Efficient Secure Query Evaluation over Encrypted XML Databases

Hui (Wendy) Wang
Department of Computer Science, University of British Columbia
2366 Main Mall
Vancouver, Canada
hwang@cs.ubc.ca

Laks V.S. Lakshmanan
Department of Computer Science, University of British Columbia
2366 Main Mall
Vancouver, Canada
laks@cs.ubc.ca

ABSTRACT

Motivated by the "database-as-service" paradigm wherein data owned by a client is hosted on a third-party server, there is significant interest in secure query evaluation over encrypted databases. We consider this problem for XML databases. We consider an attack model where the attacker may possess exact knowledge about the domain values and their occurrence frequencies, and we wish to protect sensitive structural information as well as value associations. We capture such security requirements using a novel notion of *security constraints.* For security reasons, sensitive parts of the hosted database are encrypted. There is a tension between data security and efficiency of query evaluation for different granularities of encryption. We show that finding an optimal, secure encryption scheme is NP-hard. For speeding up query processing, we propose to keep metadata, consisting of structure and value indices, on the server. We want to prevent the server, or an attacker who gains access to the server, from learning sensitive information in the database. We propose security properties for such a hosted XML database system to satisfy and prove that our proposal satisfies these properties. Intuitively, this means the attacker cannot improve his prior belief probability distribution about which candidate database led to the given encrypted database, by looking at the encrypted database as well as the metadata. We also prove that by observing a series of queries and their answers, the attacker cannot improve his prior belief probability distribution over which sensitive queries (structural or value associations) hold in the hosted database. Finally, we demonstrate with a detailed set of experiments that our techniques enable efficient query processing while satisfying the security properties defined in the paper.

1. INTRODUCTION

To handle their data management needs, companies and businesses are increasingly moving toward the so-called "database as service" (DAS) paradigm [17]. The idea is that third parties provide data management services, hosting and managing their customer's business and/or personal data. Although this offers the possibility of reliable storage of large volumes of data, efficient query processing, and savings of database administration cost for the data owner, it raises serious questions about the security of the data managed by the service provider. Given that no customer is willing to implicitly trust their data to another party, we need a mechanism for protecting the privacy of sensitive data and prevent security breaches. The same concerns exist for untrusted disk scenario [22]. One solution to this problem is that the customer encrypts the sensitive parts of their data, sends the (partially) encrypted database to the service provider, and keeps the decryption key to himself. Note that the purpose of encryption is to protect the sensitive information from the untrusted server, and is orthogonal to any access control policies. We call the customer *client* or data *owner* and call the service provider *server.*

There are two kinds of information the client may consider as sensitive: (i) *individual node with its content* (here the content of the node consists of the tag, the sub-elements and the data values of leaf descendants) and (ii) *association between data values.* E.g., in a health care database, the owner may wish to protect every patient element. Alternatively, she may wish to protect the *associations* between patient's name and disease. To capture the client's security requirements, we propose the notion of *security constraints* (SCs) that support both types of security requirements above.

To enforce the security constraints specified by the data owner, we need a way to decide which elements in the hosted data need to be encrypted. Encryption may be done at different granularities. Encrypting the entire document, while guaranteeing all SCs are enforced, quashes any hopes of query optimization: for every query, the server needs to ship the whole encrypted database to the owner, which will decrypt and process the query, at considerable burden to itself. However, when the granularity is too fine, e.g., only leaf elements are encrypted, as we will show in Section 4.1, an attacker can infer sensitive information, based on his prior knowledge of the domain values and their occurrence frequencies. This might violate one or more SCs. Thus, one of the challenges is to find a "secure" and "optimal" encryption scheme (defined in Sections 3.4 and 4.2) which guarantees that SCs are enforced while permitting efficient query processing.

After the data is encrypted and sent to the server, in order to enable efficient query processing, some metadata needs to be maintained on the server side. We want to ensure that the

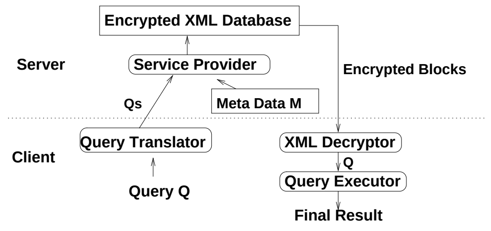


Figure 1: The Architecture

hosted database is "secure" in the presence of the metadata, i.e., that there are a large number of candidate databases, including the true hosted database, such that: (i) none of the candidate databases other than the true one contain any of the sensitive information being protected, (ii) by looking at the encrypted hosted data and the metadata, (a) the attacker cannot distinguish between any pair of candidate databases, (b) nor can he increase his belief probability that any of the databases in the candidate set is indeed the true hosted database.

Finally, we would like to make sure that even after observing a series of queries from client and responses by itself, the server's knowledge about the sensitive information in the database that is specified by the security constraints does not improve.

In a recent paper, Agrawal et al. [3] proposed a scheme for designing order preserving encryption functions, which enable efficient processing of queries over encrypted data. The strength and novelty of this approach is that range queries and aggregate queries involving MIN, MAX, COUNT can be evaluated at the server without decrypting the data. Compared with our work, they focus on the data distribution model as a combination of histogram-based and parametric techniques, while we consider the more conservative model of exact occurrence frequencies of domain values as the background knowledge possessed by the attacker.

The basic architecture and data/control flow of our system are shown in Figure 1. The client encrypts its data D using some encryption function η and some private key known only to itself. The encryption may be partial. The client also creates *metadata* M on D for the server's use. Then it sends the encrypted data $\eta(\mathsf{D})$ together with the *metadata* M to the server. When a query Q needs to be evaluated on the database D, the client translates Q into an encrypted query Q_s, and sends Q_s to the server. The server answers Q_s by using the metadata M. The answer to Q_s, i.e., $\mathsf{Q}_s(\eta(\mathsf{D}))$, consisting of a set of encrypted (and possibly some unencrypted) blocks, is sent to the client. The client decrypts the blocks as $\delta(\mathsf{Q}_s(\eta(\mathsf{D})))$, using decryption function δ, and does post-processing on the decrypted results using Q, such that $\mathsf{Q}(\delta(\mathsf{Q}_s(\eta(\mathsf{D})))) = \mathsf{Q}(\mathsf{D})$. We use XPath, the core of XQuery language, for illustrating query processing ideas in this paper.

We make the following contributions:

- We define the security properties for a hosted XML database system based on an attack model we define, consisting of *size-based attack* and *frequency-based attack* (Section 3).
- We propose a novel notion of *security constraints* (SCs) for the client to specify what information needs to be protected from the untrusted server (Section 3.2). It supports not only the protection of elements (both structure and contents) but also *associations* between data values in the database. We propose a mechanism to construct a secure encryption scheme to enforce the SCs (Section 4.1) and show that finding an optimal, secure encryption scheme for a given set of SCs is NP-hard (Section 4.2).
- We propose a scheme for the metadata to be stored on the server, to support efficient query processing. It consists of two key components: (i) the structural metadata called the *discontinuous structural interval index (DSI)* that pertains to the structure of the hosted database; (ii) a B-tree index as the metadata on the values that is based on transforming the unencrypted values by splitting and scaling techniques, so that the distribution of unencrypted data is different from that of ciphered data. We show that in the presence of meta-data, the hosted database system is secure (Section 5).
- We show how queries can be evaluated efficiently and prove the query answering is *secure*, i.e., even when the attacker observes a series of client queries and server responses, his belief in whether the sensitive information captured by the given security constraints exists in the database, does not increase (Section 6).
- We complement our analytical results with experiments on both real and synthetic data sets that measure the effect of encryption scheme granularity and query size and shape on query processing efficiency, as well as the client post-processing cost, demonstrating the effectiveness of our approach (Section 7).

We will sometimes refer to unencrypted data values as *plaintext* and encrypted values as *ciphertext*. Related work is discussed next. We summarize the paper in Section 8.

2. RELATED WORK

Hacigumus et al. [17] were one of the earliest to study the DAS model. Their focus is the evaluation of SQL queries over hosted relational databases. They use conventional encryption techniques and additionally assign a bucket id to each attribute value to facilitate efficient evaluation. The client needs to decrypt the results sent by the server and then execute a compensation query to extract the final answer. We share most of the goals with this work. Agrawal et al. [3] have proposed a method of designing order-preserving encryption functions such that the distribution of the encrypted values follows an arbitrarily chosen target distribution. Neither of the above works consider user specified security constraints. We focus on a problem of transforming the input data distribution such that by looking at both source and target distributions, the attacker cannot easily map the plaintext data to the ciphertext data, in a provable sense. This is a valuable technique since on the one hand, using order-preserving encryption, range queries (as well as aggregate queries involving MIN, MAX, COUNT) can be evaluated efficiently without decryption. At the same time, an attacker can be fooled by the target distribution. The notion of encryption scheme granularity is not an issue in their works. Finally, in our model, unlike the above works, we consider the more conservative model where the attacker may know both the domain values and their exact occurrence frequencies. All our security theorems are proved w.r.t. this background knowledge.

