Secure and Privacy Preserving Outsourcing of Tree Structured Data

Ping Lin and K. Selçuk Candan

Department of Computer Sciences and Engineering
Arizona State University
Tempe AZ. 85287, USA.
{ping.lin, candan}@asu.edu
phone:480-727-3611

Abstract. With the increasing use of web services, many new challenges concerning data security are becoming critical. Data or applications can now be outsourced to powerful remote servers, which are able to provide services on behalf of the owners. Unfortunately, such hosts may not always be trustworthy. In [1, 2], we presented a one-server computationally private tree traversal technique, which allows clients to outsource tree-structured data. In this paper, we extend this protocol to prevent a polynomial time server with large memory to use correlations in client queries and in data structures to learn private information about queries and data. We show that, when the proposed techniques are used, computational privacy is achieved even for non-uniformly distributed node accesses that are common in real databases.

Keywords: Search on Encrypted Data, Tree structured data (XML) Security, Private Information Retrieval

1 Introduction

In web and mobile computing, clients usually do not have sufficient computation power or memory and they need remote servers to do the computation or store data for them. Publishing data on remote servers helps improve service availability and system scalability, reducing clients' burden of managing data. With their computation power and large memory, such remote servers are called data stores or oracles. Typically, as the entities different than the data owners, these data stores can not be fully trusted, for they may be malicious and can be driven by their own benefits to make illegal use of information stored on them. Hence data outsourcing introduces security concerns that are different from traditional database service which always assumes that database is honest and it is the illegal users access that should protect the data from. These concerns have to be addressed effectively to convince customers that outsourcing their IT needs is a viable alternative to deploying complex infrastructures locally.

1.1 Problem Statement

Special security concerns with respect to data outsourcing can be categorized into *content privacy* and *access privacy* [12]. Clients with sensitive data (e.g., personal identifiable data) outsourced to untrusted host may require that their data be protected from such data storage oracles. This is defined as *content privacy* [12] and leads to encrypted database research [7], in which sensitive data is encrypted, so the content is hidden from the database. Sometimes not only the data outsourced to a data store, but also queries are of value and a malicious data store can make use of such information for its own benefits. This is defined as *access privacy* [12]. Access privacy leads to private information retrieval [13] research, which studies how to let users retrieve information from database without leaking (even to the server) the identity and the location of the retrieved data item.

Access privacy and content privacy are not independent. If the data has some structure, plain access may reveal this structure, hence impair content privacy. For example, if the data is in the form of an XML tree (as described in the next subsection), without proper methods to protect access privacy, the path along which to find the target data will be revealed, so will be the whole structure of the data tree, which impairs the content privacy. Hence to protect both content privacy and access privacy, we need to hide the structure of the data.

In this paper, we address secure outsourcing of tree structured data, such as XML documents. To be specific, we address hiding of tree-structured data and queries on this data. XML documents [8] have tree-like structures. XML has become a de facto standard for data exchange and representation over the Internet [9]. Some work has been done on selective and authentic untrusted third-party distribution of XML documents [3, 9–11]. The work focuses on access control and authentication of document (i.e.,query result) source and content. With more and more data stored in XML documents, techniques to hide tree structures (the content and structure of XML documents) from untrusted data stores are in great need. In an XML database, a query is often given in the form of tree paths, like XQuery. To hide XML queries and the structure of XML documents, clients need to traverse XML trees in a hidden way. Other frequently used tree-structures include indexes that are often built for convenient access to data. However, most index structures closely reflect the distribution of the data. Thus, in order to hide the data and data distribution from the database, tree structure hiding techniques must be adopted to protect index trees from oracles. Though the techniques we present in this paper can also be used for hiding index structure accesses, here we do not focus on this application. Recent work in privacy-preserving index structures includes [4].

In [1, 2], we proposed a protocol for hiding traversals of trees from oracles. Noticing that existing private information retrieval techniques require either heavy replication of the database onto multiple non-communicating servers or large communication costs or computation costs [13], we provided an one-server *tree-traversal* protocol that provides a balance between the communication/computation cost and security requirements. To protect the client from the

malicious data store, some tasks (such as traversing the tree-structures) are performed interactively. Client responsibilities include encryption and decryption of the data received from the data store during the traversal of the tree. The encryption capability required at client side can be achieved by assistant hardware equipments, such as smartcards that are cheap (generally no more than several dollars) and now commonly used in mobile environments [20]. We analyzed the overhead incurred by proposed technique, including communication cost, encryption/decryption costs, and concurrency overhead. Since [13] has argued that information-theoretical private information retrieval cannot be achieved without a significant communication overhead, our proposed method minimizes those costs in a computational hiding sense.

In this paper, we build on the approaches proposed in [1, 2] to develop a computationally secure protocol for hiding correlated accesses to tree-structured outsourced data. We find that if node accesses are uniformly distributed, the original protocol achieves computational privacy [2]. To ensure computational privacy in face of non-uniformly distributed and correlated node accesses, which actually occur in real scenarios, we propose a systematic way to enhance the preliminary protocol so that from the server's view, node accesses are uniformly distributed.

2 Related Work

Besides the common data encryption methods that hide sensitive data, there are various efforts to hide other kinds of secret information from untrusted servers. Basic methods to protect content privacy include database encryption, where critical data such as credit card number can be encrypted. DBMS suppliers such as Oracle and DB2 have provided encryption functionality. Bertino and Ferrari [10] have studied how to protect sensitive XML data content from different entities by performing differentiated encryption of various portions using multiple keys. Hacigümüs, H. et al. [7] have studied how to execute SQL over encrypted data. Other recent work on querying encrypted data includes [5] and [6].

