they provide Theorems 1 and 2 for deciding query-view security when prior knowledge does not exist or exists, respectively.

Theorem 1 (Miklau and Suciu). *A query S is query-view secure to a set of views $\{V_i\}$ iff the tuples that contribute to the results of a query S and those to the tuples of a set of views $\{V_i\}$ do not intersect* (*Theorem* 3.5 *in the Ref.* [14]).

Proof. See Ref. [14]. □

Theorem 2 (Miklau and Suciu). *Given a prior knowledge K, a query S is query-view secure to a view V iff the set of all the tuples T in a database can be partitioned into T_1 and T_2 satisfying the following condition*: (1) *K is the conjunction of two independent knowledge K_1 over T_1 and K_2 over T_2 ($K = K_1 \wedge K_2$)*; (2) *When K holds* (*i.e.*, *an adversary knows K*), {*the tuples that contribute to the results of S*} $\subseteq T_2$ *and* {*the tuples that contribute to the tuples of V*} $\subseteq T_1$ (*Theorem* 4.2 *in Ref.* [14]).

Proof. See the Ref. [14]. □

Fig. 7 describes the meaning of Theorem 2 by visualizing the explanation in Ref. [14]: "The set of all the tuples T in a database can be partitioned into T_1 and T_2 such that K is the conjunction of two independent knowledge K_1 over T_1 and K_2 over T_2. In addition, assuming K holds, S just says something about the tuples in T_2 (and nothing more about T_1). Similarly, when K holds, V just says something about the tuples in T_1 (and nothing more about T_2)".

4.2. Proof of query-view security of our method

In this section, we prove that the encrypted XML index does not make our method compromise security any more than the methods by Bertino and Ferrari [4] and Miklau and Suciu [13]. For this purpose, we first assume that these existing methods are query-view secure, and then, prove that our method is also query-view secure. When applying Theorems 1 and 2 to our proof, a tuple corresponds to an element instance since our algorithm is based on the XML model in contrast to the Ref. [14], which is based on the relational model.

If we make the assumption that these existing methods are query-view secure, the following proposition must be true. Suppose KEY_{granted} is a set of the keys possessed by a user U. The element instances encrypted using the sets of the keys $KEY_S \not\subseteq KEY_{\text{granted}}$ correspond to S in Definition 4, and the union of those encrypted using the sets of the keys $KEY_V \subseteq KEY_{\text{granted}}$ and those not

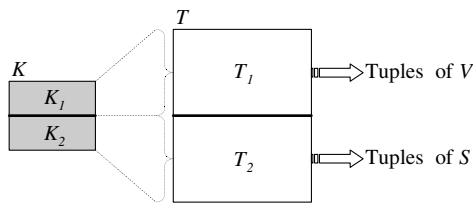


Fig. 7. The meaning of Theorem 2.

encrypted correspond to $\{V_i\}$ in Definition 4. We note that the element instances (i.e., tuples in Theorem 1) that contribute to S and those to V do not intersect according to Theorem 1.

We now present Theorem 3, which states about the query-view security of our method.

Theorem 3. *The Query-Aware Decryption algorithm in Fig. 6 using the encrypted XML index is query-view secure provided that existing methods without using the encrypted XML index are query-view secure.*

Proof. We regard the set of all the entries in the encrypted XML index as a prior knowledge K and apply Theorem 2 to prove Theorem 3. Since the existing methods without the encrypted XML index are query-view secure, the set of all the element instances T is partitioned into T_1, which is the set of the element instances encrypted using the sets of the keys $KEY_V \subseteq KEY_{\text{granted}}$ and those not encrypted, and T_2, which is the set of the element instances encrypted using the sets of the keys $KEY_S \not\subseteq KEY_{\text{granted}}$, according to Theorem 1. Now, we show that K and T satisfy the two conditions in Theorem 2. (1) K is partitioned[2] into K_1, which is the set of the index entries encrypted using the sets of the keys $KEY_V \subseteq KEY_{\text{granted}}$ and those not encrypted, and K_2, which is the set of the index entries encrypted using the sets of the keys $KEY_S \not\subseteq KEY_{\text{granted}}$. As discussed in Section 3.2, the keys used for encrypting element instances in XML data are the same as those for encrypting the index entries pointing to them. Hence, the index entries in K_1 store information about the element instances only in T_1 and do not have any information about other element instances. A similar statement applies to K_2. Thus, K_1 and K_2 satisfy the first condition in Theorem 2. (2) The element instances in T_2 constitute a set of the element instances S that are not accessible to the user, and those in T_1 constitute a set of the element instances V that are accessible to the user. Thus, T_1 and T_2 satisfy the second condition in Theorem 2. Fig. 8 illustrates the proof of Theorem 3. □

5. Performance evaluation

In this section, we evaluate the performance of the query processing method that uses Query-Aware Decryption. Section 5.1 describes the experimental data and environment. Section 5.2 presents the results of the experiments.

