Confidentiality Enforcement for XML Outsourced Data*

Barbara Carminati and Elena Ferrari

University of Insubria at Como, Via Carloni, 78 - 22100 Como, Italy
email: barbara.carminati,elena.ferrari@uninsubria.it

Abstract. Data outsourcing is today receiving growing attention due to its benefits in terms of cost reduction and better services. According to such paradigm, the data owner is no more responsible for data management, rather it outsources its data to one or more service providers (referred to as publishers) that provide management services and query processing functionalities. Clearly, data outsourcing leads to challenging security issues in that, by outsourcing its data, the data owner may potentially loose control over them. Therefore, a lot of research is currently carrying on to ensure secure management of data even in the presence of an untrusted publisher. One of the key issues is confidentiality enforcement, that is, how to ensure that data are not read by unauthorized users. In this paper, we propose a solution for XML data, which exploits cryptographic techniques and it is robust to the most common and relevant security threats. In the paper, we present the encryption methods and query processing strategies.

1 Introduction

Data outsourcing is today emerging as one of the most important trend in the area of data management. According to such paradigm, the data owner is no more totally responsible for data management. Rather it outsources its data (or portions of them) to one or more publishers that provide data management services and query processing functionalities. Main benefit of data outsourcing is cost reduction for the owner, in that it pays only for the services it uses and not for the deployment, installation, maintenance, and upgrades of DBMSs. By contrast, when data are outsourced such costs are amortized across several users. Another important benefit is scalability in that data owners could not become bottlenecks for the system, rather they can outsource their data to as many publishers as they needs according to the amount of data and the number of managed users. Clearly, data outsourcing leads to many research challenges. One of the most significant is related to security. The key problem is that, since the owner does not anymore manage its data, it may potentially loose control over them. The challenge is therefore how to ensure the most important security properties (e.g., confidentiality, integrity, authenticity) even if data are managed by a third party. A naive solution is to assume the publisher to be *trusted*, that is, to assume it always operates according to the owner's security policies. However, making this assumption is not realistic, especially for web-based systems that can be easily attacked and penetrated. Additionally, verifying that a publisher is trusted is a very costly operation.

* The work reported in this paper has been partially supported by the Italian MIUR under the project 'Web-based management and representation of spatial and geographical data'.

Therefore, the research is now focusing on techniques to satisfy main security properties even in the presence of an untrusted publisher that can not always follow owner's security policies (for instance it can maliciously modify/delete the data it manages or it can send data to non authorized users). In this paper, we make a contribution to this research by focusing on confidentiality, that is, protection against non authorized reading operations, since it represents one of the most relevant security properties. Moreover, we cast our techniques in the framework of XML [11], since it is today the de-facto standard for data modeling and exchange over the web.

When data are outsourced confidentiality has two main aspects. The first, which we call *confidentiality wrt users*, refers to protect data against unauthorized read operations by users. The second, which we call *confidentiality wrt publishers*, deals with protecting owner's data from read operations by publishers. Our solution enforces confidentiality by using cryptographic techniques: publishers manage ciphered data instead of clear-text ones. In that way confidentiality wrt publishers is always ensured. Data encryption is generated by the owner and it is driven by the specified access control policies: all data portions to which the same policies apply are encrypted with the same key. Then, each user receives only the keys corresponding to the policies he/she satisfies. This ensures confidentiality wrt users. Enforcing confidentiality through the use of encryption requires dealing with several issues. The first is encryption generation in that there is the need to devise an encryption scheme which is robust to the most relevant security attacks. For instance, if you consider keyword-based searches on textual data, if the same keyword is always encrypted with the same key, both publishers and users can infer information by analyzing the document encryption. The same problem arises at the schema level, that is, with encrypted tags or attributes names. In this work we propose an encryption scheme that avoids information leakage due to data dictionary attacks. The second main issue is how publishers can query encrypted data. In the literature, several methods exist to this purpose.[1] Some of them have been designed for relational databases [6, 7], others have been designed to query textual data [10]. Our work is inspired by both of them, since an XML document contains both text and attributes with standard domains. In the paper, we present our strategy to query XML encrypted data and we describe both publisher side and user side query processing. The work described in this paper is part of a wider project whose goal is to ensure the most relevant security properties (i.e., authenticity/integrity and completeness in addition to confidentiality) when data are managed by a third party. Therefore, before going into the details of confidentiality protection we describe the overall architecture of the system we propose. Details on techniques for authenticity/integrity and completeness verification can be found in [1]. The remainder of this paper is organized as follows. Next section surveys related work that are the basis of our proposal. Section 3 deals with confidentiality enforcement. Section 4 illustrates client-side query processing. Finally, Section 5 concludes the paper.

2 Related work

In recent years the problem of inquiring encrypted data managed by a third party has been deeply investigated by several researchers, that proposed different solutions [6, 7,

[1] Some of them are surveyed in Section 2.