Sensitive information about data may include users queries about data. Different from traditional database security which deals with preventing, detecting, and deterring improper disclosure or modification of information in databases, private information retrieval (PIR) aims to let users query a database without leaking to the database what data is queried.

The basic idea behind any information theoretic PIR scheme is to replicate the database to several non-communicating servers and pose randomized queries to each server so that from the server's view those queries are independent of the target but the user can reconstruct the target data from query results. The privacy guarantee lies in that even computationally un-bounded server can not tell the difference between any two communications for different targets. [13] showed that if one copy of database is used, the only way to hide the query in the information theoretic sense is to send the whole database to the user. In order to reduce even one bit in communication between the server and the

user, replication of the whole database is required. Hence information theoretic privacy techniques require multiple data copies and cause heavy communication overheads.

In order to achieve practical use, communication cost and the number of replicas need to be reduced. Ambainis proposed a k-server scheme that requires $O(n^{2k-1})$ communication [14] (where n is the database size). This result is further improved to $n^{O(\frac{loglogk}{klogk})}$ by Beimel et al. [15]. This is the best known k-server information theoretic private information retrieval scheme so far. However, to achieve communication that is subpolynomial in the size of the database, more than a constant number of servers are needed [13]. In real world, database replication is not a preferable solution. It may not be possible to prevent servers from communicating with each other. In computationally private information retrieval (CPIR) schemes, therefore, the user's privacy requirement is relaxed so that any two communications for different targets are indistinguishable to any polynomial time server.

CPIR schemes are built on cryptographic assumptions, which enable further reduction of the communication and the number of replicas. If a one-way function exists, then there is a 2-server scheme, such that the communication is $O(n^{\epsilon})$ for any $\epsilon > 0$ [16]. Under the Quadratic Residuosity Assumption, a one-server scheme can be constructed with sub-polynomial communication [17]. Under the ϕ-hiding Assumption, a one-server scheme with a poly-logarithmic communication can be achieved [18]. Based on Paillier cryptosystem which is secure under the Composite Residuosity Assumption, Chang [19] proposed a one-server scheme with logarithmic communication overhead which is the optimal for CPIR. Despite the reduced communication overhead, however, all above CPIR schemes [16–19] suffer from heavy computation cost (linear in size of the database size) at both the client and server sides. Smith et al. [12] employed the secure processor technique to achieve computational privacy by embedding a secure processor at the malicious server, let the clients encrypt their queries and let the secure processor decrypt the queries and read the database to retrieve the targets. Since the whole database should be read into the secure processor to hide the query from the server, cost linear in the database size is still incurred at the server side.

Most existing information theoretic PIR and computationally PIR schemes are built on binary data model, making it hard to be applied to real database. In the next section, we provide an overview of our previous work which enables a one-server CPIR scheme that can be easily applied to real data model and only involves moderate and adjustable computations and communications to gain content privacy and access privacy [1, 2].

3 Background:Private Tree Traversal with Access Redundancy and Node Swapping

In [1], we proposed a preliminary protocol to hide traversal paths on tree structured data. The protocol is a one server protocol. *Content privacy* is guaranteed

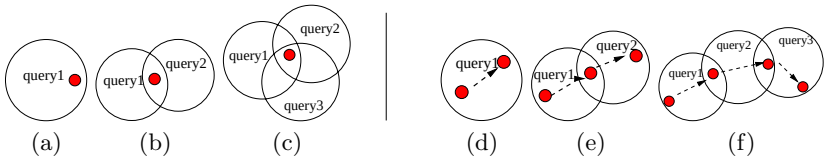

Fig. 1. (a,b,c) Leakage of the position of root node of tree structure as a result of repeated accesses and (d,e,f) node swapping eliminates leakages

through encryption of the tree nodes (data and pointers) before outsourcing, hence the host can not know the data content. *Access Privacy* is achieved by novel *access redundancy* and *node swapping* techniques. The data storage space is divided into units of nodes (called physical nodes) which may contain tree nodes (called data nodes) or have no content (empty nodes) and is organized into a multi-level structure with each level storing a corresponding level of the tree respectively. Whenever a client wants to retrieve a tree node, besides the target node, it asks a set of random nodes from the level. We define the set of nodes the client retrieves in order to get the target as *redundancy set*. Hence if the size of the redundancy set is m, the probability for the server to have a correct guess of the target is $\frac{1}{m}$; the probability for the server to have a correct guess of parent-child relationship is $\frac{1}{m^2}$; if the depth of the tree is l, the probability for the server to find the traversal path is $\frac{1}{m^l}$. The definition of redundancy set can be extended to contain multiple target nodes. If there are k targets in the redundancy set, the probability for the server to have correct guess of the targets is $\frac{1}{C_m^k}$; the probability for the server to have a correct guess of the parent-child relationships is $\frac{1}{(C_m^k)^2 \times k!}$; and the probability for the server to have correct guess of traversal paths is $\frac{1}{(C_m^k)^l \times (k!)^{l-1}}$.