Protecting data privacy in relational databases has been addressed in various works (e.g., see [13], [18]). Kantarcioglu et al. [19] studied security issues in querying an encrypted database from a cryptographic viewpoint and showed that any secure database system that meets the strict cryptographic definitions of security will require a linear database scan for answering queries, and thus will be inefficient. Damiani et al. [12] proposed an indexing scheme and an efficient query evaluation scheme for encrypted relational data, and quantitatively measured the resulting inference exposure. They showed that the indexes proposed in [17] can open a door to interference and linking attacks.

Brinkman et al. [7] proposed a method of applying queries on encrypted XML documents. Their approach stores a relational table containing structural information of the XML database at the server, thus compromising privacy of structural information. Besides, they do not support flexible levels of encryption granularity. Bertino et al. [6] proposed differential encryption schemes for XML documents using multiple keys. Miklau et al. [23] proposed a framework for enforcing access control policies on published XML documents using cryptography. The security guarantee of this framework is precisely characterized by Abadi et al. in [1]. In another paper, Miklau et al. [24] studied the problem of query-view security and showed how to decide security based on a novel information-theoretic standard they proposed for analyzing the information disclosure of published views.

Private information retrieval (PIR) (e.g., see [9]) is a related field of research. Pure information-theoretic notion of privacy requires that even a computationally unbounded server cannot distinguish between queries intended at different targets. It is not practical for database applications for the multiple copies and huge communication overheads. Aggarwal et al. [2] proposed partitioning the database between two servers so that associations between data items can be kept private from each of the servers (in an intuitive but not cryptographic sense), but can be inferred by the client. A basic assumption is that the servers cannot communicate with each other. One-server schemes with sub-polynomial communications has been proposed in [20], but imposes expensive computational overhead. Liu and Candan [21] propose a computationally PIR one-server scheme that achieves both content and access privacy with moderate computation and communication cost. However, this comes at the price of access redundancy and the client having to employ the node swapping, which can be expensive.

There has been significant research on access control, privacy preserving data mining, secure XML publishing, and trusted computation (see [2] for a survey). While they all share the concern for security and privacy with us, the assumptions, goals, and concerns are considerably different. Access control on value associations in XML has been studied in [14] and [16]. Their work is orthogonal to our concerns.

3. SECURITY ISSUES

In this section, we define the concepts of security constraints, attack model, and present several security definitions.

3.1 Encryption Scheme

Given an XML database D, we define an *encryption scheme* as an identification of those elements that are to be encrypted. Figure 2 shows an example database. The encryption scheme is illustrated by highlighting encrypted elements with boxes. We have choice in the elements that are to be encrypted. The choice is driven by what it is that the client wishes to protect from the untrusted server and by the desire to minimize the overhead imposed on the client for extracting query answers.

3.2 Security Constraints

We propose *security constraints* (SCs) as a means for the data owner to specify the information, i.e., the set of basic queries, that she intends to protect from an attacker. To be precise, a *security constraint* is an expression of the form p or of the form $p : (q_1, q_2)$, where p, q_1 and q_2 are XPath expressions. The SC p, called *node type constraint*, specifies that for every node x in the document D that p binds to, the element subtree rooted at x, consisting of its tag, its content, and structure, is considered as classified. The SC $p : (q_1, q_2)$, called *association type constraint*, says that for every node x that p binds to and for every pair of nodes y_1, y_2, containing data values v_1, v_2, that q_1, q_2 bind to in the context of x, the association between v_1 and v_2 is considered as classified.

EXAMPLE 3.1. **[Security Constraints]**
For the database in Figure 2, the client intends to protect the following information: (1) the insurance information of each patient, (2) which SSN number matches with which patient's name, (3) which patient has which diseases, and (4) which doctor treats which diseases. The security constraints are specified as follows:
SC_1: `//insurance`, to protect `insurance` elements,
SC_2: `//patient:(/pname, /SSN)`, to protect the association between patient's name and SSN number,
SC_3: `//patient:(/pname, //disease)`, to protect the association between patient's name and disease.
SC_4: `//treat:(/disease, /doctor)`, to protect the association between the name of doctors and diseases. ∎

Each SC can be seen as capturing a set of queries. A node type SC p *captures* queries of the form p, as well as queries p/a, $p//a$, $p/a[following-sibling :: b]$ etc. where a, b are element tags. The idea is that whenever p binds to an element x in the database D, whether these queries have a non-empty answer in the database D as well as the answers should be protected. E.g., in Figure 2, the SC `//insurance` captures the queries `//insurance`, `insurance/policy#`, `insurance//*/@coverage`, `//insurance//policy#`, etc.

An association SC of the form $p : (q_1, q_2)$ *captures* the queries $p[q_1 = v_1][q_2 = v_2]$ for all values v_1, v_2 such that the XPath expression above has a non-empty answer in the database D. E.g., the association SC `//patient:(pname, //disease)` captures the queries `//patient[pname=Betty][//disease=diarrhea]` and `//patient[pname=Matt][//disease=lukemia]`. Whether these queries have a non-empty answer has to be protected from the attacker.

For a SC σ and a query A captured by σ, we denote by $D \models A$ that A has a non-empty answer in the database D. Thus, for each query A captured by each SC, we need to protect the fact that $D \models A$. We implement protection of such facts from the untrusted server by encryption. Intuitively, enforcing the SCs requires encrypting certain elements in the database. E.g., the enforcement of the SC //`insurance` can be done by encrypting all `insurance` elements. The SC //`patient` : (`pname`, //`disease`) can be enforced by encrypting either `pname` elements or `disease` elements. Figure 2 shows an example of an encrypted database with all SCs in Example 3.1 are enforced.

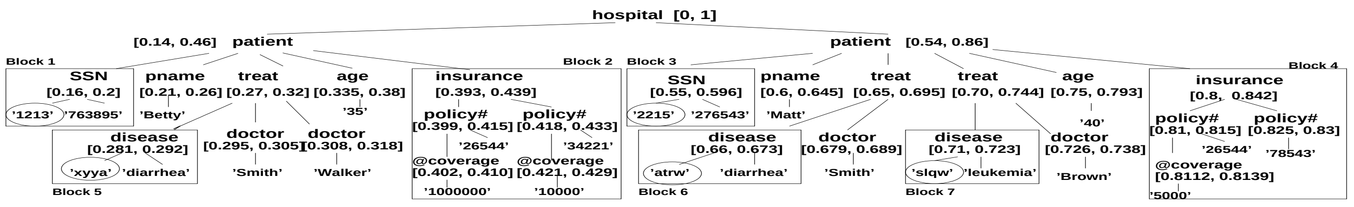


Figure 2: An Encrypted XML Health Care Database

3.3 Prior Knowledge and Attack Model

We assume the server/attacker may be curious and try to learn about the data it is hosting, but never modifies the data and always answers queries correctly. We consider the following two kinds of attacks by the server/attacker:

Frequency-based Attack The server may possess prior knowledge about the domain values and even the exact occurrence frequencies for the data values in the hosted data. By matching the plaintext and ciphertext values of the same frequency distribution, the attacker may crack the database. This is called *frequency-based attack*. The data distribution knowledge can be collected from either the encrypted database or the metadata. Thus, frequency-based attack may be based on either the encrypted database or the metadata. *We assume the server has no prior knowledge about either the tag distribution or the correlations of data values.*

Size-based Attack The attacker may have the common knowledge that the length of plaintext decides that of ciphertext. Thus if the attacker has a set of candidate plaintext databases for the hosted data, he will try to eliminate those candidate databases whose encrypted result is of different length from that of encrypted hosted data, and consequently increase the probability of the remaining candidates to be the hosted data. We call this attack *size-based attack*.

3.4 Security Definitions

We first define indistinguishability of databases w.r.t. the two possible attacks. Let $|D|$ denote the size of a database.

Definition 3.1. **[Indistinguishability of Databases]** Let $\mathcal{E}$ be an encryption function. Let D, D' be two plaintext XML databases. We say D and D' are *indistinguishable*, written D ~ D', if: (1) $|\mathcal{E}(D)| = |\mathcal{E}(D')|$, (2) For each attribute, D and D' have the same domain of values, and for each domain value v, the corresponding encrypted value in $\mathcal{E}(D)$ and $\mathcal{E}(D')$ has the same occurrence frequency. ▮

This definition ensures that the attacker cannot distinguish D and D' by either *size-based attack* (condition (1)) or *frequency-based attack* (condition (2)). Thus if there is a set of candidate databases including the true hosted database and they are all pairwise indistinguishable, we can say those candidate databases have equal probability to be the true hosted database.

Having a large set of candidate databases alone doesn't imply security. The attacker may have some prior belief probability of each candidate database to be the true hosted database. He may be able to infer something and increase the belief when looking at the encrypted hosted database along with the metadata on server. We then use *perfect security* as the security requirement for this case. Perfect security [11] is a term used in cryptography. An encryption scheme has *perfect security* provided the probability of obtaining the corresponding plaintext message M for a given encrypted (i.e., ciphertext) C, is the same as obtaining the plaintext without any ciphertext to observe, i.e., the attacker is unable to obtain any additional information from the ciphertext. We adapt this definition to databases.