[2] Miklau and Suciu [14] interpret the meaning of $K = K_1 \wedge K_2$ as a partition. Thus, we show that K is partitioned into K_1 and K_2 here.

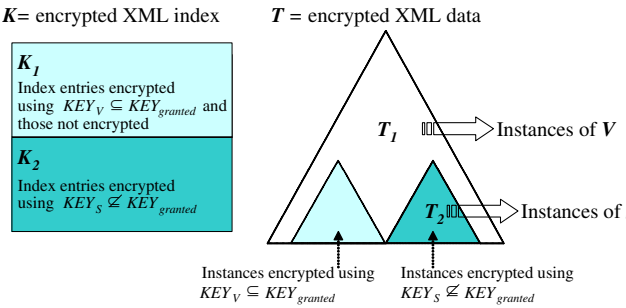


Fig. 8. The proof of Theorem 3.

5.1. Experimental data and environment

The objectives of the experiments are (1) to show that the size of the encrypted XML index is so small that the overhead of encrypting, disseminating, and decrypting this index is negligible compared with the overhead of the XML data and (2) to show that the performance of processing queries against encrypted XML data improves significantly because Query-Aware Decryption prevents unnecessary decryption. We compare the performance of our method with that of the method by Miklau and Suciu [13].[3] We call simply the method by Miklau and Suciu as *Unconditional Decryption* because it decrypts unconditionally all the `EncryptedData` elements that have been encrypted using the keys provided.

We assume that two users at different servers are involved in the experiments: the user A at the server A is a data owner, and the user B at the server B a data consumer. The user A encrypts his/her own XML data and disseminates the encrypted XML data as well as the encrypted XML index to the user B. After encryption is completed, the size of the encrypted XML index is measured at the server A. After dissemination is completed, the user B issues queries against the encrypted XML data. At this time, the query processing times of Query-Aware Decryption and Unconditional Decryption are measured at the server B.

The two parameters in Fig. 9 influence the performance of processing queries against encrypted XML data. Intuitively, $\text{ratio}_{\text{encrypted}}$ represents how many elements are encrypted, and $\text{ratio}_{\text{relevant}}$ does how many `EncryptedData` elements contain query results. Some `EncryptedData` elements are hidden when their ancestors are encrypted, but we include them when computing the two parameters.

[3] We do not compare with the method by Bertino and Ferrari [4] since they have not proposed a method for processing queries against encrypted XML data.

$$ratio_{encrypted} = \frac{\text{the total number of } \textbf{EncryptedData} \text{ elements in encrypted XML data}}{\text{the total number of elements in unencrypted XML data}}$$

$$ratio_{relevant} = \frac{\text{the number of } \textbf{EncryptedData} \text{ elements satisfying } RelevantEncryption(ED, KEYS, Q) = true}{\text{the total number of } \textbf{EncryptedData} \text{ elements}}$$

Fig. 9. Two parameters used in the experiments.

We vary $\text{ratio}_{\text{encrypted}}$ from 10% to 40% and $\text{ratio}_{\text{relevant}}$ from 0.1% to 66% (which is the maximum given $\text{ratio}_{\text{encrypted}}$ of 10–40%). We encrypt the XML data using the given values of $\text{ratio}_{\text{encrypted}}$ and $\text{ratio}_{\text{relevant}}$ as follows. Suppose E_{total} is a set of all the elements, and E_{relevant} a set of query results and their ancestors. We note that encrypting an element in E_{relevant} satisfies *RelevantEncryption*$(ED, KEYS, Q) = true$ according to Lemma 1. Thus, to satisfy the given values of $\text{ratio}_{\text{encrypted}}$ and $\text{ratio}_{\text{relevant}}$, we control the ratio of the number of encrypted elements in E_{relevant} to the total number of elements in E_{relevant} and the ratio of the number of encrypted elements in $E_{\text{total}} - E_{\text{relevant}}$ to the total number of elements in $E_{\text{total}} - E_{\text{relevant}}$. We call the former $\text{ratio}'_{\text{encrypted}}$ and the latter $\text{ratio}''_{\text{encrypted}}$. The parameters in Fig. 9 are represented as in the formulas (1) and (2). From the formulas (1) and (2), we obtain $\text{ratio}'_{\text{encrypted}}$ and $\text{ratio}''_{\text{encrypted}}$ as in the formulas (3) and (4). We select randomly $|E_{\text{relevant}}| \times \text{ratio}'_{\text{encrypted}}$ elements from E_{relevant} and $|E_{\text{total}} - E_{\text{relevant}}| \times \text{ratio}''_{\text{encrypted}}$ elements from $E_{\text{total}} - E_{\text{relevant}}$, and then, encrypt the elements selected.