Unfortunately, we find that repeated access for the same target (e.g. the root) may reveal the target, for the target is always in the intersection of the related redundancy sets. Figures 1(a), (b), (c) give an example showing how repeated access for the root may reveal the physical location of the root. In Figure 1, large circles represent redundancy sets and small circles represent the root. In [1, 2] we addressed this problem by requiring that each redundancy set should at least include one randomly selected empty node. After each retrieval the client should swap the target with the empty node, *re-encrypt* the redundancy set using a different key or encryption scheme (which is essential in order to hide the location of target and the empty) and then write the set back into the data storage space at the server. This is called *node swapping*. Node swapping ensures that after each retrieval, the target moves to a random position in the data store, hence making the distribution of data in the data storage space random, i.e., data keep randomly moving as queries are posed and answered. With node swapping, any correct guess of the target is transient and hence the information leaked by intersections is reduced (Figures 1(d), (e), (f)).

Since data nodes are constantly swapped, the parent-child links have to be properly maintained. The physical location of the root is maintained in a fixed

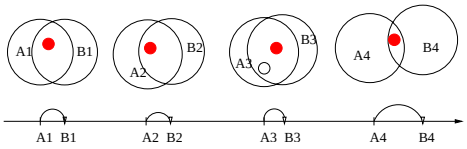

Fig. 2. Intersections may leak identical queries

encrypted special node `snode` [1, 2] which is the entry to the data store. In [1, 2], we developed highly concurrent techniques to maintain parent-child links as children nodes are swapped. Based on this, we have implemented a deadlock free private tree traversal algorithm [1, 2] to enable a client to query the tree by traversing the tree and locating the required data node. The total cost (including communication cost, read/write cost and encryption/decryption cost) of the protocol is a function of the redundancy size m, the total data points num and the node size s, i.e., the maximum number of data points a data node can contain. If c, e/d, r/w denote communication cost function, encryption/decryption cost function, read/write cost function with respect to node size s, the function is $log(\frac{num}{s}) \times m \times (c(s) + e(s) + d(s) + r(s) + w(s))$. This cost is adjustable. Compared with the costs of existing one-server PIR schemes which are linear in the database size num, this cost is moderate (poly-logarithmic in num). This cost function also shows that with m and num set, there exists an optimal node size s to minimize the total cost. Experiment results [1, 2] show that the performance of the protocol is consistent with the theoretical analysis mentioned above and, compared with one-server information theoretic private information retrieval scheme, this cost is small and hence the proposed approach is more practical.

Although this algorithm hides uniformly distributed tree node accesses, when queries are correlated, the server can learn private information about queries and data. In this paper, we present techniques that guarantees computational privacy even for non-uniformly distributed accesses that are common in real databases.

4 Problem Formation: Preventing Information Leakages caused by Intersections of Redundancy Sets

In this paper, we refer to client's retrieval of a single redundancy set as a *call*. A query then can be represented as an ordered set of calls. For instance, a path from root to a leaf would be a sequence of calls from the client to the server.

Supposing that there is a transport layer security mechanism (e.g. anonymous access protocol) that hides the identity of client, the server sees data accesses as a stream of calls from unknown origins. We define a stream of calls the server has observed during certain period of time as a *view*.

Computational privacy requires that any computationally bounded server should not be able to tell the difference between client-server communications for two different *queries*. Given the above query model, we note that the server might be able to infer information by observing the call stream, by observing the

intersections of the redundancy sets of the corresponding calls in each query. For instance, if there are two queries, Query A and Query B, that are the same and consecutive, i.e., without intermediate queries' interfering, their calls for every node on the path will be intersecting, hence providing hints to the server that two identical queries have happened. Figure 2 depicts the phenomenon, with A_i and B_i denoting A and B's corresponding calls respectively.

Correct identification of identical queries not only increases the risk of leaking the traversal path (although such leakage is transient), it also leaks the information as to how frequent queries are posed. If the server happens to know the query distribution in advance, i.e., how often every query occurs, and if there are distinguished variation among query frequencies, the server can identify the queries that have been posed.

Our goal in this paper is, therefore, provide computational privacy in the presence of correlated queries:

1. **Hiding distribution of calls**: for any two different queries Q_1 and Q_2 posed in the view, the distribution of their sequences of calls are indistinguishable in polynomial-time.
2. **Hiding intersections**: for any two queries Q_1 and Q_2 in the view, it is hard to tell if they are identical or not by observing their sequences of calls.

4.1 Privacy Guarantees for Uniformly Distributed Node Accesses

In [2], we showed that if for every level at the tree, node accesses are uniformly distributed(i.e., for every level, tree nodes on this level are accessed by clients at the same probability), and if the database is randomly initialized, then the protocol has already achieved the required computational privacy.

Hiding distribution of calls Our proof is based on the following proposition and corollary.

Proposition 1. *If the data storage space is randomly initialized and data nodes are uniformly accessed in each level of a tree, then data nodes are always uniformly distributed in each level of the data storage space.*

Corollary 1. *If the data store is randomly initialized and node accesses are uniformly distributed for every level of a tree, then redundancy sets posed are also uniformly distributed for any level of the data store.*

If the data storage space is randomly initialized and node accesses are uniformly distributed in every level, then data nodes will always be uniformly distributed in each level of the data storage space and calls for that level will also be uniformly distributed. So for two different queries, if their traversal path lengths are equal, the distribution of their sequences of calls are identical, hence indistinguishable in polynomial-time; if their traversal path lengths are not equal, clients can execute dummy calls at deeper levels to always make the same number of calls. Details of the proofs are omitted here due to space constraints.