Definition 3.2. **[Perfect Security]** Let $\mathcal{C}$ be an encrypted XML database and let $D_1, D_2, \ldots, D_n$ be the corresponding set of candidate plaintext databases, each with a *priori probability* Prob($\mathbf{D}$=D_i) of being the true hosted database. Let Prob($\mathbf{D}$=D_i|$\mathbf{C}$=$\mathcal{C}$) be the *posteriori probability* of D_i being the true hosted database given the encrypted hosted database $\mathcal{C}$. We say $\mathcal{C}$ is *perfectly secure* if Prob($\mathbf{D}$ = D_i|$\mathbf{C}$ = $\mathcal{C}$) = Prob($\mathbf{D}$=D_i), where $\mathbf{D}$ ($\mathbf{C}$) denotes the plaintext database (encrypted database). ▮

Based on the definitions of indistinguishability and perfect security, we define a *secure* encryption scheme:

Definition 3.3. **[Secure Encryption Scheme]** Given an XML database D, a set of security constraints Σ on D, an encryption scheme $\mathcal{S}$, and the ciphertext database $\mathcal{E}(D)$ encrypted by $\mathcal{S}$, we say $\mathcal{S}$ is *secure* on D if there exists a large number of databases D' such that: (1) D'~D, (2) $\forall$ security constraint $\sigma \in \Sigma$, and $\forall$ query A captured by σ, $D \models A$, but $D' \not\models A$, and (3) $\mathcal{E}(D)$ is perfectly secure. ▮

This definition requires that for a large set of candidate databases that don't contain any sensitive information (condition (2)), by looking at the encrypted hosted database alone, neither can the attacker distinguish them from each other or from the original hosted data (condition (1)), nor can he increase the probability of any of them to be the true hosted data (condition (3)). This definition formalizes what it means for an encryption scheme to enforce a given set of SCs on a database. We will show in Section 4 that our proposed encryption scheme is *secure*. The term "large" is intentionally left undefined in Definitions 3.3 and 3.4 (below). When proving security properties, we will actually show typically "large" means exponential in the size of the domain or the schema.

To facilitate efficient query processing, we need to place some metadata on the server. However, by looking at both the encrypted hosted data and the metadata on the server, the attacker should not be able to infer any sensitive information. The following definition addresses this.

Definition 3.4. **[Secure Database System]** For the XML database system DS(D, $\mathcal{C}$, $\mathcal{M}$), where D is the plaintext database, $\mathcal{C}$ is the encrypted version of D, and $\mathcal{M}$ is the metadata on D, we say DS is *secure* for a set of security constraints Σ if there exists a large number of database systems DS'(D', $\mathcal{C}$', $\mathcal{M}$') such that: (1) $\mathcal{M}$ = $\mathcal{M}$', (2) D~D', (3) $\forall$ security constraint $\sigma \in \Sigma$, and $\forall$ query A captured by σ, $D \models A$, but $D' \not\models A$, and (4) $\mathcal{C}$ is perfectly secure. ▮

This definition requires that for a large set of candidate databases that don't contain any sensitive information (condition (3)), by looking at the encrypted database as well as the metadata, the attacker cannot distinguish them from each other or from the original hosted data ((condition (1) and (2)), nor can he increase the probability of any of them to be the true hosted data (condition (4)). We will show in Section 5 that using our proposed metadata (consisting of structural and value indices), the resulting database system is secure.

Finally, we also consider attacker's possible inference when he observes query answering. To make this problem precise we consider the following predicates: Let A be any query captured by a security constraint. Let B denote an encryption block (i.e., an element that is encrypted) in $\mathcal{E}(D)$. Then we use $B(A)$ to denote that the block B satisfies the query A. Note that A may be structure-based or can be a value association. The question is whether for a specific block B, the server can learn whether or not $B(A)$ holds for any query captured by any of the SCs in Σ. More precisely, can the server's belief probabilities in these assertions $B(A)$ be improved after observing a series of queries and answers? *We assume the server has no prior knowledge about the query workload distribution.* We capture this concern in the form of a definition of secure query answering:

DEFINITION 3.5. **[Secure Query Answering]** *Let Σ be a given set of SCs, D a database, and $C = \mathcal{E}(D)$ the encrypted database obtained from D using an encryption scheme $\mathcal{S}$. Let M be the metadata placed on the server. Let A be any query captured by any SC in Σ and let $Bel(B(A))$ denote the attacker's belief probability that the proposition $B(A)$ holds for any encryption block B, after observing q queries and responses. Then we say the query answering by the server is secure if $Bel(B(A))$ does not increase after the attacker observes each additional query and response.* ▌

We will show in Section 6 that our query processing procedure is not only efficient but also secure.

4. ENCRYPTION SCHEME

Given an XML database D, it can be encrypted at any level of granularity, e.g., the whole document, chosen elements at any depth, or just the content (text values) of chosen elements. In choosing a granularity for encryption, there is a tradeoff between the data security and efficiency of query evaluation. Too coarse-grained encryption schemes, e.g., encrypting the whole database into one or very few blocks, can quash hopes of query optimization, while too fine-grained encryption schemes may allow an attacker to learn about the data, thus violating security requirements. In this section, we will first show how to construct a *secure* encryption scheme in Section 4.1. Then we will define the *optimal, secure encryption scheme* and discuss the complexity of finding an optimal, secure encryption scheme in Section 4.2.

4.1 Secure Encryption Scheme

Although the plaintext data is hidden after encryption, careless design of the encryption scheme still may allow an attacker to infer the plaintext data by frequency-based attack. E.g., assume the association between patient's age `40` and his disease `leukemia` in Figure 2 should be protected. If the client plainly encrypts each `disease` and `age` element individually, the encrypted value of `leukemia` will have the same number of occurrence as before encryption, which is also true for value `40`. Then by matching the frequency of occurrences of plaintext and ciphertext values, even both `disease` and `name` are encrypted, the attacker can easily identify the plaintext values and infer the classified association. *We assume only the distribution on the leaf nodes are known to the attacker.*

From the above example we can see that encrypting leaf nodes [1] individually is not safe. To address the security issue, we use the notion of an *encryption decoy*, which is similar to the *salt* technique in UNIX system. To be more specific, for an element e, an *encryption decoy* is a randomly generated data value d that is added as a child of e and then e and d are encrypted together. By the effect of decoy, for any two elements e, e', the result of encrypting them with their encryption decoy are not identical. E.g., we choose to encrypt the two `disease` nodes in Figure 2 of value `diarrhea` with decoy (the decoy values `xyya` and `atrw` are shown in ovals). By the effect of decoy the two `diarrhea` values will be encrypted into two distinct ciphertext values.

Let $\mathcal{S}$ be an encryption scheme, D a database, and Σ be a set of SCs. Let $\mathcal{E}(D)$ be the encrypted database according to $\mathcal{S}$, such that: (i) for every node type SC $p \in \Sigma$, p is encrypted, and (ii) for every association type SC $p : (q_1, q_2) \in \Sigma$, nodes that bind either to p/q_1 or to p/q_2 are encrypted, and (iii) every leaf element that is encrypted is encrypted with a decoy. We call $\mathcal{C} = \mathcal{E}(\mathcal{D})$ an *encrypted version* of D. We have:

THEOREM 4.1. **(Secure Encryption Scheme) :** *Let $\mathcal{S}$, D, Σ, $\mathcal{E}(D)$ be as above. Then $\mathcal{S}$ is a secure encryption scheme (Definition 3.3).* ▌

The correctness can be shown as follows. Assume for some leaf element/attribute that there are n distinct plaintext values in the original database D, each with an occurence frequency k_i. There will be correspondly m distinct ciphertext values, each with occurrence frequency 1, where $\Sigma_{i=1}^{n} k_i = m$. By simply matching the frequency of the ciphertext to that of plaintext does not help the server to crack the plaintext value. Each plaintext value can be mapped to a set of ciphertext values. The only possibility is to try out all such mappings. The number of mappings corresponds to the number of ways of partitioning m into n non-zero parts. This number is $N = \binom{m}{k_1} \times \binom{m-k_1}{k_2} \ldots \times \binom{m-k_1-\cdots-k_{n-1}}{k_n}$ $= (\Sigma_{i=1}^{k} k_i)! / \Pi_{i=1}^{n} k_i!$, which is exponential in $\Sigma_{i=2}^{n} k_i$ (proof is given in [26]), and thus satisfies the "large" requirement in Definition 3.3. E.g., for $k_1=3$, $k_2=4$, $k_3=5$, $N = (3+4+5)! / (3! \times 4! \times 5!) = 27720$. This is the number of candidate databases the attacker has to consider, where we have only taken one leaf element/attribute into account. Notice that for each of the latter databases D', $\mathcal{E}(D')$ has the same frequency distribution and the same size as $\mathcal{E}(D)$. Thus $D' \sim D$ (condition (1) of Definition 3.3). Out of those N candidates, there is only one that contains the sensitive nodes and sensitive associations (condition (2) of Definition 3.3). Finally, for those N candidates D_i ($1 \leq i \leq N$) and the given encrypted hosted database C, Prob($\mathbf{D} = D_i | \mathbf{C} = \mathcal{C}$) = Prob($\mathbf{C} = \mathcal{C} | \mathbf{D} = D_i$)$\times$ Prob($\mathbf{D} = D_i$)/ Prob($\mathbf{C} = \mathcal{C}$). Assume the encryption keys that the attacker uses are of length s. Since the keys are uniformly distributed, Prob($\mathbf{C} = \mathcal{C} | \mathbf{D} = D_i$) = 2^{-s}. Then Prob($\mathbf{C} = \mathcal{C}$) = Σ_{D_i}(Prob($\mathbf{D} = D_i$) $\times$ Prob[$\mathbf{C} = \mathcal{C} | \mathbf{D} = D_i$]) = $2^{-s} \times \Sigma_{D_i}$Prob($\mathbf{D} = D_i$). Since Σ_{D_i}Prob($\mathbf{D} = D_i$) = 1, we have Prob($\mathbf{C} = \mathcal{C}$) = Prob($\mathbf{C} = \mathcal{C} | \mathbf{D} = D_i$)$=2^{-s}$. Thus the encrypted database C is perfectly secure (condition (3) of Definition 3.3). In sum, $\mathcal{S}$ is secure.