$$\text{ratio}_{\text{encrypted}} = \frac{|E_{\text{relevant}}| \times \text{ratio}'_{\text{encrypted}} + |E_{\text{total}} - E_{\text{relevant}}| \times \text{ratio}''_{\text{encrypted}}}{|E_{\text{total}}|} \quad (1)$$

$$\text{ratio}_{\text{relevant}} = \frac{|E_{\text{relevant}}| \times \text{ratio}'_{\text{encrypted}}}{|E_{\text{total}}| \times \text{ratio}_{\text{encrypted}}} \quad (2)$$

$$\text{ratio}'_{\text{encrypted}} = \frac{|E_{\text{total}}| \times \text{ratio}_{\text{encrypted}} \times \text{ratio}_{\text{relevant}}}{|E_{\text{relevant}}|} \quad (3)$$

$$\text{ratio}''_{\text{encrypted}} = \frac{|E_{\text{total}}| \times \text{ratio}_{\text{encrypted}} \times (1 - \text{ratio}_{\text{relevant}})}{|E_{\text{total}} - E_{\text{relevant}}|} \quad (4)$$

We use the XMark [16] benchmark data set, which is a synthetic data set. We measure the elapsed time for executing an XPath query against the encrypted XML data because the CPU cost of decryption is the prevailing cost. The XPath query executed is *//site//open_auctions[.//bidder]//seller*. We also tested with various other queries, but omit the experimental results for them since they have a similar tendency. This tendency happens because the data decryption time takes a very large portion of the query processing time.

Now, we summarize in Table 1 the ranges of the parameters varied in the experiments.

We conduct all the experiments on a SUN Ultra 60 workstation with a 450 MHz CPU and 512 MB of main memory. To implement the experimental

Table 1
The ranges of the parameters

Parameter	Range
$\text{Ratio}_{\text{encrypted}}$	10–40%
$\text{Ratio}_{\text{relevant}}$	0.1–66%
Data size	10 KB–1 MB

program, we need two functions: encrypting/decrypting XML data and executing XPath queries against XML data. We use Apache XML-Security [2] and Apache XALAN [3], respectively. Apache XML-Security is a library supporting the XML encryption standard. Apache XALAN is a main-memory XSLT processor supporting XPath queries.

The encryption/decryption algorithm significantly influences the performance of processing queries. We adopt the RSA (version 1.5) algorithm according to the XML encryption standard [11]. The worse the performance of the encryption/decryption algorithm, the bigger the difference in performance between Query-Aware Decryption and Unconditional Decryption will be since, in Unconditional Decryption, the cost wasted by unnecessary decryptions gets proportionally larger.

5.2. Results of the experiments

5.2.1. The size of the encrypted XML index

Fig. 10 shows the sizes of the encrypted XML data and encrypted XML index generated from the XMark benchmark data of 1 MB. The results indicate that the size of the encrypted XML index is less than $\frac{1}{20}$ of the size of the encrypted XML data. The size of the encrypted XML data increases as

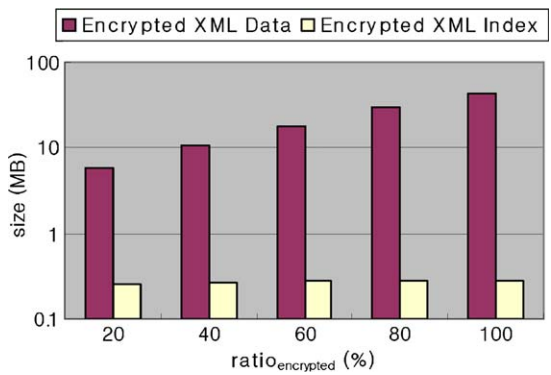


Fig. 10. The sizes of the encrypted XML data and encrypted XML index.

$\text{ratio}_{\text{encrypted}}$ does, while that of the encrypted XML index does not. This is caused by the difference in the way they are encrypted. When the XML index is encrypted, the Key Name, Element Type, and Occurrences fields are replaced simply with an encrypted string. When the XML data are encrypted, however, the tags of `EncryptedData`, `EncryptionMethod`, `KeyInfo`, `CipherData`, and `EncryptionProperties` elements are inserted according to the XML encryption standard [11] in addition to the encrypted string replacing an element name and element body. These tags inserted for each encrypted element is the main reason for the increased size. Besides, experimental results show that the index encryption time is much smaller (less than 12.6%) than the data encryption time.