Hiding intersections As to the second privacy requirement, the only hint server has for identifying identical queries from views are intersections between calls: if two identical queries are posed consecutively without any interfering calls, their corresponding calls will intersect. However, if node accesses are uniformly distributed, even if the server happens to know that at some time t, there is a call belonging to Q_1 such that one of its target is the data node nd, and the server observes that some time later there is a call belonging to Q_2 and the call intersects with Q_1's call at nd, it can not judge whether the later call is also targeting at nd or not, hence having no hint whether Q_2 is identical to Q_1. If, m denotes the size of the redundancy sets, n denotes the number of data nodes in tree level where nd is located and N denotes the total number of nodes in the data storage space of the level, k denotes the number of target nodes per call, then

- the call of Q_2 may intersect with the call of Q_1 at nd because it also sets target on nd. This probability $P_1 = \frac{C_{n-1}^{k-1}}{C_n^k}$
- the call of Q_2 may intersect with the call of Q_1 at nd because it selects nd as one of its random nodes. This probability $P_2 = \frac{C_{n-1}^{k}}{C_n^k} \times \frac{C_{N-2k-1}^{m-2k-1}}{C_{N-2k}^{m-2k}}$.

If there are enough empty nodes in the level to expand the data storage space, e.g., n empty nodes, and if m is large enough, e.g., at least $4k$, P_2 will be no less than P_1. Hence if the data storage space for the tree is expanded linearly and calls contain enough random nodes (only linear redundancy), from the intersections, the server can not tell whether two queries are identical or their corresponding redundancy sets just happen to intersect and the probability for the polynomial time server to have correct judgement whether a view contain identical queries or not cannot be non-negligibly better than random guessing.

4.2 Naive Approaches for Providing Privacy Guarantees for Non-uniformly Distributed Node Accesses

In real situations, queries and node accesses are not always uniformly distributed. If there are big variations among node access distributions, higher frequencies of intersections will be more likely a sign of repeated occurrence of high frequency queries than just random intersections. Thus intersections may leak information about occurrence of identical high frequency queries, hence enabling the server to deduce query frequencies. If the server knows in advance the tree structure and how often each tree node is accessed, such information leakage increases the risk of leaking the queries that have been posed.

These attacks by the server would rely on the intersection property mentioned above. Intuitively, such an attack by the server can be prevented by ensuring that intersections do not reveal much information. This can be achieved by modifying the client/server protocols such that the redundant sets intersect at multiple nodes as well as by inserting appropriate dummy/interfering calls.

Intuitively, dummy interfering calls add ambiguity and reduce the probability with which the server can identify the calls that correspond to identical queries.

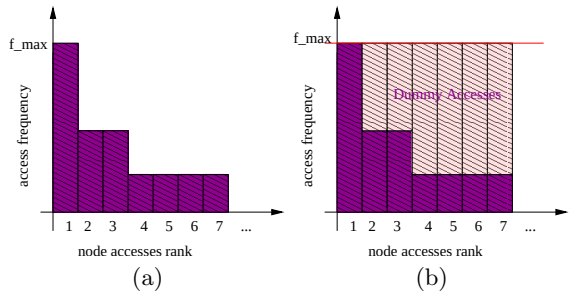

Fig. 3. Naive uniform approach: (a) original node accesses distribution and (b) adding dummy node accesses enables uniform distribution

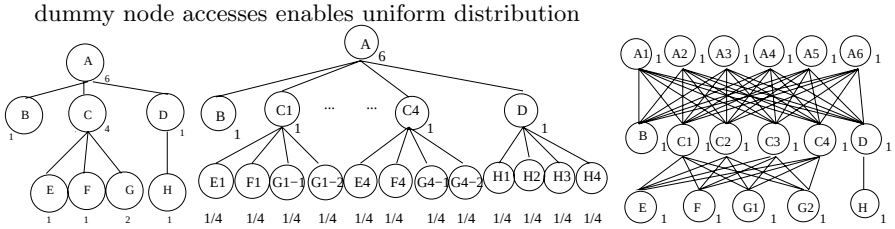

Fig. 4. Replication: (a) original tree and node access frequencies, (b) and (c) two different ways for replicating nodes

Note that in order to provide efficient and provable security, the process of introducing dummy calls have to follow a strategy which randomizes the intersections with minimal overhead. In this section, we first discuss three naive approaches to minimize information leakage by intersections. In the next section, we present a systematic and efficient way to choose proper dummy interfering calls.

Naive Uniform Approach: A straight forward approach to make tree nodes to be uniformly accessed is to generate enough dummy accesses for low frequently accessed tree nodes so that all tree nodes are accessed at the same frequency with the most often accessed one. Figure 3 depicts the approach. In Figure 3, the X axis denotes node access rank (node ranked 1 is mostly accessed) and the Y axis denotes the number of accesses for each tree node during unit time interval. Figure 3(a) shows the original node accesses distribution and Figure 3(b) shows the uniform distribution after enough dummy accesses (depicted by grey columns) are generated. The cost of this approach is $\sum_i (f_{max} - f_i)$ (f_{max} is the maximum access frequency and f_i is the access frequency for node ranked i). In general, this naive approach leads to a large number of dummy accesses (for example, this would require exponential dummy calls for trees where the root is accessed once for each leaf), which would cause a heavy cost for the system.