10] for different data models and domains. In general, the most appropriate solution to query encrypted data mainly depends on the nature of the data being queried, that is, the data domain and the underlying data model. Thus, in order to select the most appropriate technique to inquire XML encrypted data, we need to take into account the characteristics of XML data. XML documents often contain data with heterogeneous domains (e.g, textual data, date, integer). For this reason, we believe that in the scenario considered in this paper, it is not enough to use a single technique to inquiry encrypted XML data. Rather, different techniques should be combined into a unified framework to manage data with different domains. For instance, the work by Hacigumus et al. [6, 7] develops a method to query encrypted data stored in relational databases. From such work we borrow the method to query non textual data. By contrast [10] deals with keyword-based searches on textual data. We use some of the methods proposed in [10] to solve a twofold issue, that is, querying textual data and avoiding information leakage due to data dictionary attacks at the schema level. However, with difference to our proposal, such approaches only consider confidentiality wrt publishers, whereas they do not consider confidentiality wrt users. In what follows, we briefly introduce the techniques proposed in [6, 7] and [10], whereas details on how these two techniques are used in our framework are presented in Section 3. Confidentiality enforcement through the use of encryption techniques has also been investigated by us in [4]. The current work extends our previous work along several relevant directions. The first is that in this paper we provide a comprehensive method to query encrypted XML documents, which exploits a variety of strategies to query XML documents with heterogeneous content. In [4] we adapt the strategy proposed in [6, 7] to query both textual and non textual attributes as well as data with textual domain. The technique proposed in [6, 7] has a major shortcoming, i.e., to be prone to data dictionary attacks by both publishers and users. This kind of attack can be for instance perpetrated when the same data portion (e.g., attribute value, tag name) repeatedly appears in an XML document ciphered with the same key. In this paper, we solve this problem by applying a combination of strategies to treat XML data, able to trade-off between query expressivity and robustness to security threats. Another weak point of the solution proposed in [4] is information leakage due to inferences at the schema level in that tags/attributes with the same names and covered by the same access control policies are encrypted with the same key. Therefore, by analyzing document encryptions both publishers and users can infer information on the schema of some document portions, even if they are not allowed to access such portions. This is a relevant security threat since tags and attributes names can convey semantic relevant information (for instance, by exploiting such kind of attack a user can infer the existence of an element named `salary` even if he/she is not allowed to access it according to owner's security policies).

2.1 Hacigums et al.

The approach proposed by Hacigumus et. al [6, 7] exploits binning techniques and privacy homomorphisms to inquiry encrypted relational data. Binning techniques are used to perform selection queries on encrypted relation data, whereas privacy homomorphisms are used to make a third party able to perform aggregate queries over encrypted tuples. In our framework, we assume that users submit queries through XPath [11], with the obvious intention to extend the approach to support XQuery [11]. Thus, since

XPath expressions do not contain aggregate functions, in what follows we review only the approach for selection queries.

The underlying idea of the approach is the following: given a relation R, the owner divides the domain of each attribute in R into distinguished partitions, to which it assigns a different id. Then, the owner sends the publisher the encrypted tuples, together with the ids of the partitions corresponding to each attribute value in R. The publisher is able to perform queries directly on the encrypted tuples, by exploiting the received partition ids. The idea is that a user, before submitting a query to a publisher, rewrites it in terms of partition ids. As an example, consider the relation $Employee(eid, ename, salary)$, and, for simplicity, consider only the *salary* attribute. Suppose that the domain of *salary* is in the interval [500k, 5000k], and that an equi-partition with 100k as range is applied on that domain. Suppose that a user wants to perform the following query: "SELECT * FROM Employee WHERE salary =1000k". It translates it into the query: "SELECT * FROM Employee WHERE salary=id_{1000k}", where id_{1000k} is the id of the partition containing the value 1000k. Clearly, users should receive by the owner information on the techniques used to partition data and generate ids. The publisher is then able to answer such query by exploiting only the received ids. The publisher returns an approximate result wrt the original query. For instance, with reference to the above example, it returns all the tuples of the *Employee* relation whose *salary* attribute belongs to the range [1000K, 1099K). A further query processing has thus to be performed by the user to refine the received answer.