4.2 Optimal Secure Encryption Scheme

[1]In this paper we assume that the data values in the XML documents are only attached to the leaf nodes and do not consider mixed content.

For a given set of security constraints, there may be multiple secure encryption schemes. Using the database in Figure 2 as an example, to protect the `disease` data, one of the secure encryption schemes is to encrypt the whole document as a block. Some other possible encryption schemes are: encrypt individual `treat` elements without decoy, or encrypt individual `disease` node with decoy. Different encryption schemes have different query processing performance with regard to the cost of transmission, decryption and query post-processing.

To address the efficiency issue of query processing, we then define the *optimal* (and *secure*) encryption scheme. Notice that each encryption scheme can be viewed as a set of encryption blocks. Thus, we abuse the notation and write $b \in \mathcal{S}$, for an encryption block b. The size of an encryption block b is the number of nodes in it, including any decoy elements. The size of an encryption scheme $\mathcal{S}$ is the sum of sizes of its encryption blocks, i.e., $\mid \mathcal{S} \mid = \sum_{b \in \mathcal{S}} \mid b \mid$.

Definition 4.1. **[Optimal Secure Encryption Scheme]** Let Σ be a set of user specified SCs on a database D. Then an *optimal secure encryption scheme* is a secure encryption scheme $\mathcal{S}$ that is of the smallest size on D, i.e., $\mid \mathcal{S} \mid \leq \mid \mathcal{S}' \mid, \forall \mathcal{S}'$ that are secure on D. ▌

Optimal secure encryption scheme is not unique. E.g., to enforce the security constraints specified in Example 3.1 on the instance shown in Figure 2, besides encryption on two `insurance` nodes individually, we can encrypt two `pname` nodes with decoy and three `disease` nodes with decoy, or we can encrypt two `SSN` nodes with decoy and three `disease` nodes with decoy. Both of them has the same size. We next address the complexity of finding an optimal secure encryption scheme.

Theorem 4.2. **(Complexity of Optimal Secure Encryption) :** Given an XML database D and a set of SCs Σ, finding an optimal secure encryption scheme for Σ on D is NP-hard. ▌

The complexity is in the size of SCs. The proof is by reduction from VERTEX COVER, an NP-complete problem [25] and is given in [26].

Given the NP-hardness result, we seek approximations. Indeed, weighed vertex cover has several approximation algorithms (see Motwani [25]). In principle, we can adapt any of them to devise an algorithm for finding an encryption scheme whose cost is no worse than twice the optimal cost.

5. METADATA ON SERVER

To enable efficient query processing, we propose to add some metadata on the hosted data at the server. Our proposal consists of two parts: a structural index and a value index. In this section, we will present techniques for these two indices and show that when using a secure encryption scheme enforcing given SCs, in the presence of the proposed indices, the database system as a whole is secure.

5.1 Structural Index

Recall that the server stores encrypted blocks, corresponding to subtrees of the XML data, in addition to unencrypted data, as determined by the chosen encryption scheme (e.g., see Figure 2). We want the server to be able to efficiently locate the encryption blocks as well as unencrypted data nodes satisfying user query without the server who has access to the data and metadata learning any sensitive information.

```
Function callInterval()
Input: Node p with interval [min, max], weight $w_i^1$, $w_i^2$
Output: Interval of p's ith child $c_i$, $[min_i, max_i]$
Assume p has N children.
$d = (max - min)/(2 \times N + 1)$
$min_i = min + (2 \times i - 1) \times d - d \times w_i^1$
$max_i = min + 2 \times i \times d + w_i^2 \times d$
```

Figure 3: Calculating DSI Index

We propose a novel index scheme, called *discontinuous structural interval (DSI) index*,[2] as an effective way to index tree-structured data. The DSI index for a database tree D is obtained as follows. The root is assigned the interval $[0, 1]$. Inductively, children of an internal node p are assigned subintervals of p's interval, but *gaps* are introduced between the intervals of (1) p and its first child, (2) adjacent children of p, and (3) the last child of p and p itself. We differentiate the gap length by assigning two arbitrary weights w_1 and w_2 (known only to the client), where $w_1, w_2 \in (0, 0.5)$ is a real number. The weights are generated at random before assigning an interval to a node. Then, we calculate the interval of every node by using the algorithm shown in Figure 5.1, where we assume we know node p's interval $[min, max]$ and calculate its child intervals. In general, d=(max - min)/(2N + 1)(where N = number of p's children), $max_i = min_i + (1 + w_i^1 + w_i^2)d$, and $min_{i+1} = min_i + (2 - w_{i+1}^1 + w_i^1)d$. The gap is $min_{i+1} - max_i = (1\text{-}w_i^2\text{-}w_{i+1}^1)d > 0$. In particular, the lower bound of the first child is more than that of its parent, i.e., $min_1 > min$, and the upper bound of the last child is less than that of the parent, i.e., $max_N < max$.

XPath axes *descendant*, *following*, *following-sibling* (and their symmetric counterparts) are all computed efficiently just as using a regular (continuous) interval index. The *child* axis is computed using $\text{child}(x, y) \leftrightarrow \text{desc}(x, y)$ $\wedge \nexists z : \text{desc}(x, z) \wedge \text{desc}(z, y)$.

5.1.1 Representing DSI index

DSI Index Table We represent the DSI index using the *DSI index table*. It stores the mapping between the tags (in encrypted format, if the element was encrypted) and their DSI index entries. We chose one-time-pad encryption scheme (a.k.a. Vernam cipher) [15] for the tag encryption, because of its perfect security property. Note that simply storing *all* intervals reveals too much information to the server. Thus we do the following: for those adjacent nodes that are of the same tag and are encrypted in the same block, we group their intervals into one by using the lower bound of the left-most node as the lower bound, and the upper bound of the right-most node as the upper bound. For example, two adjacent nodes "policy#" in block 2 (Figure 2) will be represented by a single interval [0.399, 0.433]. Figure 4 (b) shows the DSI index table of the instance in Figure 2.

Note that if the same grouping technique is applied on a set of sibling nodes when a *continuous* indexing scheme such as in [4] is used, the grouping may cause discontinuity of the index and the server consequently may find out the existence of grouping, and further possibly the exact structure of the tree. An example to show such information leakage on the continuous index is shown in [26]. Compared with continuous index, by looking at the indices in the DSI interval table, the server cannot decide whether there exist groupings be-

[2]As contrasted with the well-known continuous interval indexing scheme [4].

hind the intervals, or how many elements have been grouped together. E.g., by looking at the entry "U84573" with DSI index [0.16, 0.2] in *DSI index table* (Figure 4 (b)), which corresponding to the `SSN` node in the block 1 in Figure 2, the server cannot decide whether [0.16, 0.2] represents one node or multiple nodes that have been grouped together.

Encryption Block Table: In addition to the DSI index described above, we also keep auxiliary information on the encrypted blocks. Specifically, for each encrypted block, which is essentially a subtree, we call the interval of the subtree root, the *representative interval* of the block. Thus, we maintain a mapping between representative intervals and block IDs. The block IDs can be viewed as a pointer to the physical location of the encrypted blocks in the database. This mapping is also represented as a table, called the *encryption block table*. E.g., for our running example of Figure 2, the DSI index table as well as the encryption block table is shown in Figure 4.

ID	Representative Interval
1	[0.16, 0.2]
2	[0.393, 0.439]
3	[0.55,0.596]
...	...

(a) Encryption Block Table

Tag	DSI Index
U84573	[0.16, 0.2], [0.55, 0.596]
patient	[0.14, 0.46] [0.54, 0.86]
...	...