5.2.2. The query processing time of Query-Aware Decryption

Fig. 11 shows the elapsed time for a query *//site//open_auctions*[*.//bidder*]*//seller* against the XML data generated by encrypting the XMark benchmark data of 1 MB.[4] The index decryption time in Query-Aware Decryption is less than 15 ms, which is a very small portion of the query processing time. In Fig. 11(a), (b), and (c), we fix $\text{ratio}_{\text{encrypted}}$ to 10%, 20% and 40% and vary $\text{ratio}_{\text{relevant}}$ from 0.1% to 66%, 33% and 16%, respectively. The possible maximum values of $\text{ratio}_{\text{relevant}}$ are derived using the formula (2) with $|E_{\text{total}}| = 17{,}132$ and $|E_{\text{relevant}}| = 1145$. The sizes of the encrypted XML data are 3.2 MB, 5.6 MB and 11.5 MB in Fig. 11(a), (b) and (c), respectively.

Fig. 11 shows that the elapsed time of Query-Aware Decryption decreases as $\text{ratio}_{\text{relevant}}$ decreases. It is natural because Query-Aware Decryption decrypts only those `EncryptedData` elements satisfying $\mathit{RelevantEncryption}(ED, KEYS, Q) = \mathit{true}$. On the other hand, the elapsed time of Unconditional Decryption is insensitive to $\text{ratio}_{\text{relevant}}$ since it decrypts all the `EncryptedData` elements unconditionally. The minor increment in Unconditional Decryption is related to the positions of the encrypted elements: more ancestors of the query results are encrypted when $\text{ratio}_{\text{relevant}}$ gets higher, and the decryption time slightly increases when the elements on the same path are encrypted recursively [13]. The advantage of Query-Aware Decryption disappears when $\text{ratio}_{\text{relevant}} = 100\%$. However, $\text{ratio}_{\text{relevant}}$ is very small in a real environment since the number of query results is generally much smaller than the total number of elements in the XML data, thus making Query-Aware Decryption effective.

Fig. 11 shows that the difference between Query-Aware Decryption and Unconditional Decryption becomes larger as $\text{ratio}_{\text{encrypted}}$ increases. Compared with Unconditional Decryption, Query-Aware Decryption improves the

[4] We tested with the XMark benchmark data of 10 KB, 100 KB and 1 MB. The tendency is similar for other sizes of the XML data. Hence, we only show the result for 1 MB.

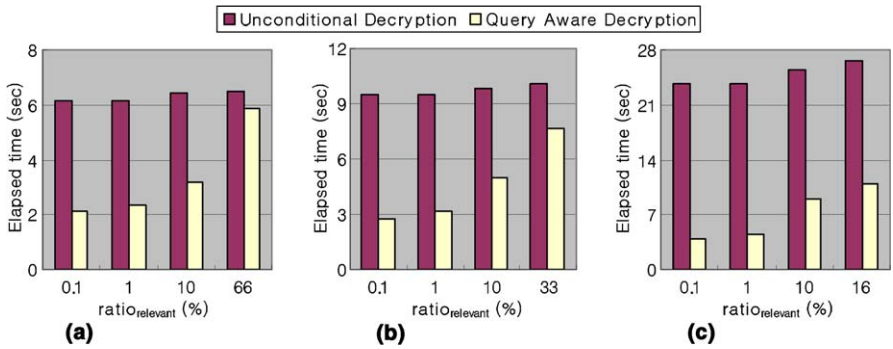


Fig. 11. The query processing time against the encrypted XML data. (a) $\text{ratio}_{\text{encrypted}} = 10\%$, (b) $\text{ratio}_{\text{encrypted}} = 20\%$, and (c) $\text{ratio}_{\text{encrypted}} = 40\%$.

elapsed time by up to 2.9 times in Fig. 11(a), 3.5 times in Fig. 11(b), and 6.1 times in Fig. 11(c). This is because, for a higher $\text{ratio}_{\text{encrypted}}$ value, Unconditional Decryption performs more decryptions, while Query-Aware Decryption effectively filters out unnecessary decryptions.

We tested multiple queries using the same XML data as before. We executed 100 queries randomly selected from a set of 10 queries and measured the average elapsed time. Here, we stored the decrypted portions of XML data into the cache and used them in subsequent queries. We observed that the average elapsed time of Query-Aware Decryption becomes similar to that of Unconditional Decryption. This result is natural because most of the accessible portions of XML data resided in the cache. However, it is not desirable to use this scheme in a practical environment since it can compromise security, for example, by leakage of decrypted data.