Replication Approach: The idea of this naive approach is to replicate nodes that are accessed with higher frequencies into multiple copies so that each copy is accessed at the same frequency with the lowest frequency node. There are two ways to replicate nodes. The first is to make node accesses uniformly distributed at every level by applying replication repeatedly from the top to

bottom, making access frequency uniform at every level. Figure 4(b) gives an example how to replicate the tree shown in Figure 4(a) in a way that at every level nodes are uniformly accessed. In this figure, the number associated with a tree node denotes its access frequency. Note that with this approach, the nodes of different levels have different access frequencies. Therefore level information is leaked. The second approach prevents leakages of level information by applying replication to all nodes of the tree, making them accessed at the same frequency with the least accessed tree node. Figure 4(c) depicts the tree structure after applying the second replication approach to the tree in Figure 4(a). Replication is simple and does not need dummy node accesses, hence is query efficient. However, since every high frequency nodes are replicated according to the lowest frequency, huge disk space is required to replicate the tree in both of these replication approaches. For the first approach, a space of size exponential in the height of the node is required to get just one extra copy of this single node. The second replication approach needs less replication (exponential in the tree depth). However, since every copy of the parent should maintain addresses of its children, when a child is swapped, all parents have to be updated to refer to the new address of the child. This will increase the access frequency of the copies of the parent, making the unified access non-uniform. Furthermore, for both approaches, if the content of a node is updated, all copies of the node should also be updated, making update a costly task and changing the uniform accesses of nodes. This problem is inherent with replication.

Clique Approach: Another naive approach to minimize information leaked by intersections is to generate calls for queries such that any two queries' corresponding calls intersect. If we use a graph to represent a view so that every vertex depicts a query and every edge depicts the intersection between two queries, with this approach, the view forms a clique. The idea behind this approach is to make the probability for non-identical queries to intersect equal the probability for identical queries to intersect (both equal 1) so that there is no way for the server to find identical queries by observing intersections. This approach is also too costly. It requires a large redundancy set, for a call should contain more than half of the data storage nodes from its level to be able to intersect with corresponding calls of all other queries. Actually this cost is comparable to sending the whole database to the client.

5 Proposed Approach: Clustering Node Accesses into Uniform Chains

From the naive approaches described above, we observe that challenges associated with hiding information that may be leaked by intersections are (a) making node accesses seem uniform, (b) without introducing many dummy node accesses, (c) without large redundancy set sizes, and (d) without requiring large storage space. In this section, we propose a *frequency clustering and chain-merging* approach to minimize information leaked by intersections.

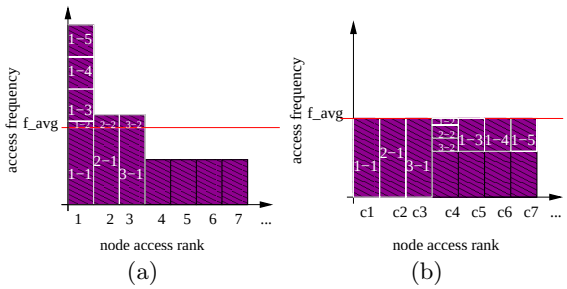

Fig. 5. Frequency clustering: (a) Splitting high frequency accesses and (b) access frequencies are uniformly distributed after merging

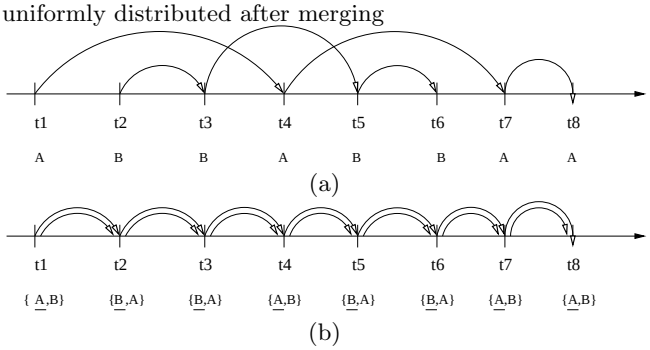

Fig. 6. Chain merging: (a) Two access chains for node A, B respectively (each edge denotes an intersection in redundancy sets as described in Section 4); (b) A's access chain and B's access chain are merged together into one single chain(each consecutive pair of calls has two intersections)

Let D denote the total number of tree nodes. The idea behind the approach is as follows: if we can fractionally cluster D nodes into D equivalent classes such that the total access frequency for each equivalent class is almost equal to the average access frequency, and if we can merge node accesses in each equivalent class together, the server's view of node accesses will become uniformly distributed. This is depicted in Figure 5. Accesses to high frequency nodes are split(Figure 5a). For example, the node ranked first is split into segments 1-1, 1-2, 1-3, 1-4, 1-5. Then as shown in Figure 5(b), accesses to high frequency nodes are clustered with accesses to low frequency nodes. For example, segment 1-5 is clustered with 7 to form cluster c7, segments 1-2, 2-2, 3-2 are clustered with 4 to form cluster c4. The total frequency for each cluster adds up to f_{avg}. The splitting and clustering depicted in Figures 5 exhibits the process of fractional clustering of node accesses. For example, the first cluster $c7$ in Figure 5(b) depicts that with some clustering probability (the ratio between the frequency of the segment 1-5 and the total frequency of the first ranked node access), the first ranked node access is clustered with the seventh ranked node access.

The challenge, however, is to make accesses for different nodes in each cluster look like the same. Figure 6(a) shows two different chains formed by accesses of nodes A and B that are clustered together. The shape of the individual access

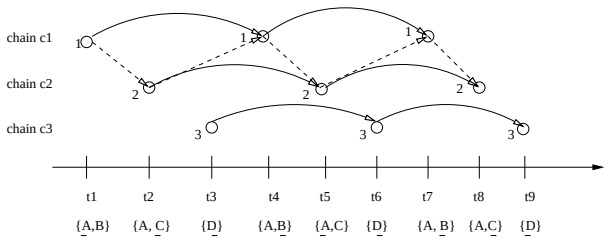

Fig. 7. Crossing among chains are denoted with dashed arrows

chains can leak information about access frequency to the server. Unless the node accesses in each cluster resembles each other, the server will observe individual accesses. To address this challenge, we propose *chain merging* (Figure 6). By merging chains, we mean that whenever a node in a given cluster is accessed by a client, then the client also accesses other nodes in the cluster together with this node in a single redundancy set. Figure 6(b) shows when accesses of A and B are merged. The access *chains* of A and B are merged together into one single chain and the resulting chain is uniform looking.