2.2 Song et al.

Another approach that has greatly inspired the present work is the one proposed by Song et al. [10] for keyword searching on encrypted textual data. Given a text consisting of a sequence of words: $W_1, W_2, \ldots W_n$, the basic scheme proposed in [10] first encrypts each word using a symmetric encryption algorithm $E_k()$, with a single secret key k. Then the scheme generates the XOR of each encrypted word with a pseudo-random number. The resulting ciphered words are then outsourced to the third party. According to this scheme, when a user needs to search for a keyword W, it generates the encrypted word $E_k(W)$ and computes $E_k(W) \oplus S$, where S is the corresponding pseudorandom number. This simple scheme allows the third party to search for keyword W in the encrypted data, by simply looking for $E(W) \oplus S$, thus without gaining any information on the clear text. Since occurrences of the same word are xored with different pseudorandom numbers, by analyzing the distribution of the encrypted words, no information could be inferred regarding the clear text. In particular, in [10] the authors propose a generation process for pseudorandom numbers, that makes users able to locally compute pseudorandom numbers, without any interaction with the data owner. The scheme exploits a symmetric encryption function $E()$, and two pseudorandom numbers generator functions, namely F and f. In the following, with the notation $E_k(x)$ ($F_k(x)$, $f_k(x)$, respectively), we denote the result of applying E (F, f, respectively) to input x with key k. The scheme considers as input a set of clear-text words,

$W_1, W_2, \ldots, W_l$, with the same length n.[2] Given these set of words, the steps needed to generate the corresponding ciphered words are the following:

- data owner generates a sequence of pseudorandom values: $S_1 \ldots S_l$, of length $n-m$;[3]
- for each word W_j, the outsourced ciphered word C_j is generated according to the following formula: $C_j = E_k(W_j) \oplus < S_j, F_{K_j}(S_j) >$, where $K_j = f_k(FB_j)$, and FB_j denotes the first $n-m$ bits of $E_k(W_j)$.

Let us see now how query evaluation and decryption take place. When a user needs to search for a keyword W_i, he/she sends the third party $E_k(W_i)$ and key K_i, which can be locally computed by the user. Then, for each outsourced ciphered word C_i, the third party: *1)* calculates $C_i \oplus E_k(W_i)$; *2)* takes the first $n-m$ bits bts of the resulting value, and computes $F_{K_i}(bts)$, where K_i is the key received by the requiring user; *3)* if the result of $F_{K_i}(bts)$ is equal to the $n-m+1$ remaining bits, then the ciphered word C_i is returned. Indeed, if C_i contains the searched encrypted word, then $E_k(W_i) \oplus < S_i, F_{K_i}(S_i) > \oplus E_k(W_i) =< S_i, F_{K_i}(S_i) >$, for the properties of the XOR operator. When a user receives C_i as answer of a query, he/she is not able to extract the value $E_k(W_i)$ from C_i and thus to decrypt it, by simply using the decryption key k. Therefore, the scheme proposed in [10] assumes that users know S_i.[4] In such a way, user is able to recover FB_i, by xoring S_i with the first $n-m$ bits of C_i. Having FB_i, the user can generate K_i (i.e., $K_i = f_k(FB_i)$)), which can be used to compute $F_{K_i}(S_i)$. Finally, having $< S_i, F_{K_i}(S_i) >$ the user is able to extract from C_i the encrypted word $E_k(W_i)$, and to decrypt it.

3 Confidentiality enforcement

Enforcing confidentiality in an outsourced-based architecture requires to address confidentiality wrt both publishers and users. Since publishers could be untrusted, it is necessary to devise some mechanisms to avoid their malicious usage of owner's data. On the other hand, users are usually entitled to access only selected portions of owner's data based on their characteristics and profiles.

Confidentiality requirements wrt users are usually modelled through a set of access control policies stating who can access what portions of the owner's data. In traditional client-server architectures confidentiality wrt users is usually enforced by means of access control mechanisms (i.e., reference monitors), which mediate each user access request by authorizing only those in accordance with the owner's access control policies. A fundamental requirement is therefore the presence of a trusted environment hosting the reference monitor. In traditional scenarios, this environment is provided by the entity managing the data, i.e., the owner's DBMS server. Enforcing access control in a

[2] This set of words can be obtained by partitioning the input clear-text into atomic quantities (on the basis of the application domain), and by padding and splitting the shortest and longest words.

[3] Parameter m can be properly adjusted to minimize the number of erroneous answers due to collision of pseudorandom numbers generator $F()$ and $f()$.

[4] Users are able to generate it using the pseudorandom number generator and knowing the seed.

third party scenario would imply the delegation of the reference monitor tasks to publishers. However, since we are considering a scenario where assumptions on publisher trustworthiness cannot be done, such solution is no longer applicable.

To enforce confidentiality wrt both users and publishers we therefore propose an alternative solution based on cryptographic techniques. The underlying idea is that data owners outsource to publishers an encrypted version of the data they are entitled to manage, without providing them the corresponding decryption keys. Such encryption is generated in such a way to minimize the risks of data dictionary attacks. Therefore, the publisher is not able to access and, as a consequence, to misuse the outsourced data, since they are ciphered. To enforce confidentiality wrt users we propose a particular document encryption, hereafter called *well-formed encryption*, where different portions of the same document are encrypted with different keys, on the basis of the access control policies specified by the owner. In order to correctly enforce access control, it is therefore necessary to selectively distribute secret keys to users. According to the proposed architecture (see [4]), the appropriate keys are distributed by owner in such a way that each user obtains all and only the keys corresponding to the policies he/she satisfies. Policies are specified according to a credential-based access control model for XML data proposed by us in [2]. Appropriate keys for each user are therefore determined by evaluating user credentials against the specified policies. The well-formed encryption and the selective distribution of secret keys ensure confidentiality wrt users. Indeed, each node of the resulting encrypted document is accessible only to authorized users, that is, those users who have received the appropriate keys directly by owners. Even in the case that an untrusted publisher returns unauthorized nodes to a user, he/she is not able to access it, since he/she has not been provided with the corresponding keys.