(b) DSI Index Table

Figure 4: Meta Data

In sum, the structural index at the server consists of the DSI index table and the encryption block table. We have the following theorem:

THEOREM 5.1. **(Security of Structural Index) :** Let DS(D, $\mathcal{C}$, $\mathcal{SI}$) be a database system, where D is an XML database, $\mathcal{C}$ is the encrypted version of D, and $\mathcal{SI}$ is the structural index of D. Then DS($\mathcal{D}$, $\mathcal{C}$, $\mathcal{SI}$) is *secure* (Definition 3.4). ▮

For lack of space, we sketch the proof here (detailed proof can be found in [26]). We assume there are m encryption blocks in $\mathcal{C}$, with each encryption block containing n_i leaf nodes that are represented by k_i intervals ($n_i \geq k_i$, $1 \leq i \leq m$). Then we prove there are $\Pi_{i=1}^{m}\binom{n_i-1}{k_i-1}$ -1 database systems DS'(D', $\mathcal{C}$', $\mathcal{SI}$') s.t. DS' satisfies the conditions in Definition 3.4. Let B_i be the ith encryption block that is obtained by encrypting the subtree t rooted at node x. Assume there are n_i leaf nodes in B_i represented by k_i intervals in the DSI index table, with each interval representing (a group of) m_i leaf nodes. Then there are $\binom{n_i-1}{k_i-1}$ assignments of values for $m_i (1 \leq i \leq k_i)$ such that $\Sigma_{i=1}^{k_i} m_i = n_i$. E.g., for n_i=7 and k_i=3, we can have 7=1+1+5, =1+2+4, =2+3+2, ..., – 15 possible assignments to assign 7 leaf nodes to 3 intervals (Figure 5 shows three such possibilities), with each assignment representing a possible structure of the subtree. As shown in Figure 5, the DSI intervals of each possible subtree are identical (condition (1) of Definition 3.4). Furthermore, the encryptions of those 15 possible subtrees are indistinguishable since they have the same size (condition (2) of Definition 3.4). Out of those 15 candidates there is only one true database that contains the sensitive nodes (condition (3) of Definition 3.4). In general, for m encryption blocks we have $\Pi_{i=1}^{m}\binom{n_i-1}{k_i-1}$ -1 such candidates in total, where the number $\Pi_{i=1}^{m}\binom{n_i-1}{k_i-1}$ is exponentially "large" in k_i, the number of intervals in the encryption block B_i, E.g., when n=15, k=5, $\binom{n-1}{k-1}$=1001. The proof that each candidate database satisfies the perfect security (condition (4) of Definition 3.4) is similar as the proof in Theorem 4.1.

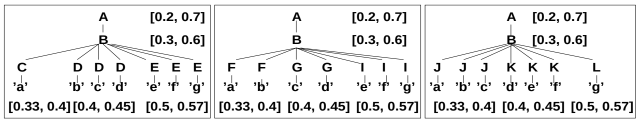


Figure 5: Possible Structure of Node A

5.2 Value Index

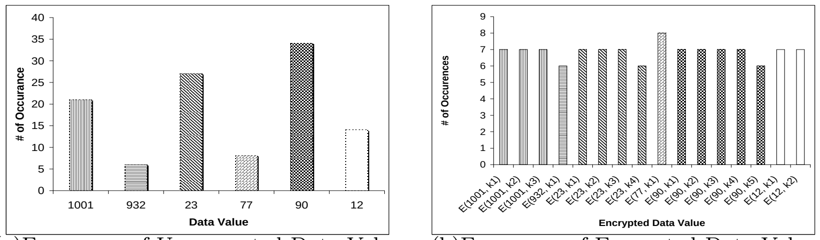
For value indexing, we choose B-trees so as to support range queries. Each data entry of the B-tree will be of the form $\langle evalue, Bid \rangle$, where $evalue$ is the encrypted version of the value from the database and Bid is the ID of the encrypted block containing an occurrence of $evalue$.

Simply encrypting the data values that we want to index on and storing the ciphertext in the B-tree will result in encrypted values following the same distribution as the original plaintext values, opening the door fo frequency-based attack. To mitigate this, we advocate "splitting" each plaintext value into one or more ciphertext values in such a way that regardless of the original distribution, the target distribution remains flat, i.e., uniform. Splitting is not enough to guard against frequency-based attack, so we further use "scaling" on the split data so that the attacker cannot uniquely crack the identity of ciphertext values with the aid of the data frequency knowledge. The details are discussed in this section. Note that the "splitting" and "scaling" techniques are applied on the metadata, while the "encryption decoy" (Section 4.1) is applied on the database.

Agrawal et al. [3] proposed a method of designing order preserving encryption functions such that the distribution of the encrypted values follows an arbitrarily chosen target distribution, regardless of the input distribution. However, the input distribution considered in their paper is a combination of histogram-based and parametric techniques, while we consider a more conservative model where the attacker knows the exact occurrence frequencies of domain values. To protect the data against *frequency-based attack*, we *map the same plaintext values to different ciphertext values*, with the target domain size becoming larger than the input domain size. We describe our approach next.

5.2.1 Splitting and Scaling Values

Let A be an attribute or atomic element being encrypted. Suppose there are k distinct values $\{v_1, ..., v_k\}$ in the (active) domain whose numbers of occurrences are $\{n_1, ..., n_k\}$. Without loss of generality, suppose $v_1 < v_2 < \cdots < v_k$ (alphabetical ordering is used for categorical domain). To map each v_i to multiple ciphertext values such that they have similar number of occurrences, we look for 3 consecutive positive integers, $m-1$, m, and $m+1$, such that each n_i can be expressed in a form: $n_i = k_1^i \times (m-1) + k_2^i \times m + k_3^i \times (m+1)$, where k_1^i, k_2^i, and k_3^i are non-negative integers. The intuition is to map a data value with n_i occurrences to k_1^i + k_2^i + k_3^i distinct encrypted values, each with $m-1$, m or $m+1$ occurrences. Thus in the distribution of the encrypted values, the frequency of each value will be nearly uniform for the server/attacker. E.g., Figure 6(a) shows an input where values follow a somewhat skewed distribution. This distribution is transformed into a nearly flat distribution (Figure 6(b)) where every frequency is $6, 7$, or 8. E.g., the input value "90" with number of occurrences 34 (34=1*6+4*7+0*8), is encrypted into 1+4+0 = 5 different



(a)Frequency of Unencrypted Data Value (b)Frequency of Encrypted Data Value

Figure 6: Data Distribution before Encryption & after Encryption

encrypted values in Figure 6(b) by using 5 distinct keys, each distinct encrypted value having the number of occurrences 6 or 7.

Notice that the existence of the triple $(m-1, m, m+1)$ is always guaranteed. Ignore the case where some values have just 1 occurrence, as this case is dealt with below. Then (2,3,4) is a triple using which any set of numbers greater than 1 can be expressed as a linear combination as above. We choose the maximum value of m for which the triple $m-1, m, m+1$ works, so intuitively the number of keys needed is reduced. Once the triple $m-1, m, m+1$ is chosen, for n distinct plaintext values, $K = max_{1\leq i\leq n}\{k_1^i + k_2^i + k_3^i\}$ is the maximum number of keys needed for encryption. For those values v_i with just 1 occurrence, if the chosen chunk sizes are $(m-1, m, m+1)$, then we split v_i into m values. In this case, the number of keys required would be m, which in general may be larger than K, the maximum number of keys needed as determined above.

A key requirement of this encryption is that the ciphertext values corresponding to different plaintext values should never straddle each other. More precisely, we require (*): for any two values $v_i < v_j$, and for any ciphertext values c_i^m, c_j^n associated with v_i, v_j respectively, it is necessary that $c_i^m < c_j^n$. We describe a method for achieving such an encryption. Let enc denote any order-preserving encryption function, such as was proposed by [3]. Choose K distinct random numbers $w_1, ..., w_K \in (0, 1/(K+1))$. Without loss of generality, let $w_1 < w_2 < \cdots < w_k$. Let there be n distinct plaintext values ordered as $v_1, v_2, \ldots, v_n$. Let $\delta = max_{1\leq i\leq n-1}(v_{i+1} - v_i)$. Let n_i be the number of occurrences of value v_i in the original database. Recall that out of the n_i occurrences, chunks of size $m-1$, m, or $m+1$ need to be mapped to a given encrypted value. Then we transform the first chunk of occurrences of v_i to $enc(v_i + w_1\delta)$. More generally, we map the n-th chunk of occurrences to $enc(v_i + (\sum_{1\leq j\leq n} w_j)\delta)$. In other words, the n-th chunk is displaced from v_i by a fraction of the gap δ given by the sum $w_1 + \cdots + w_n$. The result is encrypted using an order-preserving encryption function. It is straightforward to show that the resulting encryption satisfies the required condition (*) above. Splitting of a ciphertext value c into multiple values can be easily implemented by adding randomly chosen low order bits to c.