With this approach, the malicious server will observe access chains, each of which occurs with approximately f_{avg} frequency. Hence access frequency distributions will not be leaked by intersections (except the average access frequency). To achieve chain merging, however, there are three further challenges that have to be addressed:

- *Maximum cluster size:* The maximum number of node accesses per cluster should be constrained by a subpolynomial; i.e., we should avoid clusters that require redundancy sets the sizes of which are polynomial in the database size. Otherwise, merging them into one would require large redundancy, which would cause heavy communication overheads. This is discussed in Section 5.1.
- *Cluster directory size and access frequency:* We need a storage and search efficient directory structure to maintain the clustering information so that whenever a client needs to access a target node, it can quickly find the corresponding clusters and identify all nodes in those clusters. This directory will be maintained in the server in encrypted manner. Since each node access is preceded by a directory lookup, the size of the directory structure as well as its access pattern should not leak further information. This is discussed in Section 5.2
- *Chain crossings:* a node may belong to more than one cluster, making multiple chains *cross* with each other. Figure 7 gives an example. In this example, there are three clusters: $c_1 = \{A, B\}$, $c_2 = \{A, C\}$, $c_3 = \{D\}$. The axis depicts time line and the small circle that above time t_i represents the cluster that is accessed at time t_i. Since A is in clusters c_1 and c_2, we can see that besides the intersections which form a chain for each cluster (depicted by solid arrows), there are some crossing intersections (depicted by dashed arrows) between chains c_1 and c_2. However, since c_3 shares no elements with

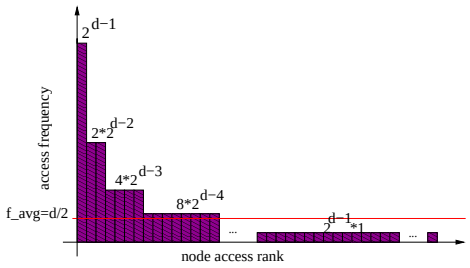

Fig. 8. Node access distribution example: a balanced tree with uniform leaf accesses

c_1 and c_3, there is no crossing associated with c_3. Such *crossing*s between chains should have uniform pattern, otherwise a powerful server may infer extra information from their distribution. For this example, the server may deduce that cluster c_1 and cluster c_2 share some high frequency element. This is discussed in Section 5.3

In the rest of the paper, we address each of these challenges.

5.1 Minimizing the Maximum Cluster Size

In this section, we show how to restrict the maximum number of accesses per cluster to 3 (except for boundary clusters). The procedure starts from the highest and lowest frequency nodes and progressively moves towards medium frequency nodes. We first split the highest frequency node access into segments of enough volume to fill the gap between the lowest frequency and the desired average frequency. Then, we use the highest frequency node to fill as many of the lowest frequency node accesses as possible. In most cases this scheme will lead to clusters of size 2. In some cases, it may be that the segment of the highest frequency node access can not fill the gap between the low frequency and the average frequency and need to cluster with a segment from the next highest frequency, resulting in a cluster size of 3. In this way, we can continue using segments of the highest available frequencies to fill the unfilled lowest frequencies, getting clusters of size 2 or 3.

An extreme exception may happen at the final stage of the above process when the highest available frequencies are just above f_{avg}, and the unfilled lowest frequency is below f_{avg}, and the rank of highest available frequencies are immediately before the rank of unfilled lowest frequency. Let i denote the rank of the final unfilled lowest frequency node. Because the available highest frequencies f_{i-j}, f_{i-j+1}... f_{i-1} are just above f_{avg}, and f_i is pretty much below f_{avg}, hence a number of node accesses (ranked $i-j$, $i-j+1$, ...$i-1$) are needed to fill the gap between f_i and f_{avg}. In the case of trees accesses (shown by Figure 8) we can show that the number of nodes in the extreme cluster is bounded by the depth, d, of the tree. Suppose there are d distinct access frequencies (one for each level) and there are 2^{i-1} different i^{th} ranked node accesses, each with access frequency 2^{d-i}. This extreme case happens if f_{i-1} is just above $\frac{d}{2}$, i.e., $f_{i-1} = \frac{d}{2}+1$

if d is even, or $f_{i-1} = \frac{d+1}{2}$ if d is odd. In both cases, the final boundary clusters have a size of $O(d)$. In conclusion, if D denotes the total number of nodes, this way of clustering generates $O(D)$ clusters of size 2, at most d boundary clusters of size 3, final boundary clusters of size $O(d)$. The advantage of this is that the number of chains that has to be merged and the required cluster size are both small.

5.2 Implementing a Storage and Search Efficient Cluster Directory

Directory information of clusters is available in two places: **(a)** a cluster table C maintains which nodes are in a given cluster and what their frequency shares are; and **(b)** for each child node in the tree, its parent keeps the list of clusters the child belongs to and the corresponding clustering probabilities (determined by the ratio of the child's frequency shares and its total access frequency) for the child. The cluster table C is encrypted and stored in a fixed address in the server space (this is called `snode` in $[1, 2]$). Entry i of the cluster table C records the identifiers of all nodes the accesses of which are clustered into cluster c_i. The tree structure is also encrypted (Section 3).