Applying a cryptographic-based solution in a third party scenario implies to address two main issues. The first is related to the fact that publishers operate on ciphered data. Therefore, we need some mechanisms to make them able to perform queries over them. In Section 3.1, we show how Hacigumus et al. and Song et al. approach can be adopted in the context of XML. The second issue is the definition of encryption strategies able to reduce as much as possible security threats that can be perpetrated against the system. In particular, we have addressed information inference threat that a publisher (user) can perpetrate by analyzing the distribution of encrypted nodes. To overcome this drawback, in Section 3.2, we show how Song et al. scheme can be adopted to encrypt tagnames and attribute names and values, thus to avoid inference.

3.1 Inquiring encrypted XML data

As introduced in Section 2, selecting the most appropriate solution to query encrypted data mainly depends on the nature of the data, that is, the data domain and the underlying data model. Usually, an XML document contains data to be modeled into elements, which in turn could contain other elements in accordance to the structure of the modelled data. Whereas attributes are usually exploited to better describe data contained into the corresponding elements. Let us consider, for instance, an XML document modelling a book. It is reasonable to suppose that in such an XML document, `Chapter` elements contain book's chapters, whereas their attributes `Title` and `Number` store additional information about book's chapters. Therefore, in devising methods to query XML encrypted data, we need to consider that elements could be searched based on their attributes values and/or their contents. Thus, we need a method that makes

the publisher able to perform logical comparisons on encrypted attribute values, as well as keyword-based searches on encrypted element contents and textual attributes.

To cope with this requirement, we have adopted two different strategies to query XML encrypted data. In particular, we use an approach similar to the one proposed by Song et al.'s [10] to query elements and attributes with textual domain. By contrast, for non textual elements and attributes, we exploit the method proposed by Hacigumus et al. [6]. Thus, we assume that before encrypting a node n, the encryption process determines n's data domain. This task can be performed by means of information contained in the XMLSchema. Let us see in more details which are the query strategies for textual and non-textual encrypted data.

Textual data. Song et al.'s approach [10]. In particular makes a publisher able to search for a specific keyword on encrypted textual data without loss of data confidentiality. Applying such an approach to our scenario requires two adjustments wrt the original formulation. The first is because the scheme proposed in [10] works for words of the same length, whereas we need to consider words of variable length. Therefore, we adapt the scheme proposed in [10] to the management of variable length words, as follows. Let $\mathbf{W}$ be the longest word in the owner's dictionary, and let $L_{\mathbf{W}}$ be the length of $\mathbf{W}$. Thus, for each sequence of clear-text words: $W_1, W_2, \ldots, W_l$, with length $L_j \leq L_{\mathbf{W}}$, the steps for ciphered words generation are the following:

- for each word W_j, data owner pads it with a sequence $pdbts$ of $L_{\mathbf{W}} - L_j$ bits;
- data owner generates a sequence of pseudorandom values $S_1 \ldots S_l$, of length $L_{\mathbf{W}} - m$;
- for each keyword W_j the outsourced ciphered word C_j is generated by the following formula: $C_j = E_k(W_j||pdbts) \oplus < S_j, F_{K_j}(S_j) >$, where $K_j = f_k(FB_j))$, and FB_j denotes the first $n - m$ bits of $E_k(W_j||pdbts)$.

According to this scheme, users have to know $L_{\mathbf{W}}$, that is, the length of the longest word managed by the data owner, to be able to pad the searched keyword W_i, before submitting the query.

Finally, we need to define a method for keywords selection. In particular, given a node n, we need to state how to select keywords from n's content. A naive solution is to split the content into separate $L_{\mathbf{W}}$ blocks, and to treat them as distinguished keywords. This solution however is useless if we consider that the resulting ciphered words should be exploited for the search. Thus, it is obvious that some content analysis should be performed over the node's content before keywords selection. In particular, the solution we propose requires a first phase during which the owner preprocesses the textual data contained into an XML node and extracts a set of keywords.[5] Then, each keyword is ciphered according to the scheme introduced above.