As an example, suppose two successive values are 23 and 32 and that we have determined $K = 4$. Say 23 was broken into $23 = (2\times 5) + (1\times 6) + (1\times 7)$. Notice that the sum of coefficients is $2+1+1 = 4$. The chunk sizes are $5, 6, 7$. The gap is $\delta = 32-23 = 9$. Suppose the chosen random numbers are $w_1 = 0.05, w_2 = 0.1, w_3 = 0.05, w_4 = 0.02$. Then 23 would be displaced to each of the values – $23 + 0.05\times 9 = 23.45$, $23.45 + 0.1\times 9 = 24.35$, $24.35 + 0.05\times 9 = 24.80$, $24.80 + 0.02\times 9 = 24.98$. The displaced values would then be encrypted using some order preserving encryption function. The value 32 would be handled in a similar fashion.

If the domain is not real or rational, then we map it to such a domain. The client keeps the mapping between categorical values and natural numbers.

A key property of the splitting technique described so far is that the total number of occurrences of any value is unchanged by the transformation. That is, $\sum_{1\leq i\leq k} n_i = \sum_{1\leq j\leq \ell} f_j$, where n_i (f_j) are the occurrence frequencies of plaintext (ciphertext) values, and k (ℓ) is the number of distinct plaintext (ciphertext) values. An attacker with an exact knowledge of occurrence frequencies of input values can group adjacent ciphertext values together until they match a particular occurrence frequency of an input value. We use "scaling" to defend against such attack. Let s_i be a randomly chosen scale factor. We typically want to use a small real number in the range $[1, 10]$ since the index size is affected by the scale factor. When we split a value v_i into ciphertext values $c_i^1, ..., c_i^n$, we replicate each index entry corresponding to c_i^j, s_i times. Note that all s_i replicas of the index entry for c_i^j will point to the same encryption block. The scale factor s_i is randomly chosen independently for each value v_i. The ciphertext value distribution is not uniform any more. However, without knowing the scaling factor used, it is not possible for the attacker to crack the identity of ciphertext values. The price we pay is of course that instead of K keys we now have $2K$ keys (K w's for splitting and K s's for scaling).

Theorem 5.2. **(Security of Value Index) :** Let DS(D, $\mathcal{C}$, $\mathcal{VI}$) be a database system, where D be an XML database, $\mathcal{C}$ be the encrypted version of D, and $\mathcal{VI}$ be the value index of D. Then DS(D, $\mathcal{C}$, $\mathcal{VI}$) is *secure* (Definition 3.4). ▮

We illustrate the key proof idea with an example. Suppose there are three plaintext values v_1, v_2, v_3 ($k = 3$) with occurrence frequencies $30, 10, 20$ respectively. Suppose we decided to map v_1, v_2, v_3 to six ciphertext values $c_1, c_2, \ldots, c_6$ (n=6), where each c_i has occurrence frequency of 30. Then each of the following encryptions and scaling could give rise to the observed target distribution: (1) $v_1 \mapsto c_1$ (scale=1), $v_2 \mapsto c_2$ (scale=3), $v_3 \mapsto \{c_3, ..., c_6\}$ (scale=6), (2) $v_1 \mapsto c_1$ (scale=1), $v_2 \mapsto \{c_2, c_3\}$ (scale=6), $v_3 \mapsto \{c_4, c_5, c_6\}$ (scale=4.5), ..., (10) $v_1 \mapsto \{c_1, ..., c_4\}$ (scale=4), $v_2 \mapsto c_5$ (scale=3), $v_3 \mapsto c_6$ (scale=1.5). The number of mappings together with appropriate scalings is determined by the number of ways of partitioning the set $\{v_1, ..., v_n\}$ into k non-empty subsets $\langle S_1, ...S_k\rangle$, in an order-preserving way, i.e., $\forall v_i \in S_i, \forall v_j \in S_j$, $i \neq j$, we have $v_i < v_j$. This number is $\binom{n-1}{k-1}$. Each of each of these mappings (with appropriate scalings) might give rise to the observed ciphertext distribution. Out of these, only one mapping/scaling combination correctly determines the identity of the ciphertext values. We use this idea to construct databases D' where the values v_1 and v_2 both occur but they do not occur as an association, i.e., there is no matching of the XPath expression `p[q1=v1][q2=v2]` in D'. For those $\binom{n-1}{k-1}$ databases D', they have the same value index (condition (1) of Definition 3.4), same size (after encryption), and same exact occurrence frequency distribution for values as D. Thus $D \sim D'$ (condition (2) of Definition 3.4). However, only one database contains the sensitive associations (condition (3) of Definition 3.4). The number $\binom{n-1}{k-1}$ is exponentially "large" in k, the number of distinct plaintext values of attribute A. E.g., when n=15, k=5, $\binom{n-1}{k-1}$=1001. The proof that each candidate database satisfies the perfect security is similar to the proof of Theorem 4.1 (condition (4) of Definition 3.4). The details are omitted for lack of space but can be found in [26].

Note that by using different encryption techniques for the database and for the value index, we can ensure their encrypted value sets are disjoint, so by accessing them together, the attacker cannot infer any useful information he otherwise could not.

This theorem shows that by collecting the distribution of the encrypted data from the B-tree index, the server cannot infer any association (or atomic query) that would violate any SC $p : (q_1, q_2) \in \Sigma$. The price of this protection is that the size of the B-tree index is more than it would be for an unencrypted database, The increase in size is proportional to the scaling used. Secondly, because of splitting, aggregate queries involving COUNT cannot be evaluated without decryption, although queries involving MAX/MIN can still be evaluated correctly without decryption. Details of query processing are discussed in Section 6.

We refer to the particular encryption technique described here as *order preserving encryption with splitting and scaling* (OPESS).

6. QUERY PROCESSING

The major steps in query answering are as follows: (1) The client translates the query Q into Q_s by replacing the tags and data values individually with their encrypted format, whenever the tag/value was encrypted in $enc(D)$. (2) The server labels each (encrypted) query node with its DSI index entry retrieved from the *DSI index table* and computes structural joins, which prune index entries at query nodes. After the pruning, the remaining indices represent the nodes that satisfy the structural predicates of the query. (3) For the value-based constraints in the query, the server looks up the B-tree and returns the intervals (index entries) that represent the encryption blocks that meet the value constraints. (4) The server then "joins" the result of step 2 with the results of step 3 and obtains a set of unencrypted nodes, or a set of block IDs if the output node is encrypted in some blocks. The corresponding nodes or the encrypted blocks satisfy both structural and value constraints of the query and are returned to the client. (5) The client decrypts the received encrypted blocks, removes the decoy if any, and applies the original query Q on the remaining decrypted result. These steps are elaborated below. We use the query shown in Figure 7 (b) as a running example.

6.1 Query Translation at the Client

We encrypt the tags and value constraints individually in the query while still preserving the query structure. Recall that the constraint graph (used in the proof of Theorem 4.2) records the nodes that are encrypted. For any query node that has the same tag as one of those nodes, its tag will be encrypted by Vernam cipher scheme [15] with the same keys used for the construction of *DSI index table*. Furthermore, if such query node has a value-based constraint, the value will be encrypted by the same encryption function and the same keys as used for the construction of the B-tree index. We make use of the nice property of OPESS functions in translating value constraints in the query. The details of the transformation is shown in Figure 7(a). Note that data values with multiple occurrences and unique occurrence are both translated in the same fashion. Figure 7(b) shows an example of query translation on the client based on the instance shown in Figure 2. In this example the `patient` node is kept unencrypted, while `insurance` and `coverage` nodes are replaced with their encrypted formats.

6.2 Query Translation at the Server

After receiving the query Q_s from the client, the server answered the query by the following steps:

Step 1: Translation of Query Structure: The server first obtains index entries associated with each query node type by consulting the DSI index table. Note that some nodes may have a tag which is encrypted. E.g., the intervals [0.16, 0.2] & [0.55, 0.596] will be returned by retrieving "U84573" (the encrypted value of query node "SSN") from the *DSI index table* (Figure 4(b)). We call the interval(s) of the distinguished (i.e., query answer) node distinguished interval(s).

Once query nodes have lists of intervals associated with them, the server computes any of the standard structural join algorithms to prune away intervals that do not match structural constraints of the query. The remaining intervals represent tuples of nodes that satisfy the structural constraints of the query.

Step 2: Translation of Value-based Constraints: Each value-based constraint can be treated as a triple <*tag*, *op*, *value*>, where *op* $\in <, >, \leq, \geq, =\}$. Here *tag* and *value* are both in encrypted format. Each value-based constraint is translated as a range query. This is necessary because of the splitting that was applied before encryption. Figure 7(a) shows how each value-based constraint is translated into corresponding range query on the B-tree index. The server retrieves the index entries from a B-tree index, corresponding to entries that match the triple. Recall that these index entries correspond to encrypted blocks. So for the value-based constraint `@coverage`>`100000`, the server retrieves the entry pointing to block 2.

Step 3: Obtaining Final Results: Step 1 yields index entries (intervals) corresponding to the distinguished node of the query, while step 2 yields index entries associated with encrypted blocks. In this final step, the server computes a final set of structural joins to determine the subset of encryption block id's that contain the distinguished node intervals. The resulting set of block id's are used to fetch the blocks from the encrypted database $enc(D)$, which are then shipped to the client. In our example, block 1 will be returned to the client.