While traversing from a parent node to a child node, a client first identifies which clusters the child node belongs to and picks one of those clusters using the associated clustering probabilities. Figure 9 provides an example. In this figure, the node A is fractionally clustered into 4 different clusters: $\{A, B\}$, $\{A, C\}$ $\{A, D\}$, and $\{A, E\}$. Each time A is accessed from its parent, one of these 4 clusters will be chosen based on the clustering probabilities stored in the pointer. Therefore, the resulting cluster access pattern can be represented as a random walk graph shown in Figure 9(a), whose vertices (i.e, clusters) have equal number of visits (i.e., uniform access distribution). The weights associated with arrows are traversal probabilities for A that are calculated from A's clustering probabilities. For example, P_{12} denotes the probability with which the next access of A is clustered into c_2 if the current access of A is clustered into c_1. Once the cluster to be used is identified, the client uses the cluster table C to find other nodes in the chosen cluster and requests all the nodes in the cluster in a single redundancy set. Since the cluster table and pointers are encrypted, the only information the server observes are the *sizes* of the entries in the directory and their *access frequencies*. We can hide entry sizes from the server by extending all of them to the maximum cluster size. Since chains are uniformly accessed, entries of C are uniformly accessed, giving no extra information per access except (possibly) the random identifier of the cluster for which we do not care.

The cluster table search cost is $O(1)$: the client only needs to access one entry of C per access and each entry has a constant size (as discussed above). The storage cost of the cluster table is proportional to the size of the database. Since the cluster table maintains pure node identifiers and generally the size of a node is much greater than the size of a node identifier, the cluster table is small compared to the size of the database. However, unless properly designed, the storage cost for pointers can be prohibitively expensive: each parent stores all the clusters each child belongs to and, since (in the worst case) a node can

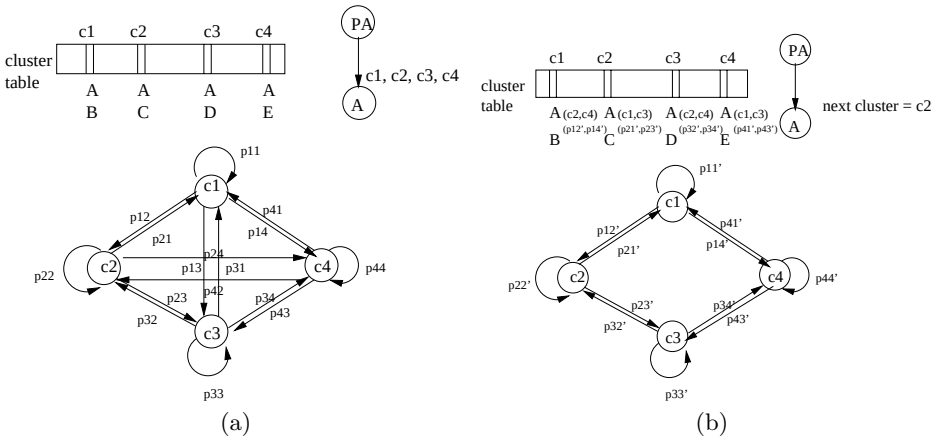

Fig. 9. Reducing pointer costs: (a) In the original structure, the pointer has to maintain all clusters of node A; (b) in the new structure, the pointer only maintains the *next cluster*.

be clustered into D (the size of the database) many clusters, the pointers could become prohibitively large.

To prevent this, instead of storing all the clusters of a given child, the parent should store only a small (constant) number of clusters of the child. This would ensure that the storage requirement is small; however since not all clusters are available, the clustering probabilities for the child would be altered, destroying the uniform access distribution of the cluster chains. In order to prevent the access distribution of the chains diverging from uniform, we need to readjust traversal probabilities and modify the cluster table to let it contain more information.

Figure 9(b) shows how to reduce the storage cost for pointers from P_A to A. In this example, we first reduce the number of cluster neighbors from 3 to 2. To achieve this, we need to recompute the random walk traversal probabilities. The probabilities associated with the new random walk graph are computed so that the clustering probabilities of A remain the same and the access frequency of each cluster is kept uniform. The cluster table C is modified so that: each entry of cluster reflects the two possible *next clusters* for A and their corresponding traversal probabilities, based on the new random walk graph. Then, the parent-child pointer from P_A to A is modified such that, instead of maintaining all clusters that A belongs to, it only maintains the *next cluster* that will be used when A is accessed. Based on the pointers and traversal probabilities stored in the corresponding entry of the cluster table, the value of *next cluster* will be updated after each access.

5.3 Eliminating the Chain Crossing Problem

To address the chain crossing problem, we present two solutions. For the first solution, we borrow the idea from replication approach presented in Section 4.2:

Each node is replicated according to its contribution to its clusters. For each corresponding cluster, the node has a replica. When the node needs to be accessed, one of the replicas is chosen based on how the nodes accesses are distributed among its clusters (i.e., the clustering probabilities). We call this solution *merge-replication*. Since copies of a node are independently accessed, crosses among the chains that are caused by sharing of the node is removed. Since the maximum size of the clusters is small, the amount of total replication is also small. When a child is swapped and the references to it needs to be updated, the client needs to update the physical address *only* in the corresponding entry in the cluster table. However, this solution is restricted to read only applications for update is inherently costly for replication.