Non-textual data. To make publishers able to evaluate queries on encrypted non-textual data, we adapt Hacigums et al.'s approach [6]. First, we have to deal with partition generation. In general, the choice of the most appropriate partitioning technique mainly depends on the node domain. For instance, for numeric data (such as integer, real, etc.), a strategy based on an equi-partitioning of the domain could be appropriate. By contrast, for temporal data a partitioning based on time intervals could be more appropriate. Therefore, in our system we associate a different partitioning

[5] Several techniques developed in the Information Retrieval field can be used to this purpose [9].

```
<Sec-Info>
  <Node-Info   Name='C_{K_1}(CD)'>
    < Query-Info> ...</Query-Info>
  </ Node-Info>
  <Attributes>
    <Node-Info   Name='C_{K_2}(Price)'   Value='C_{K_2}(30)'>
      <Query-Info> ... </Query-Info>
    </ Node-Info>
  </Attributes>
</Sec-Info>
```

Fig. 1. An example of `Sec-Info` element

function with each possible data domain, with the exception of textual domain. Thus, given a node n of a document d, by analyzing the XML schema defining d it is possible to select the appropriate partitioning function. The partitioning function takes as input the node value and returns the id of the partition to which the node belong to, generated according to the data domain.

3.2 Document encryption

Document encryption requires a first phase during which each node of the input document is associated with the proper secret key. Therefore, each node of the input document is first marked with the set of access control policies applied to it, and then a different secret key is generated for each different configuration of policies that applies to a document portion. Then, all the nodes are encrypted with the corresponding secret keys. In particular, we use an encryption strategy that preserves as much as possible the original structure of the XML document. In the well-formed encryption, the encryption of an XML element e is an XML element $\overline{e}$ having as tagname the encryption of $e.tagname$[6] and as element content the encryption of $e.content$. The well-formed encryption is therefore an XML document that preserves the elements relationships of the original document. We have decided to preserve as much as possible the structure of the original document in the well-formed encryption since this simplifies query formulation. Indeed, users formulate queries through XPath expressions that exploit the structure of the XML document.

However, preserving the original document structure implies some security threats. An untrusted publisher (or user) could infer information by analyzing the distribution of encrypted nodes. This threat could be easily perpetrated against tagnames, attribute names and values, since an XML document may often contain repeated elements and several attributes with the same names (or values). To overcome this drawback, rather than symmetric encryption we apply the scheme proposed by Song et al. (see Section 2.2) to encrypt tagnames and attribute names and values. By contrast, for element contents we have decided to adopt traditional symmetric encryption (e.g., TDES, AES), that requires less computational resources. This choice is motivated by the fact that probability of having a number of occurrences of the same encrypted element content sufficient for data dictionary attacks is small. This probability is further reduced in our context, where elements with the same content are encrypted with different keys, if they are protected by different access control policies.

[6] In the paper, given an element e (i.e., an attribute a) we use the dot notation to identify its tagname (name) $e.tagname$ (i.e., $a.name$) and content (value) $e.content$ (i.e., $a.value$).

To make publishers able to evaluate queries on encrypted XML documents, the resulting document encryption is complemented with additional information, called *query processing information*. This information consists of partition's ids or ciphered keywords associated with attribute values or element contents. Query processing information are encoded by an XML element, called `Sec-Info`, which is inserted into each element of the well-formed encryption (see Figure 1) and encodes query processing information of both the element itself and all its attributes. The `Sec-Info` element associated with an element e contains a mandatory subelement, named `Node-Info`, whose `Query-Info` subelement stores the query processing information corresponding to e. The `Query-Info` subelement contains the ciphered words extracted from $e.content$ or the id of the partition to which e belongs to, if element e has non-textual domain. By contrast, to model query processing information corresponding to each attribute of element e, the `Sec-Info` contains an `Attributes` subelement, which in turn contains a different `Node-Info` subelement for each attribute of e. The `Node-Info` subelement corresponding to attribute a of element e contains a `Query-Info` subelement storing the id of the partition to which the value of a belongs to or the ciphered keywords extracted from it, if attribute a has textual domain.

Algorithm 1 *The element encryption algorithm*

INPUT:
1. An XML element e
2. The encryption key K corresponding to the policy configuration applied to e

OUTPUT:
An XML element, $\overline{e}$, containing the encryption of e and its query processing information

1. Let $\overline{e}$ be an empty XML node;
2. Set $\overline{e}.tagname$ equal to $C_K(e.tagname)$;
3. Set $\overline{e}.content$ equal to $E_K(e.content)$;
4. Let se be an empty `Sec-Info` element;
5. Let ni be an empty `Node-Info` element subelement of se;
6. Set the `Name` attribute of ni equal to $C_K(e.tagname)$;
7. **If** the domain of e is textual:
   Let WS be the set of keywords extracted from $e.content$;
   For each $w \in WS$: Insert $C_K(w)$ into the `Query-Info` subelement of ni;
   Else
   Let $PF()$ be the partitioning function associated with e's domain;
   Insert $PF(e.content)$ into the `Query-Info` subelement of ni;
8. Return $\overline{e}$;