6. Conclusions

In this paper, we have proposed the notion of *Query-Aware Decryption*, which allows us to decrypt only those encrypted data that contain query results. For Query-Aware Decryption, we disseminate the encrypted XML index that summarizes the structure of the XML data. We have shown, in Lemma 1, that the encrypted elements containing query results can be identified using the encrypted XML index. We have implemented the query processing method using Query-Aware Decryption as the algorithm Query-Aware Decryption.

We have formally proven, in Theorem 3, that dissemination of the encrypted XML index does not cause additional security compromise. That is, on the assumption that existing methods without the encrypted XML index do not compromise security, Theorem 3 guarantees that our method with the index

still does not compromise security. We have used the query-view security [14] model as the criterion for checking security compromise. Miklau and Suciu [14] provided Theorem 2 to decide query-view security when prior knowledge exists. We have regarded the encrypted XML index as prior knowledge and applied Theorem 2 to prove that our method is query-view secure.

We have performed extensive experiments on the performance of Query-Aware Decryption. Experimental results show that (1) the size of the encrypted XML index is less than $\frac{1}{20}$ of the size of encrypted XML data, and (2) Query-Aware Decryption improves the query performance by up to six times compared with Unconditional Decryption. The results also show that the advantage of Query-Aware Decryption becomes more marked as the number of encrypted elements increases or as the number of encrypted elements that contain query results decreases.

The primary contribution of this paper is to propose an efficient method for processing queries against encrypted XML data using the new notion of Query-Aware Decryption. We believe that our work provides a practical method—implementable in P2P applications disseminating encrypted XML data. We are investigating how to incorporate our method into a real P2P application in a future work.

Acknowledgement

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

References

[1] N. Ahituv, Y. Lapid, S. Neumann, Processing encrypted data, Commun. ACM 30 (9) (1987) 777–780.
[2] Apache Software Foundation, XML-Security version 1.1. Available from: <http://xml.apache.org/security/c/>, 2004.
[3] Apache Software Foundation, Xalan-C++ version 1.8. Available from: <http://xml.apache.org/xalan-c/>, 2004.
[4] E. Bertino, E. Ferrari, Secure and selective dissemination of XML documents, ACM Trans. Inform. Syst. Security 5 (3) (2002) 290–331.
[5] S. Boag et al., XQuery 1.0: An XML Query Language, W3C Working Draft, November 2003.
[6] D. Booth et al., Web Services Architecture, W3C Working Group Note, February 2004.
[7] N. Bruno, N. Koudas, D. Srivastava, Holistic Twig Joins: Optimal XML Pattern Matching, in: Proceedings of 2002 ACM SIGMOD International Conference on Management of Data, ACM SIGMOD, Madison, Wisconsin, June 2002, pp. 310–321.
[8] L.M. Chan et al., Dewey Decimal Classification: A Practical Guide, second ed., OCLC Forest Press, 1996.

[9] J. Clark, S. DeRose, XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999.
[10] D. Dobkin, A.K. Jones, R.J. Lipton, Secure databases: protection against user influence, ACM Trans. Database Syst. 4 (1) (1979) 97–106.
[11] T. Imamura, B. Dillaway, E. Simon, XML Encryption Syntax and Processing, W3C Recommendation, December 2002.
[12] C. Faloutsos, Access methods for text, ACM Comput. Surveys 17 (1) (1985) 49–74.
[13] G. Miklau, D. Suciu, Controlling access to published data using cryptography, in: Proceedings of 29th International Conference on Very Large Data Bases, Berlin, Germany, September 2003, pp. 898–909.
[14] G. Miklau, D. Suciu, A formal analysis of information disclosure in data exchange, in: Proceedings of 2004 ACM SIGMOD International Conference on Management of Data, ACM SIGMOD, Paris, France, June 2004, pp. 575–586.
[15] M. Parameswaran, A. Susarla, A.B. Whinston, P2P networking: an information-sharing alternative, IEEE Comput. 34 (7) (2001) 31–38.
[16] A.R. Schmidt et al., XMark: A benchmark for XML data management, in: Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China, August 2002, pp. 974–985.
[17] D.X. Song, D. Wagner, A. Perrig, Practical techniques for searches on encrypted data, in: Proceedings of 2000 IEEE Symposium on Security and Privacy, Oakland, California, May 2000, pp. 44–55.
[18] C. Zhang et al., On supporting containment queries in relational database management systems, in: Proceedings of 2001 ACM SIGMOD International Conference on Management of Data, ACM SIGMOD, Santa Barbara, California, May 2001, pp. 425–436.