6.3 Security of Query Answering

E: *an order-preserving encryption function;*
$w_1, w_2, \ldots, w_n$: $w_i < w_{i+1}$
$v = v_1 \Rightarrow v \geq E(v_1 + w_1 * \delta)$ and $v \leq E(v_1 + (\sum_{1 \leq j \leq n} w_j)\delta))$;
$v < v_1 \Rightarrow v < E(v_1 + w_1 * \delta)$;
$v > v_1 \Rightarrow v > E(v_1 + (\sum_{1 \leq j \leq n} w_j)\delta))$;
$v \leq v_1 \Rightarrow v \leq E(v_1 + (\sum_{1 \leq j \leq n} w_j)\delta))$;
$v \geq v_1 \Rightarrow v \geq E(v_1 + w_1 * \delta)$;

(a) Translation of Value-based Constraints

```
Original query Q: //patient [.//insurance//@coverage≥'10000']//SSN
Translated query Q': //patient[.//X95SER// TYOPOA≥764398]//U84573
''U84573'':  encrypted value of tag ''SSN'';
''X95SER'': encrypted value of tag ''insurance'';
''TYOPOA'': encrypted value of tag ''coverage'';
''764398'':  the maximum of the encrypted value of value ''10000''
```

(b) Example: Query Translation on Client

Figure 7: Query Translation on Client

We have shown our encryption scheme is secure (Theorem 4.1). We also have shown that by adding metadata on server side, the metadata is secure (Theorems 5.1 and 5.2). The next question is what if an attacker gets to observe client queries and server responses. Could he infer any information that would violate the SCs? We have:

THEOREM 6.1. **(Secure Query Answering) :** The query answering on the server is *secure* (Definition 3.5). ▮

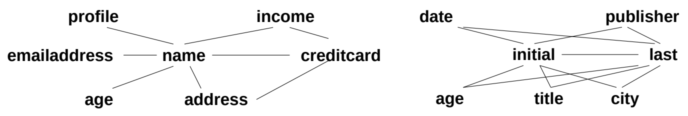
The proof for the belief probability associated with queries captured by node type security constraints $//a$ is based on the perfect security property of the Vernam cipher [15] we chose for the tag encryption in the *DSI index table* and the query translation. Since each element with tag a is encrypted by Vernam cipher scheme, due to the latter's perfect security property, the server cannot improve the belief probability of $B(Q)$ where Q is a query captured by the SC, by observing client queries and server answers.

The proof for the protection on association type security constraints is based on our design of OPESS encryption technique. For the SC $//a : (b_1, b_2)$, the values of at least one of the b_1, b_2 should be encrypted in the instance. Without loss of generality, we assume b_1 is encrypted. Let k (n) be the number of distinct plaintext (ciphertext) values of b_1 ($n > k$). Then when the server processes the query $p[//b_1=v_1][//b_2=v_2]$ for the first time, in which v_1 is encrypted while v_2 is not, the server's belief that v_1 is associated with v_2 is changed from $1/k$ to $1/\binom{n-1}{k-1}$ (the reason why we get $\binom{n-1}{k-1}$ can be found in the proof of Theorem 5.2). Since $\binom{n-1}{k-1} \geq k$ (because $\binom{n-1}{k-1} > \binom{n-1}{1} \geq k$), the belief of $B(p[//b_1=v_1][//b_2=v_2])$ that is changed from $1/k$ to $1/\binom{n-1}{k-1}$ is not increasing. For the following similiar queries, the probability of v_1 is associated with v_2 will stay at $1/\binom{n-1}{k-1}$.

6.4 Query Post Processing

The query post-processing is straightforward: upon receiving the result blocks from the server, the client first decrypts them if needed. If there exists the encryption decoy, the decoy is removed and the client applies the original query to the remaining data. For our example, the client decrypts block 1 that is returned by the server, removes the decoy "1213", and applies the original query Q on the remaining decrypted result. The query result is exactly the same as applying Q on the unencrypted database. It is straightforward to see that this yields the same answer as Q applied to the original database D.

A final note is that if the query involves aggregation involving the functions MIN/MAX, we can still evaluate them without decrypting data. However, if it involves other aggregate functions, the values would have to be decrypted by the client and then evaluated. Compared to [3], we are unable to process COUNT queries without decryption. But unlike their scheme, our scheme assumes conservatively that the server/attacker knows the exact occurrence frequencies of domain values.



(a) XMark Database (b) NASA Database

Figure 8: Constraint Graphs

7. EXPERIMENTAL EVALUATION

We ran a battery of experiments to measure the query evaluation performance on both client and server sides and explored various factors that impact the query performance, e.g., various encryption schemes, query size, etc.. In this section, we describe our experiments and provide an analysis of our observations.

7.1 Experimental Set up

Setup: We used one *sparc* workstation running SunOS 5.9 with 8 processors each having a speed of 900MHz and 32GB of RAM up as a *server*, and one *w*orkstation running Linux version 2.6.5 with 1 processor having a speed of 900MHz and 512MB of RAM up as a *client*. The communication speed between these two machines is 100Mbps.

Data sets: We used both synthetic and real data sets for the experiments. The synthetic data we used is from the *XMark* benchmark[3]. The real dataset is the *NASA* database from the XML data repository of University of Washington[4].

Security Constraints: We defined a set of security constraints on both *XMark* database and *NASA* database, which are represented by the *constraint graph* (which has a node for every tag appearing in the SCs and an edge representing every association type SC).

Various Encryption Schemes: To study the influence of different encryption schemes on the query processing performance, we designed the following four schemes for the experiments:

- *Optimal* (or *opt*) scheme: the document is encrypted by the optimal secure encryption scheme. The optimal encryption scheme encrypt nodes `name` and `creditcard` on *XMark* database, and encrypt nodes `initial` and `last` on *NASA* database.
- *Approximate* (or *app*) scheme: the document is encrypted by the scheme obtained by the approximation algorithm by Clarkson ([10]). The approximation encryption scheme encrypt nodes `name`, `income` and `address` on *XMark* database, and encrypt nodes `initial`, `date`, `publisher`, `age`, `title` and `city` on *NASA* database.

[3]http://monetdb.cwi.nl/xml/
[4]http://www.cs.washington.edu/research/xmldatasets/www/repository.html

- *Sub* scheme: the document is encrypted on those nodes that are parents of the encrypted node in scheme 1.
- *Top* scheme: the whole document is encrypted as one block.

Query Set: We created three kinds of queries for each encrypted document: (1) Q_s, the queries output the children node of the root of the document, (2) Q_m, the queries output the nodes on the $[h/2]$ level, where h is the depth of the document tree, and (3) Q_l, the queries output the leaf nodes. For each category of queries, we creates 10 queries and reported the average.

We used the *XQEngine* engine for query evaluation. All values reported are the average of 5 trials after dropping the maximum and minimum for different workloads.

7.2 Division of Work between Client and Server

We measured the following parameters for each query: query translation time on client, query translation time on server, query processing time on server, transmission time of the answer from server to client, decryption time on client, and query post-processing time on client. Since we ran the experiment on a fast-speed network, the transmission time is negligible comparing with other time factors. The query translation time on client and server are both negligible too: even for document size of 50MB and the query of 20 nodes, the translation time on client is less than 5ms and the query translation time on server is around 13ms, which is only 1/3000 of query processing time by server. Thus we mainly studied query evaluation time on server, decryption time on client, and the query post-processing time on client. From the performance result of *NASA* database shown in Figure 9, we observe that the decryption cost is always the largest among the three measured factors. We also notice that the query processing time on server is always larger than that on client. This is because: (1) the encrypted data on the server is always of larger size than the decrypted data on the client, and (2) the whole dataset is used for the query processing on the server, while only the relevant data is used on the client. Similiar results are observed on *XMark* database.

7.3 Our Approach VS. Naive Method

We compare the performance of our techniques with the naive method, which is that the server sends the whole encrypted document to the client, the client decrypts it and applies the query on the decrypted result. We ran the experiments on both *XMark* and *NASA* databases encrypted by various encryption schemes, and observed that our method reached a substantial saving ratio: for both *XMark* database and *NASA* database on *opt*, *app* and *sub* encryption schemes, the query evaluation time by our technique is only 11% - 28% of that by the naive method, while *top* scheme has the same performance as naive method. For space issue, the experiment results are omitted.

7.4 Effect of Various Encryption Schemes

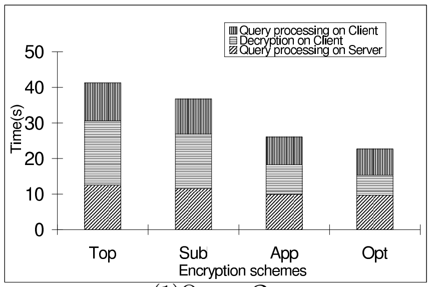
Firstly, we measured how the encryption time and the size of the encrypted document varies on both *XMark* and *NASA* databases for various schemes. The results show that *app* scheme takes the longest time for encryption since the number of elements that are encrypted by *app* scheme is the largest out of the four schemes. The results also show that the encrypted document by *sub* scheme is of the largest size. This is because compared with *top* scheme that encrypts the root element of the document, there are thousands of elements that are encrypted by *sub* scheme, each with extra encryption information (e.g., the `EncryptionType`, `EncryptionMethod` elements defined by W3C's XML encryption standard) added. On the other hand, compared with *app* and *opt* schemes, the *sub* scheme has similiar number of elements that are encrypted, with each encryption block of larger size than that of *opt* and *app* scheme. We also observe that the *opt* scheme always reached the best performance for both encryption time and size. The results can be found in [26].