Let us now see in more details how given a document d, the generation of $\overline{d}$, that is, the document to be outsourced, is carried on. This process is realized by means of two different phases. During the first phase, each element e of d is encrypted, according to the strategy previously explained, and inserted into $\overline{d}$. Additionally, the `Sec-Info` element associated with e is generated. Then, attribute encryption is performed and the `Sec-Info` element is updated accordingly. Algorithm 1 deals with element encryption. The algorithm takes as input an element e and the encryption key corresponding to the policy configuration applied to e. First, it generates the encryption of the tagname and element content (steps 2 and 3). This is done by means of $C_K()$ function implementing the Song et al.'s scheme, and $E_K()$ function that performs symmetric encryption. Then, Algorithm 1 generates the query processing information associated

with the input element. In particular, in case of textual domain, the system extracts the set of meaningful keywords from *e.content* and generates the corresponding ciphered keywords, which are then inserted into the `Query-Info` element. By contrast, for element with non-textual domain, the query processing information consists of the id of the partition to which e belongs to.[7] In order to complete the well-formed encryption we need to consider all attributes of e. This is done during a second phase, implemented by Algorithm 2. This phase considers each attribute a and generates the encryption of its name and value, according to Song et al.'s scheme. Moreover, it generates the query processing information related to a, in the same way as Algorithm 1. The resulting information is then inserted into the `Sec-Info` element associated with the element to which a belongs to. This information is stored as a new `Node-Info` element inside the `Attributes` subelement of `Sec-Info`.

Algorithm 2 *The attribute encryption algorithm*

INPUT:
1. An XML attribute a
2. The encryption key K corresponding to the policy configuration applied to a

OUTPUT:
The updated `Sec-Info` containing the encryption of a and its query processing information

1. Let f be the element containing a;
2. Let se be the `Sec-Info` element in f;
3. Let ni be an empty `Node-Info` element;
4. Set the `Name` attribute of ni equal to $C_K(a.name)$;
5. Set the `Value` attribute of ni equal to $C_K(a.value)$;
6. **If** a has a textual domain:
   Let WS be the set of keywords extracted from $a.value$;
   For each $w \in WS$: Insert $C_K(w)$ into the `Query-Info` subelement of ni;
   Else
   Let $PF()$ be the partitioning function associated with a's domain;
   Insert $PF(a.value)$ into the `Query-Info` subelement of ni;
7. Insert ni in the `Attributes` subelement of se;
8. Return se;

4 Client side query processing

In the proposed system, users submit queries through a *client*, i.e., a program that users download from the owner site, and which makes them able to submit encrypted queries to publishers, and verify security properties on the received answers. In this section we show how the client is able to submit queries to publisher and decrypt the resulting nodes. Before going into the details we introduce the *query template*.

4.1 Query template

The query template has a twofold goal. The first is to make a user able to verify the completeness of a query result,[8] the second is to make a user able to formulate queries

[7] We assume that our system manages a library of partitioning functions, which associates with each different data domain a unique partitioning function.

[8] By completeness we mean that a user receives all document portions he/she is authorized to see according to the owner's access control policies.

and to decrypt the received results. We do not go into the details of completeness verification, since it is outside the scope of this paper (interested readers could refer to [4]). Rather, we focus on the information the query template contains for query processing.

Query templates are generated by the owner for each outsourced document and make available to all users for downloading. Query template contains the encrypted structure of the corresponding XML document and it is generated by using the same encryption strategy employed for XML documents. Moreover, the query template contains the encrypted pseudorandom numbers associated with each node, and needed by clients for document decryption. More precisely, all additional information associated with an element e needed for both query processing and completeness verification are encoded by an XML element, which is inserted as direct child of e into the query template. The element is defined according the syntax of the `Sec-Info` element (cfr. Section 3.2). All information is therefore stored into the `Node-Info` element. Each `Node-Info` element contains an additional attribute called `N` storing the encrypted pseudorandom number associated with the element tagname or attribute name to which the `Node-Info` element refers to.[9] Pseudorandom number is encrypted with the encryption key associated with the element/attribute to which it refers to. By contrast, attribute values can be often split into several words, where each of them is associated with a different pseudorandom number. Therefore, the encryption of these numbers are placed as content of an additional subelement, called `Numbers`, inside the `Node-Info` element.

By having the query template users are able to generate the authorized view of the structure of the document to be inquired, which can be exploited to formulate queries.