If we do not use replication, the crossings will exist and be visible to the server. Therefore, in the second solution, we embed the existing chain crossings into dummy chain crossings that are uniformly distributed among the existing chains. Since, as described in the previous subsection, the number of crossings per cluster can be limited through random-walk based readjustment of the traversal probabilities, the amount of dummy crossings per cluster are small. Details of this process are omitted.

6 Conclusion

In this paper, we build on the protocol we presented in [1, 2] to develop a protocol that enable secure outsourcing of tree structured data and hides correlations in tree traversals from the untrusted server in computational privacy sense. The early protocol [2] ensured privacy when accesses were uniformly distributed. To ensure computational privacy in face of non-uniformly distributed node access which actually occur in real scenarios, in this paper, we presented a systematic way to enhance this protocol so that from the server's view, node accesses are uniformly distributed. Since a lot of data, such as XML, has tree-like structures and queries can be expressed as traversal paths on these trees, this protocol can be utilized for secure outsourcing of XML documents. Compared with existing private information retrieval techniques [13, 22], our protocol does not need replication of databases and it requires less communication, and is thus practical.

References

1. Lin P., Candan K.S.(2004) Hiding traversal of tree structured data from untrusted data stores. Proc. of the 2nd Inernational Workshop on Security In Informtion Systems, WOSIS 2004. pp. 314–323.
2. Lin P., Candan K.S. (2004) Ensuring privacy of tree structured data and queries from untrusted data stores. *Information System Security*, May/June 2004. pp. 22–38.
3. Yang X., Li C. (2004) Secure XML publishing without information leakage in the presence of data inference. VLDB 2004.
4. Hore B., Mehrotra S., Tsudik G. (2004) A privacy-preserving index for range queries. VLDB 2004.

5. Song, D. X., Wagner, D., Perrig, A. (2000) Practical Techniques for Searches on Encrypted Data. IEEE Symposium on Security and Privacy 2000. pp 44–55.
6. Damiani, E., Vimercati, S. D. C., Jajodia, S., Paraboschi, S., Samarati, P.(2003) Balancing confidentiality and efficiency in untrusted relational DBMSs. Proc. of ACM Conference on Computer and Communications Security, 2003. pp 93–102.
7. Hacigümüs, H., Iyer, B.R., Li, C., Mehrotra, S.(2002) Executing SQL over encrypted data in the database-service-provider model. Proc. of ACM SIGMOD International Conference on Management of Data. Madison, Wisconsin, USA, June 3-6, 2002. pp. 216–227.
8. Paparizos, S., Al-Khalifa, S., Chapman, A., Jagadish, H. V., Lakshmanan, L. V. S., Nierman, A., Patel, J. M., Srivastava, D., Wiwatwattana, N., Wu, Y., Yu, C.(2003) TIMBER: A Native XML Database for Querying XML. Proc. of ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003. pp. 672.
9. Bertino, E. Carminati, B., Ferrari, E., Thuraisinggam, B. M., Gupta,A.(2002) Selective and authentic third-party distribution of XML documents. MIT Sloan Working Paper No. 4343-02.
10. Bertino, E., Ferrari, E.(2002) Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security*, 5(3). pp. 290–331.
11. Damiani, E., di Vimercati, S. D. C., Paraboschi, S., Samarati, P. (2000). Securing XML documents. Proc. of the 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000. pp. 121–135.
12. Smith, S. W., Safford, D.(2001) Practical server privacy with secure coprocessors. *IBM Systems Journal*, Vol. 40, No. 3. pp. 683–695.
13. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.(1995) Private information retrieval. Proc. of 36th IEEE Symposium on Foundations of Computer Science. Milwaukee, Wisconsin, USA, October 23-25, 1995. pp. 41–50.
14. Ambainis, A.(1997) Upper bound on the communication complexity of private information retrieval. Proc. of ICALP 1997. Bologna, Italy, 7-11 July 1997. pp. 401-407.
15. Beimel, A., Ishai, Y., Kushilevitz, E., and Raymond, J. F.(2002) Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. Proc. of 43rd IEEE Symposium on Foundations of Computer Science. Vancouver, BC, Canada, Nov 16-19 2002. pp. 261–270.
16. Chor, B., Gilboa, N.(1997) Computationally private information retrieval. Proc. of the 29th Annual ACM Symposium on the Theory of Computing. El Paso, Texas, USA, May 4-6, 1997. pp. 304–313.
17. Kushilevitz, E., Ostrovsky, R. (1997) Relication is not needed: single database. Computationally-private information retrieval. Proc. of the 38th IEEE Symposium on Foundations of Computer Science. Miami Beach, Florida, USA, Otc 19-22, 1997. pp. 365–373.
18. Beimel, A., Ishai,Y., Kushilevitz, E., Marlkin, T.(1999) One way functions are essential for single-server private information retrieval. Proc. of the 31st Annual ACM Symposium on Theory of Computing. Atlanta, Georgia, USA. May 1-4, 1999. pp. 89–98.
19. Chang Y.C.(2004) Single database private information retrieval with logarithmic communication. eprint 2004/036.
20. Bouganim, L., Pucheral, P.(2002) Chip-secured data access: confidencial data on untrusted servers. Proc. of 28th VLDB, Hongkong, China, 2002. pp. 131–142.
21. Bayer, R., Schkolnich, M.(1977) Concurrency of operations on B-trees. *Acta Informatica*, Vol. 9. pp. 1–21.

22. Chor, B., Gilboa, N., Naor, M.(1997) Private information retrieval by keywords. Technical Report TR CS0917. Technion Israel, 1997.