Secondly, we measured the query evaluation performance for different encryption schemes. Figure 9 shows the result of *NASA* database. The observation is that for the same query, the query processing time on server, the decryption time on client and the query processing time on client decreases in the order of *top*, *sub*, *app*, *opt* scheme. However, the query processing time on server decreases slower than decryption time and query processing time on client, i.e., the improvement of query performance by better encryption scheme is mainly on the client side. Another observatation is that for all results, the query performance of *app* scheme is around 1.1-1.3 times of that by *opt* scheme, i.e., *app* scheme is a reasonable alternative for *opt* scheme. The similiar results are observed from *XMark* database, which are put in [26].

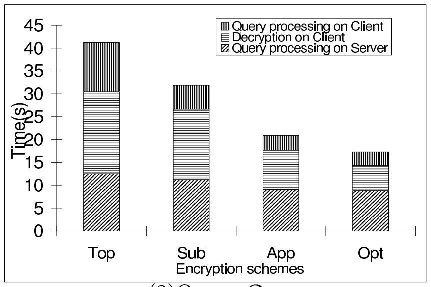
We also measured how much the *app* scheme and *opt* scheme win the *top* and *sub* schemes. We use *saving ratio* to show the result. The *saving ratio* $S_{a/t}$ ($S_{a/s}$) of *app* over *top* (*sub*) is defined as $S_{a/t} = \frac{T_t - T_a}{T_t}$ ($S_{a/s} = \frac{T_s - T_a}{T_s}$), where T_t be the query evaluation time by the *top* scheme, T_a be the time by the *app* scheme, and T_s be the time by the *sub* scheme. We also define $S_{o/t}$ and $S_{o/s}$ in the similiar way. Intuitively, the closer the saving ratio to 1, the better. We showed the results in Figure 10. The figure shows that both *app* and *opt* schemes gets better saving ratios over *top* scheme than the *sub* scheme. It also shows that the saving ratio grows when the output node of query is closer to the leaf(e.g. *opt* scheme reaches the highest ratio 0.64 over *top* scheme and 0.53 over *sub* scheme for query Q_l on *NASA* database). This is because when the output node is closer to the leaf, it is closer to the node that is encrypted by the *opt* and *app* scheme. Consequently the cost of decryption and query processing on client is smaller.

8. CONCLUSIONS

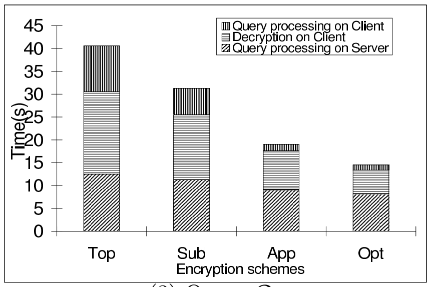
In this paper, we considered the problem of efficient evaluation of XPath queries on encrypted XML databases hosted by an untrusted server. We proposed security constraints as a means for the client to specify her security requirements. We formally defined the attack model and semantics of security of encryption scheme, metadata and query processing. We showed our approach of constructing a secure encryption scheme and showed that finding an optimal secure encryption scheme is NP-hard. We proposed a metadata mechanism (including structural and value indices) at the server side that enables efficient query evaluation. We showed that with presence of the structural and value metadata, for a large set of candidate databases that don't contain any sensitive information, the attacker cannot distinguish them from the original hosted data, neither can he increase the probability of any of them to be the original hosted data. We also proved that an attacker cannot improve his belief probabilities about sensitive structural or value association in-



(1)Query Q_s

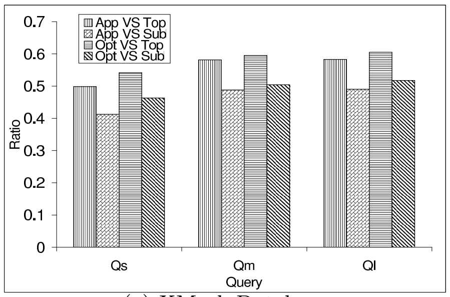


(2)Query Q_m

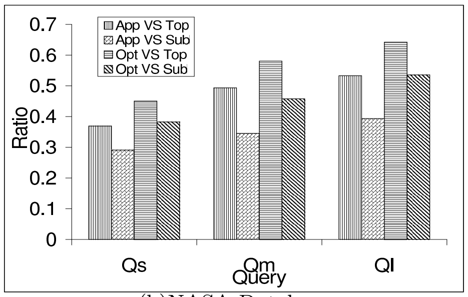


(3) Query Q_l

Figure 9: Query Performance of Various Encryption Schemes, NASA Database, 25MB



(a) XMark Database



(b)NASA Database

Figure 10: App and Opt Schemes VS Top and Sub Schemes

formation about the database, even after observing a series of queries and responses. We complemented out analytical results with a comprehensive set of experiments. Our experiments show that our techniques can delivered the efficient query evaluation while ensure the enforcement of the security constraints.

Possible future work includes: (1) The security achieved comes at the price of increase in data size (e.g., scaling used in OPESS) and the need to decrypt data to answer some queries. Can we characterize the tradeoff between security and efficiency? (2) Our current scheme cannot provide security against an attacker who has the prior knowledge of tag distribution, query workload distribution, or correlations between data values. Extensions for handling these attack modesl are important. (3) Developing a secure encryption scheme for efficiently supporting updates is another important problem.

9. REFERENCES

[1] M. Abadi et al., "Security Analysis of Cryptographically Controlled Access to XML Documents", PODS 2005.

[2] G. Aggrawal et al., "Two can Keep a Secret: A distributed Architecture for Secure Database Services", CIDR 2005.

[3] R. Agrawal et al., "Order Preserving Encryption for Numeric Data", SIGMOD 2004.

[4] S. Al-Khalifa et al., "Structural Joins: A Primitive for Efficient XML Query Pattern Matching", ICDE 2002.

[5] M. Bellare, "Practice-oriented provable-security", ISW 1997.

[6] E. Bertino et al., "Secure and selective dissemination of XML documents", ACM Transactions on Information and System Security, 5(3), Page 290-331, 2002.

[7] R. Brinkman et al., "Efficient Tree Search in Encrypted Data", 2nd International Workshop on Security in Information Systems, April 2004.

[8] Y.Chang, "Single database private information retrieval with logarithmic communication", eprint 2004/036.

[9] B. Chor et al., "Private information retrieval", Proc. of 36th IEEE Symposium on Foundations of Computer Science, 1995.

[10] K.L. Clarkson, "A modification of the greedy algorithm for vertex cover", Information Processing Letters, 16 (1983), pp.23-25.

[11] S. Claude, "Communication theory of secrecy systems."

[12] E. Damiani et al., "Balancing confidentiality and efficiency in untrusted relational DBMSs. Proc. of ACM Conference on Computer and Communications Security, 2003.

[13] D. Denning, "Cryptography and Data Security", Addison-Wesley, 1982.

[14] B. Finance et al., "The Case for Access Control on XML Relationships", CIKM, 2005.

[15] S. Goldwasser et al., "Cryptography: Lecture Notes", pp.83, August 2001.

[16] V.Gowadia et al., "RDF Metadata for XML Access Control", ACM Workshop on Security, 2003.

[17] H. Hacigumus et al., "Executing SQL over Encrypted Data in the Database-Service-Provided Model". SIGMOD 2002.

[18] B. Hore et al., "A Privacy-Preserving Index for Range Queries". VLDB 2004.

[19] M. Kantarcioglu et al., "Security Issues in Querying Encrypted Data", Technical Report, Purdue U., 2004.

[20] E. Kushilevitz et al., "Replication is not needed: Single Database, Computationally-Private Information Retrieval", FOCS97.

[21] P. Lin et al., "Secure and Privacy Preserving Outsourcing of Tree Structured Data", SDM 2004.

[22] U. Maheshwari et al., "How to Build a Trusted Database System on Untrusted Storage", OSDI 2000.

[23] G. Miklau et. al, "Controlling Access to Published Data Using Cryptography", VLDB 2003.

[24] G. Miklau et. al, "A Formal Analysis of Information Disclosure in Data Exchange", SIGMOD 2004.

[25] Motwani et al., "Lecture Notes on Approximation Algorithms", 1992.

[26] H. Wang et al., "Efficient Secure Query Evaluation over Encrypted XML Databases on Unstusted Server: Full Paper", ftp://ftp.cs.ubc.ca/local/laks/papers/xcryp-fullpaper.pdf.