4.2 Query generation

In proposed framework, we assume that users submit queries through XPath expressions. XPath allows one to traverse the graph structure of an XML document and to select specific portions on the document according to some properties, such as the type of the elements, or specified content-based conditions. In general, an XPath expression consists of a *location path*, that allows one to select a set of nodes from target documents, which in turn consists of one or more *location steps*, separated among each other by a slash. A location step contains: an *axis*, specifying the tree relationships between the nodes selected by the location step and the current node (e.g., ancestor, ancestor-or-self, attribute, child, descendant, descendant-or-self); a *node test*, used to identify a node within an axis, by specifying a node type or the node name (e.g., text(), node()); and zero or more *predicates*, placed inside square brackets, used to further refine the set of nodes selected by the location step (e.g., [@Price='30']). By means of predicates an XPath expression can specify conditions on attributes through comparison operators (e.g., <,>,=), as well as conditions on textual data, in that it is possible to retrieve XML nodes containing a specified keyword (supported by the *contains()* function).

In order to submit a query to a publisher, the first step that a user has to perform is to download the query template of the requested document from the owner site. By decrypting the query template, the client can locally generate the structure of the

[9] We assume that tagnames and attribute names are shorten than $L_{\mathbf{W}}$, i.e., the max length, and thus are treated as a unique word. Therefore, they are associated with a unique pseudorandom number.

authorized view, which can be displayed to the user in order to formulate XPath queries on it. Such queries should then be translated by the client into one or more XPath expressions that could be evaluated by publishers on the encrypted XML documents. In order to do that there are two main transformations to which each location step of a user XPath expression must undergo before being submitted to a publisher. The first implies to cipher the tagnames specified in the node test of the location steps with the proper keys. Note that according to the well-formed encryption, nodes with the same name could be ciphered with different keys, based on the access control policies applied to them. Thus, the ciphering of a location step does not always return a unique value, which implies that for each user's query the client could generate more ciphered queries. The second transformation implies the translation of the query conditions in terms of partition ids and/or encrypted keywords. This is applied to each predicate composing the input XPath expression. If the predicate contains comparison operators (e.g., <,>,=), the client substitutes the values appearing in the predicate with the id of the partition to which it belongs to. This is done by using the information about the partitioning functions obtained during the subscription phase. By contrast, if the predicate exploits the contains() function, the client translates the condition by replacing the searched keyword W_j, with the corresponding encrypted word. We recall that according to the adopted scheme, for searching a keyword W_j publishers must be provided with its encryption and with key K_j, which must be sent by the client to publishers together with the submitted query.

To better understand query processing, let us consider an example. In particular, consider an XML document modelling a CD catalog, where a different `CD` element is inserted for each different CD in the catalog. The `CD` element contains an attribute, i.e., `Price`, storing the CD's price and two subelements, i.e., `Title` and `Author`, containing the CD's title and author, respectively. Let us suppose moreover that on this document two different access control policies apply, namely P_1 and P_2. The first authorizes all users to access all the nodes except for `Price` attributes. By contrast, P_2 grants store-staff users the access to the whole document. The outsourced document is thus encrypted with two different encryption keys, namely K_1 and K_2. K_1 is associated with policy configuration consisting of both P_1 and P_2 and encrypts all the nodes except for `Price` attributes. K_2 corresponds to policy P_2 and encrypts only the `Price` attributes. A store-staff member u receives by the owner both K_1 and K_2. Suppose now that u wants to submit the following query: `/CD`[@`Price`=30K]/`Title`[contains(.,'Overture')], which returns all the CDs having price equal to 30K and the keyword *Overture* in the title. Before submitting this query the client transforms it, by, at first, ciphering the tagname in each location step obtaining thus the following expression: $/C_{K_1}$(`CD`) [@C_{K_2}(`Price`)='30K']/C_{K_1}(`Title`)[contains(.,'Overture')]. Note that clients can easily cipher a tagname by simply extracting from the query template the corresponding pseudorandom number. Then, each condition specified in the predicates is translated. This is done by considering the query processing information of the node on which the predicate is specified. Let us for instance consider the predicate "contains(.,'Overture')" on element `Title`. According to the proposed strategy this can be evaluated by searching a ciphered word among those contained in the query processing information of the `Title` element that matches E_{K_1}('Overture'). Thus, the predicate that should be evaluated by the publisher is:

$/C_{K_1}(\texttt{CD})/C_{K_1}(\texttt{Title})/\texttt{Sec-Info}$ $/\texttt{Node-Info}/\texttt{Query-Info}[\text{contains}(.,\text{'}E_{K_1}(Overture)\text{'})]$.[10]

Furthermore, we have to note that in the proposed encoding the ciphered name of an attribute a is stored into the `Name` attribute of a `Node-Info` subelement of `Attributes`, which is contained into the `Sec-Info` element associated with the element e to which a belongs to. This implies that in our example conditions on price should be verified against the query processing information contained into the `Node-Info` element, whose `Name` attribute is equal to $C_{K_2}(\texttt{Price})$. Thus, predicates on the price attributes should be evaluated by the publisher as:

$/C_{K_1}(\texttt{CD})//\texttt{Attributes}/$ $\texttt{Node-Info}[\texttt{@Name}=C_{K_2}(\texttt{Price})/\texttt{Query-Info}$ [contains(.,'PF(30')],

where PF(30) returns the id of the partition to which the value 30 belongs to. The resulting XPath expression generated by the client is thus:

"$/C_{K_1}(\texttt{CD})/$ $C_{K_1}(\texttt{Title})/\texttt{Sec-Info}/\texttt{Node-Info}/\texttt{Query-Info}[\text{contains}(.,\text{'}E_{K_1}(Overture)\text{'})]$
AND
$/C_{K_1}(\texttt{CD})//\texttt{Attributes}/\texttt{Node-Info}[\texttt{@Name}=C_{K_2}(\texttt{Price})/\texttt{Query-Info}[\text{contains}(.,\text{'PF(30')})]$".

4.3 Decryption of query answers

Once the query has been evaluated, the publisher returns to client the encrypted nodes identified by the submitted XPath expressions. According to the adopted document encryption strategy, the client needs to perform two different decryption processes in order to decrypt the obtained nodes: one for tagnames, attribute names and values, and the other for decrypting element contents.

Decryption of tagnames, attribute names and values. Tagnames, attribute names and values are ciphered according to Song et al.'s scheme (cfr. Section 2.2). Given a ciphered word C_j, in order to extract from it the encrypted word $E_K(W_j)$, and thus to decrypt it, the client must be provided with the corresponding pseudorandom numbers. Such numbers are retrieved by the client from the query template (cfr. Section 4.1). By decrypting the pseudorandom numbers, the client is thus able to decrypt the ciphered tagnames, attribute names, and attribute values.

Decryption of element contents. Element contents are encrypted by means of a symmetric encryption algorithm. Therefore, the decryption of element content is simply performed by first retrieving the encryption key associated with the element, and then by executing the proper symmetric decryption algorithm.

5 Conclusions

In this paper, we have proposed a method based on cryptographic techniques for confidentiality enforcement on outsourced XML data. Main benefits of the proposed solution are that it does not make any assumption on the trustworthiness of publishers and it is robust to data dictionary attacks both at the schema and document content level. In the paper, besides illustrating the cryptographic schemes, we show by an example how

[10] Note that publishers implement a different contains() function wrt Xpath parsers, by, however, preserving the semantics.

client-side query processing takes place. Due to space limitations, we have not considered other important issues related to key management, for instance those related to the number of keys that need to be managed. To limit the number of keys we use a hierarchical key management scheme similar to the one proposed by us for temporal access control policies [3], which requires to permanently store a number of keys linear in the number of access control policies.

We are currently implementing the proposed strategies to test the performance of the system in different environments. Moreover, we are currently investigating techniques to efficiently manage updates to policies and documents, that would require a partial re-encryption of documents as well as an update of the related security information. In particular, in order to efficiently manage update operations, we plan to adopt a strategy similar to the one in [5] that incrementally maintains document encryptions, by changing all and only those portions which are really affected by the administrative operation, without the need of re-encrypting the document from scratch.

References

1. E.Bertino, B.Carminati, E.Ferrari, B. Thuraisingham, A. Gupta. Selective and Authentic Third-Party Distribution of XML Documents. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 16(10):1263–1278, 2004.
2. E. Bertino and E. Ferrari. Secure and Selective Dissemination of XML Documents. *ACM Transactions on Information and System Security (TISSEC)*, 5(3):290–331, 2002.
3. E. Bertino, B. Carminati, and E. Ferrari. A Temporal Key Management Scheme for Broadcasting XML Documents, In Proc. of the *9th ACM Conference on Computer and Communications Security (CCS'02)*, Washington, November, 2002.
4. B. Carminati, E. Ferrari, and E. Bertino. Securing XML Data in Third-Party Distribution Systems. In Proc. of the *ACM Fourteenth Conference on Information and Knowledge Management (CIKM'05)*, Bremen, Germany, November, 2005.
5. B. Carminati, E.Ferrari. Management of Access Control Policies for XML Document Sources. *International Journal of Information Security*, 1(4):236–260, 2003, Springer.
6. H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in the Database Service Provider Model. In Proc. of the *ACM SIGMOD 2002*, Madison, WI, USA, June 2002.
7. H. Hacigumus, B. R. Iyer, and S. Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In Proc. of the *9th International Conference on Database Systems for Advanced Applications*, Jeju Island, Korea, March 2004.
8. R.C. Merkle. A Certified Digital Signature. In *Advances in Cryptology-Crypto '89*, 1989.
9. G. Salton and M. McGill. *Introduction to Modern Information Retrival*. McGraw-Hill, 1983.
10. D. X. Song, D. Wagner and A. Perrig. Practical Techniques for Searches on Encrypted Data, In Proc. of the *IEEE Symposium on Security and Privacy*, Oakland, California, 2000.
11. World Wide Web Consortium. `http://www.w3.org`.